

Dossier de Projet

Site e-commerce LaFleur

Titre de Développeur Web

Web Mobile

Titre RNCP de Niveau V

Par Antoine Verlyck - 2023

Table des matières

I - Liste des compétence.....	4
II - Résumé du projet.....	5
III - Cahier des charges.....	5
1. Besoins du client.....	5
2. Spécification fonctionnelles.....	5
Boutique en ligne.....	5
Panneau administrateur.....	6
Blog.....	6
IV - Spécification techniques.....	6
1. Général.....	6
2. Panneau administrateur.....	6
3. Blog.....	6
4. Boutique en ligne.....	6
5. Sécurité.....	7
V - Réalisation.....	7
1. Gestion de projet.....	7
2. Conception.....	9
Base de donnée.....	9
Conception graphique.....	10
3. Développement.....	12
Langages et technologies utilisées.....	13
Langage PHP.....	13
Langage HTML.....	13
Langage CSS.....	14
Langage JavaScript.....	15
Framework Laravel.....	16
Panneau administrateur.....	17
Boutique.....	24
Blog.....	31
4. Sécurité.....	32
VI - Tests.....	34
1. Tests unitaire.....	34
2. Validation W3C et test de performance.....	35
VII - Veilles technologiques.....	36
1. Tailwind.....	36
2. Select2 et JQuery.....	36
VIII - Documentation anglophone.....	37

I - Liste des compétence

Activité 1

Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application
- Réaliser une interface utilisateur web statique et adaptable
- Développer une interface utilisateur web dynamique
- Réaliser une interface utilisateur avec une solution de gestion de contenu ou e-commerce

Activité 2

Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données
- Développer les composants d'accès aux données
- Développer la partie back-end d'une application web ou web mobile
- Élaborer et mettre en œuvre des composants dans une application de gestion de contenu ou e-commerce

II - Résumé du projet

Les fleuristes, Sabine et Guillaume Cholet, tenant la boutique « Lafleur » à Lourmarin souhaitent ouvrir un site. Le projet est constitué de 3 plateformes, une boutique en ligne, un panneau administrateur et un blog.

La boutique en ligne devra leur permettre de vendre des produits, ceux-ci seront livrables aux villages alentours. Les produits en rupture de stock resteront visible en ligne mais non commandables. Pour une commande inférieure à 50 euros, des frais de livraison de 2,99 euros seront appliqués. Si l'utilisateur voulant commander habite trop loin, il devra être informé que sa ville n'est pas desservie. Les livraisons seront programmées pour le lendemain mais l'utilisateur pourra choisir une autre date. Les utilisateurs pourront filtrer leur recherche de produit par catégorie ou par couleur. Une loterie sera mise en place lors d'évènement permettant aux utilisateurs d'obtenir des cadeaux pour chaque commande passé.

Le côté administrateur sera accessible uniquement par les gérants. Ceux-ci pourront ajouter, modifier ou supprimer des fleurs ou produits. Ils pourront consulter les commandes et modifier l'état de celles-ci. Un message les préviendra si un stock de fleur est bientôt écoulé.

Le blog contiendra des articles sur les fleurs accompagnés d'images. Le blog ainsi que la boutique en ligne devront avoir des liens permettant de naviguer de l'un à l'autre facilement.

III - Cahier des charges

1. Besoins du client

Les clients demandent un site permettant aux utilisateurs de passer des commandes en livraison, une vue administrative concernant la gestion du stock, les articles en ventes et les commandes. Ils veulent également un blog afin d'y publier des articles sur les fleurs de leur région. Lors d'évènements particuliers, les gérants souhaitent une loterie permettant aux clients de gagner un cadeau pour chaque commande effectué pendant cet évènement.

2. Spécifications fonctionnelles

Boutique en ligne

- Des frais de livraison de 2,99 € seront appliqués pour chaque commande inférieur a 50 €.
Au-delà de 50 € les livraisons seront gratuites.
- Les livraisons seront limitées à Lourmarin et ses alentours.
- Les articles en rupture de stock resteront visible mais non commandables.
- Un espace client permettant l'inscription / connexion des utilisateurs.
- Si le client habite dans un village non livrable un message devra le prévenir.
- La livraison est prévu pour le lendemain de la commande mais le client peut choisir une date de livraison différente.
- Les utilisateurs pourront filtrer les articles de la boutique par catégorie, défini par le client, ou par couleur.
- La loterie se présentera sous forme d'une machine à sous lors de la validation d'une commande.

Panneau administrateur

- Le côté administrateur sera accessible uniquement par les gérants.
- Possibilité d'ajouter, modifier et supprimer des fleurs et produits.
- Les gestionnaires pourront ajouter les quantités de fleurs entrant au stock.
- Un message apparaîtra si une fleur arrive bientôt en rupture de stock
- Les commandes pourront être consultées et leurs états (paiement, livraison) pourront être modifiés.

Blog

- Le blog devra mettre en avant la région des clients et la nature en général.
- Il contiendra des articles sur la flore de la région, en particulier les fleurs, tout en contenant beaucoup d'images.
- Des liens seront présent depuis le blog vers la boutique et inversement.
- Des annonces publicitaires seront présente pour chaque évènement particulier.

IV - Spécifications techniques

1. Général

- Développer sous *Windows 10* et *Debian 11*
- *Visual Studio Code* – IDE, Environnement de développement
- *Git* – Logiciel de gestion de versions décentralisé, utilisé avec *GitHub*.
- *Phpmyadmin* – application web de Système de Gestion de Base de Données relationnelle

2. Panneau administrateur

- *Laravel* – Framework PHP
- *Tailwind* – design et responsive du panneau administrateur
- *Select2* – librairie *JQuery*

3. Blog

- *WordPress* – CMS, Système de gestion de contenu

4. Boutique en ligne

- *HTML 5* – structure statique et sémantique
- *CSS 3* – feuille de style, design et responsive
- *JavaScript* – requête *AJAX*
- *POO* – Programmation Orienté Objet
- *PHP* – Composant d'accès aux données (*PDO*, *PHP Data Object*) et partie back-end
- Architecture *MVC* – Modèle Vue Contrôleur

- *Figma* – Outil de design pour créer la maquette et le storyboard
- *MySQL Workbench* – création du *MLD* (Modèle Logique des Données) de la *BDD*
- *Canva* – Outil de design pour la création du logo

5. Sécurité

- *MySQL* – requête préparés afin de réduire le risque d'injection SQL
- *CSRF* – Cross-Site Request Forgery

V - Réalisation

1. Gestion de projet

Pour ce projet de développement web, j'ai choisi d'utiliser Git avec GitHub. Travaillant seul sur ce projet, avec un PC fixe chez moi et un PC portable à distance, Git m'a été très utile pour récupérer les différentes versions du projet sur ces 2 PC. Utilisant GitHub depuis le début de ma formation, j'ai choisi de continuer sur cette plateforme en ligne. Git est un outil indispensable pour tout projet de développement informatique.

Celui-ci présente de nombreux avantages :

- Versionner le projet, c-à-d garder un historique du projet à chaque sauvegarde, permettant de revenir à une version plus ancienne du projet en cas de problème majeur.
- Travail collaboratif par la création de branches. Git offre la possibilité de faire la copie d'une version du projet afin d'y ajouter une fonctionnalité puis de fusionner cet ajout à la version principale. Ce qui permet à chaque développeur d'une équipe de travailler sur sa fonctionnalité sans gêner ni être gêné par ses équipiers.
- Ce système de branches est aussi utile afin de garder plusieurs versions principales du projet, en gardant une version en développement et une autre en production, le projet sera toujours disponible pour les utilisateurs grâce à la branche de production et sur l'autre branche l'équipe de développeur aura l'occasion de tester toutes les fonctionnalités ajoutées par l'équipe afin de veiller au bon fonctionnement du projet lors de la mise en production.

Voici quelques commandes importantes de Git :

- ➔ `git init` sert à initialiser git dans un projet, cette commande est à effectuer à la racine du projet.
- ➔ `git add -option`, cette commande permet d'ajouter les fichiers modifiés, ajoutés ou supprimés dans une liste qui pourra par la suite être versionné et envoyé sur la plateforme en ligne.
- ➔ `git commit -m « message »` servira à créer la version du projet en fonction des fichiers inclus dans la liste avec `git add`. Cette commande est obligatoirement accompagnée d'un message destiné à décrire de façon succincte les changements/ajouts effectués.
- ➔ `git push` sera utilisé afin d'envoyer le ou les commits effectué(s) sur la version en ligne, on appelle « remote » la/les version(s) du projet sur la plateforme en ligne.
- ➔ `git pull` est la commande permettant de télécharger les commits envoyés sur la plateforme puis de mettre à jour le projet localement.

- `git clone -adresse_web_du_projet` sert à télécharger un projet d'un serveur distant en local lorsque celui n'est pas déjà présent sur la machine.
- `git branch -nom_branch`, cette commande permet de créer une nouvelle branche au projet, habituellement il y a toujours 2 branches principales, main et dev, main étant la branch de production et dev la branch de test. Les autres branch créées seront destinées à développer des fonctionnalités précise dans le but d'être fusionner avec la branch dev et lorsque les tests sont validé, dev sera fusionné avec la branch main.
- `git checkout -nom_branch` servira à se déplacer dans la branch voulue.
- `git merge` est la commande utilisée pour fusionner deux branch ensemble.

```
commit 94124713befd550b443373953d4de475966575aa
Author: Antoine Verlyck <antoine.verlyck@gmail.com>
Date: Tue May 2 08:43:43 2023 +0200

    ajout fichier js loterie

commit 574793de10a7c9224879d2a0bc39284cfc904e66
Author: Antoine Verlyck <antoine.verlyck@gmail.com>
Date: Sun Apr 30 17:19:36 2023 +0200

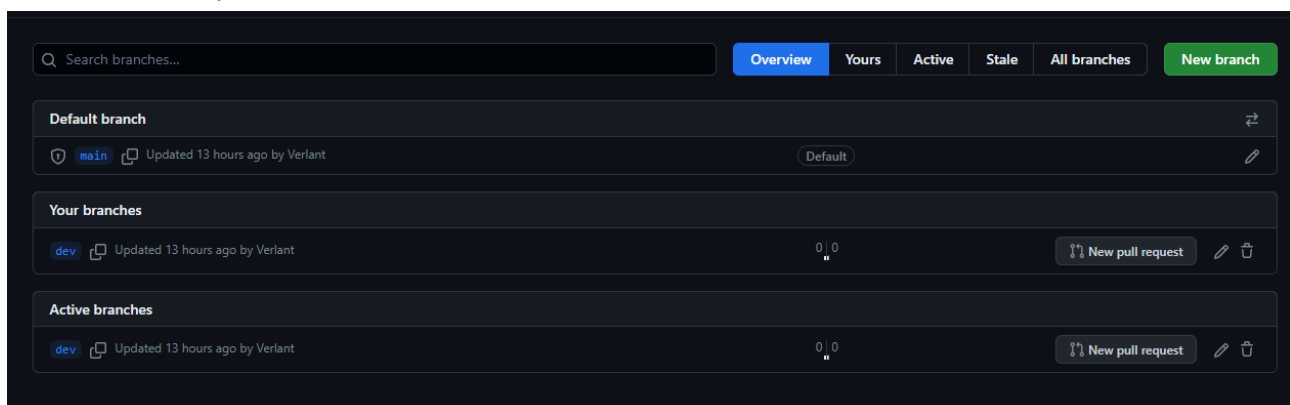
    panier terminer, espaceClient : reste a gérer la modification des informations clients. Reste a donné la possibilité de choisir une date de livraison par le client, re
ccessibles aux clients dont la ville n'est pas livrable, reste a faire la lotterie

commit b710aa42a67bb0416fda0da9f8d18f75ecb47afe
Author: Antoine Verlyck <antoine.verlyck@gmail.com>
Date: Sat Apr 29 21:42:06 2023 +0200

    ajout produit on click boutique (fetch) + inscription client + connexion + début espaceClient \n reste a faire : \n fin espaceClient , commandes, modif infos client
```

Commande « `git log` » fournissant l'historique des commits effectué.

Git peut être utilisé localement mais son plus grand avantage est de le coupler avec une plateforme en ligne, comme GitHub, doté d'un serveur. Cela permet d'avoir une version du projet commune en ligne et accessible n'importe où avec internet.



Liste des branch github du projet lafleur.

Mon choix d'IDE pour ce projet fut Visual Studio Code car il est gratuit et je code avec celui-ci depuis le début de ma formation, j'y suis donc habitué. Il a l'avantage d'être compatible avec la majorité des langages de programmation et facilement personnalisable grâce à un grand nombre d'extension disponible. Son inconvénient est qu'il n'est optimisé pour aucun langage particulier, l'abondance d'extensions peut devenir encombrante lors d'un manque de paramétrage efficace et enfin il peut être gourmand en ressource lorsqu'il est utilisé pour des projets importants et complexes.

2. Conception

Pour la partie conception, j'ai fais le choix d'utiliser Figma pour créer le design du site, MySQL Workbench afin de construire le *MLD* de la base de donnée ainsi que Canva pour dessiner le logo.

Base de donnée

MySQL Workbench est un logiciel de gestion de *BDD* informatisé, il permet de créer et modifier des *MLD* (Modèle Logique de Données) mais aussi d'ajouter, modifier, supprimer des données ou la base de donnée elle-même. Il est possible de faire cela graphiquement comme n'importe quel logiciel ou par des requêtes SQL. Afin de créer une *BDD* efficace pour ce projet j'ai conçu un dictionnaire de données en fonction des demandes du client ainsi qu'une recherche des entités dans lesquelles placer ces données.

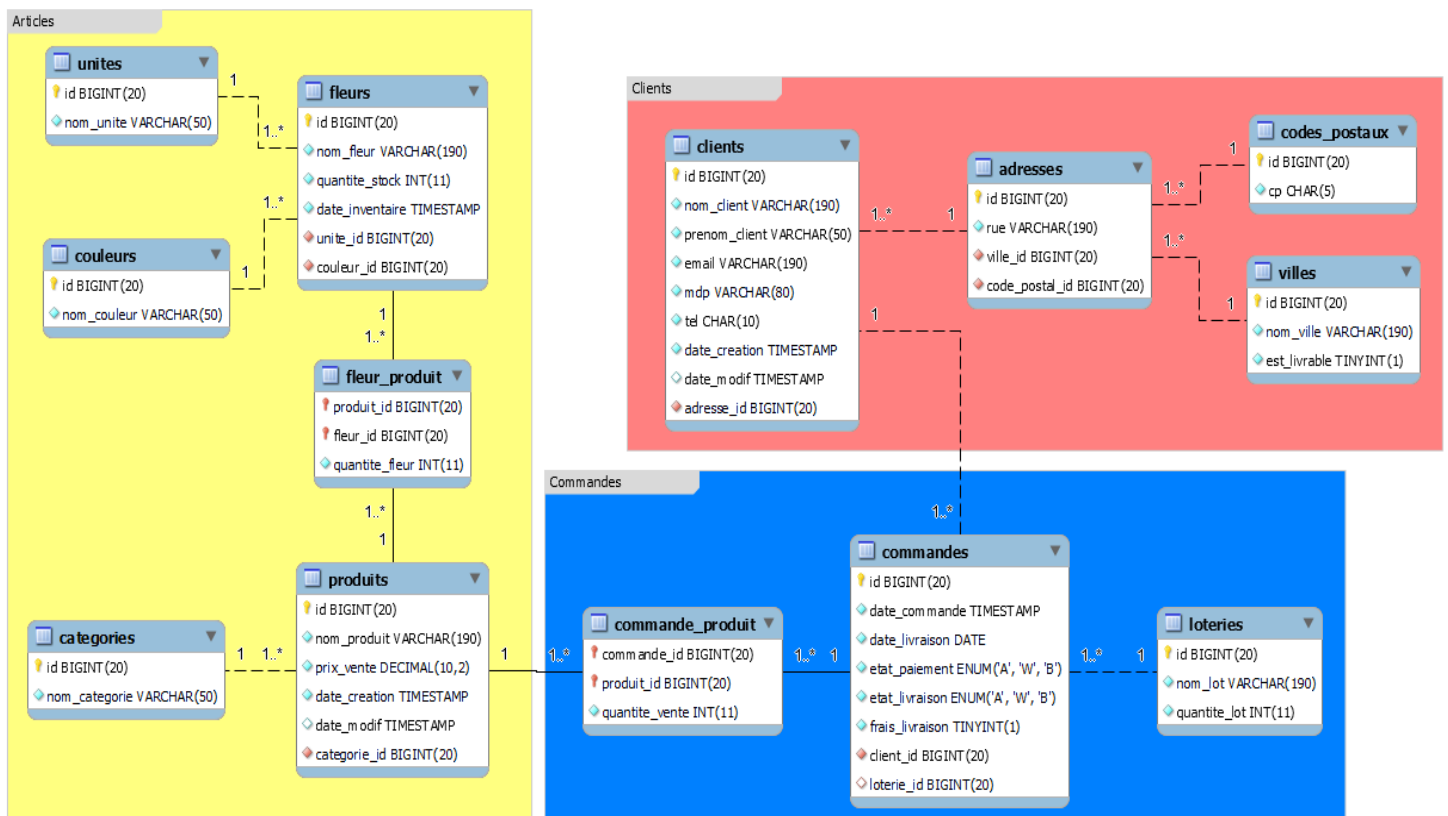
DESCRIPTION	CODE	TYPE
Nom du client	'nom_client	VARCHAR 190
Prenom du client	'prenom_client	VARCHAR 50
Email du client	'email	VARCHAR 190
Mot de passe du compte client	'mdp	VARCHAR 80
Téléphone du client	'tel	CHAR 10
Date de création de compte client	'date_creation	TIMESTAMP
Date de modification d'un compte client	'date_modif	TIMESTAMP
Adresse du client	'rue	VARCHAR 190
Ville de l'adresse du client	'nom_ville	VARCHAR 190
Code postal de l'adresse du client	'cp	CHAR 5
Ville livrable ou non	'est_livrable	BOOLEEN
Nom d'une fleur	'nom_fleur	VARCHAR 190
Quantité de fleur disponible en stock	'quantite_stock	INT
Date du dernier inventaire du stock	'date_inventaire	TIMESTAMP
Nom de l'unité à laquelle est vendue la fleur (tige, fleuron, gramme, etc..)	'nom_unite	VARCHAR 50
Nom de la couleur d'une fleur	'nom_couleur	VARCHAR 50
Nom d'un produit en vente	'nom_produit	VARCHAR 190
Prix d'un produit en vente	'prix_vente	DECIMAL (10,2)
Date de création d'un produit à la vente	'date_creation	TIMESTAMP
Date de modification d'un produit à la vente	'date_modif	TIMESTAMP
Nom de la catégorie d'un produit (remerciement, mariage, sentiments, etc..)	'nom_categorie	VARCHAR 50
Quantité de fleurs contenue dans un produit	'quantite_fleur	INT
Date de commande	'date_commande	TIMESTAMP
Date de livraison	'date_livraison	DATE
Suivie du paiement, A = validé, W = en attente, B = annulé	'etat_paiement	ENUM('A','W','B')
Suivie de livraison, A = validé, W = en attente, B = annulé	'etat_livraison	ENUM('A','W','B')
Frais de livraison ou non	'frais_livraison	BOOLEEN
Nom des lots disponible à la loterie	'nom_lot	VARCHAR 190
Quantité de lots disponible	'quantite_lot	INT
Quantité d'un produit acheté par un client lors d'une commande	'quantite_vente	INT

Dictionnaire de données

CODE	Client	Adresse	Ville	Code postal	Fleur	Unité	Couleur	Produit	Catégorie	Commande	Loterie
nom_client	x										
nom_client	x										
email	x										
mdp	x										
tel	x										
date_creation	x										
date_modif	x										
rue		x									
nom_ville			x								
cp				x							
est_livrable			x								
nom_fleur					x						
quantite_stock					x						
date_inventaire					x						
nom_unite						x					
nom_couleur							x				
nom_produit								x			
prix_vente								x			
date_creation								x			
date_modif								x			
nom_categorie									x		
quantite_fleur					x			x			
date_commande										x	
date_livraison										x	
etat_paiement										x	
etat_livraison										x	
frais_livraison										x	
nom_lot											x
quantite_lot											x
quantite_vente								x		x	

Recherche des entités

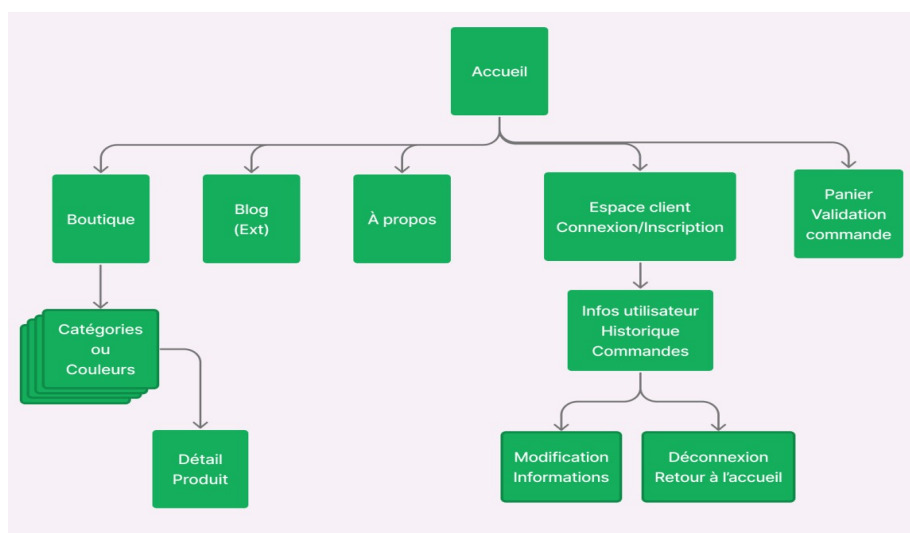
A partir de là j'ai entamé la création du *MLD*, sachant que j'utiliserai le framework *Laravel*, j'y ai appliqué la convention de nommage des entités afin d'arriver à cela.



Les champs `date_modif` sont NULL par défaut car ils ne servent que lors de modification de données. J'ai choisi de rendre le champ `loterie_id` NULL également, ce qui correspond au gain d'un lot par un client lors d'une commande. Si celui-ci gagne à la loterie le champ sera alors rempli par l'id correspondant au lot remporté.

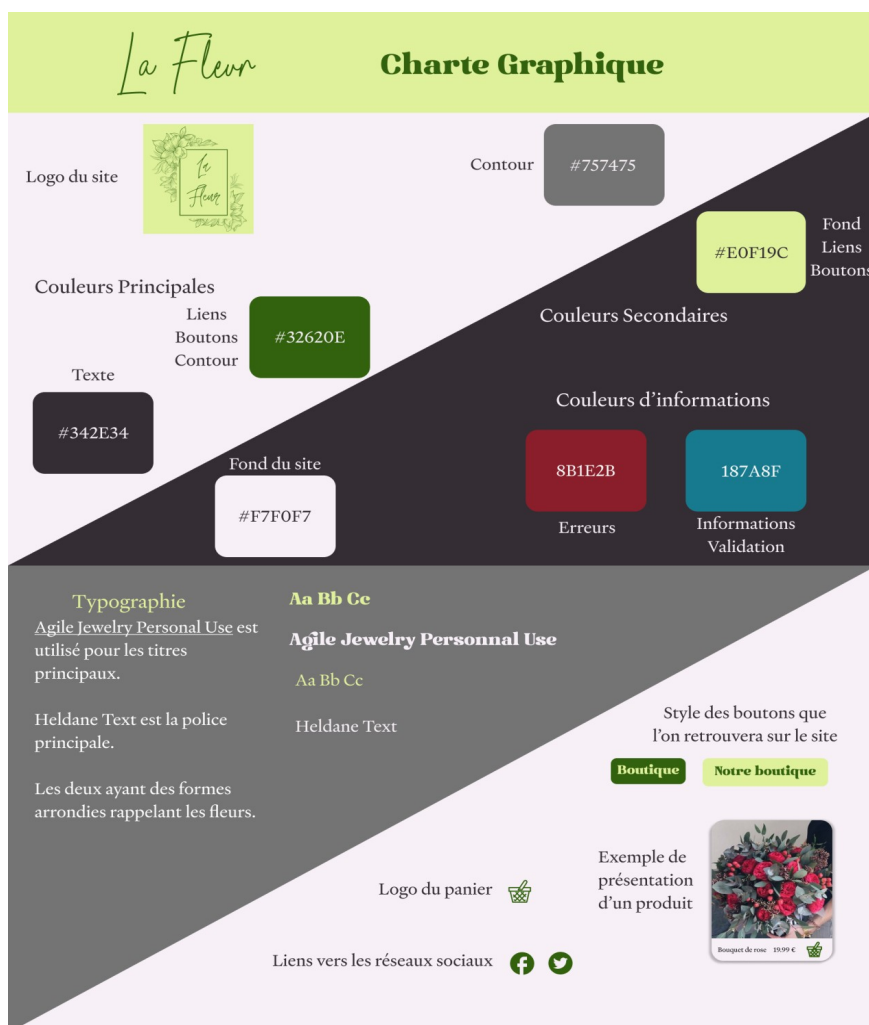
Conception graphique

Figma est un outil de design graphique, il permet, entre autres, de fabriquer des maquettes statiques ou des prototypes dynamiques de logiciels, sites ou applications informatiques. Afin de réaliser la maquette du projet, j'ai démarré par construire l'arborescence du site.



Arborescence du site

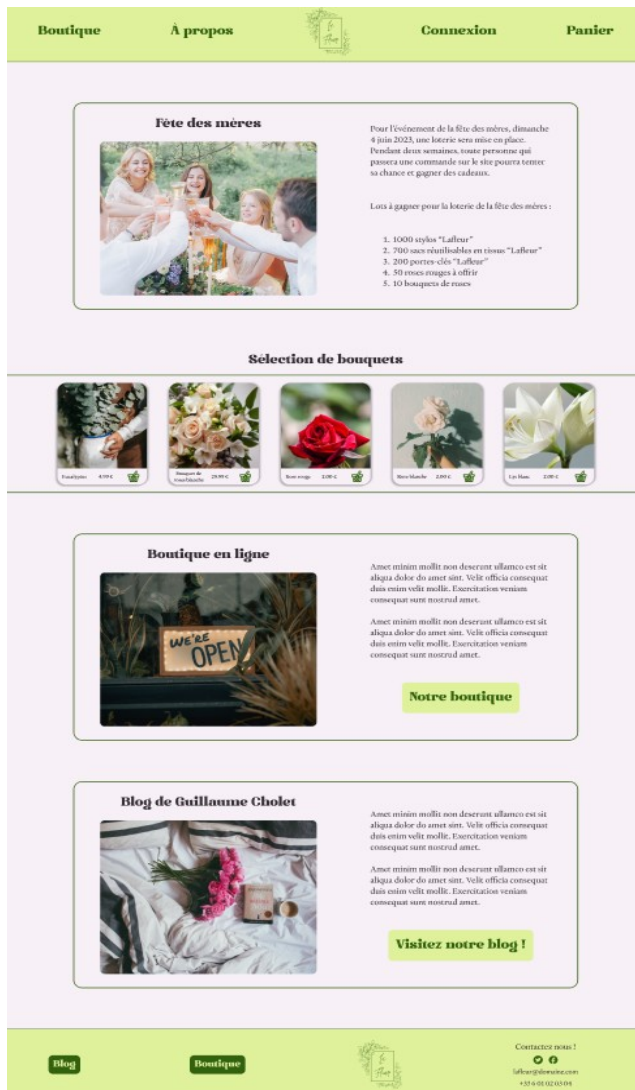
J'ai ensuite entamé une recherche graphique afin de proposer une charte graphique aux clients. Voici la charte graphique validé par les fleuristes :



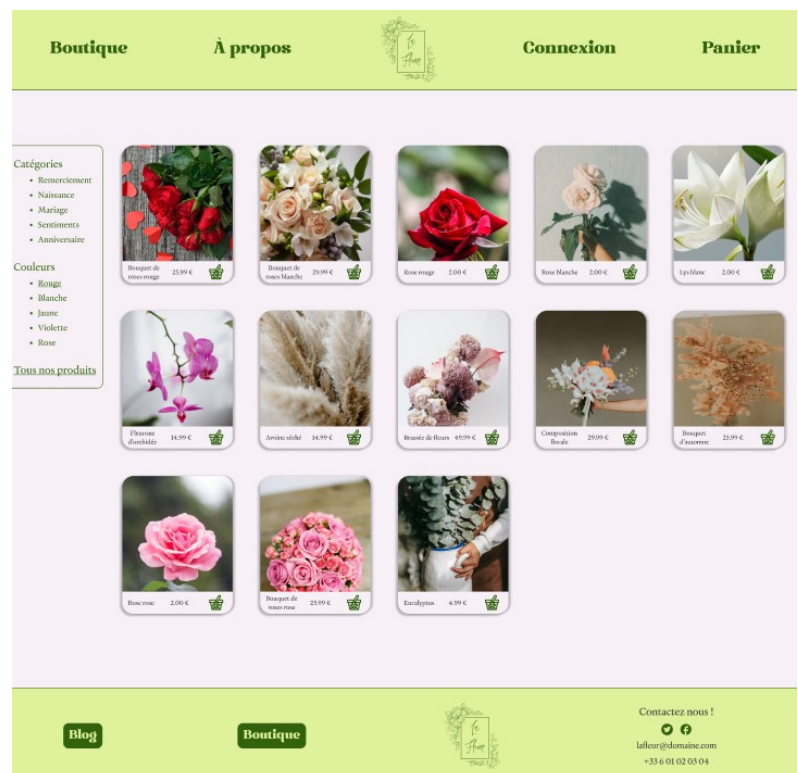
Charte graphique du projet

A la suite de cela, j'ai pu passer à la réalisation de la maquette du site. Le site comprendra une entête de page contenant les liens menant aux pages principales, c-à-d l'accueil, la boutique, l'espace client, le panier et la page à propos. Le lien de la page d'accueil se fera par le clic sur le logo situé au milieu de l'entête. Un pied de page sera également présent, celui-ci contiendra un lien vers la boutique en ligne, le blog des clients ainsi que les informations de contact et réseaux sociaux des fleuristes.

Sur la page d'accueil sera présent un bloc rassemblant les informations liés au prochain évènement et sa loterie associé mise en place par les clients. Une deuxième section servira à la présentation des derniers articles mise en vente sur la boutique. Puis deux dernières parties serviront à mettre en avant la boutique ainsi que le blog LaFleur. Pour la boutique, un menu comportant des filtres seront présent sur la gauche de la page et les articles seront visible sur le reste de la page.



Page Accueil



Page Boutique

3. Développement

Pour la partie développement du projet, j'ai décidé de démarrer par le côté back-end avec le panneau administrateur réalisé à l'aide du framework *Laravel*. Je suis ensuite passé au développement de la boutique en ligne LaFleur en utilisant les langages *HTML*, *CSS*, *JavaScript*, *PHP*. Enfin, j'ai terminé le projet par la réalisation du blog sur *WordPress*.

Langages et technologies utilisées

Langage PHP

Ce langage est exécuté sur un serveur, contrairement aux langages HTML, CSS et JavaScript qui sont traduits par le navigateur côté client, il génère le HTML qui sera envoyé aux clients. Cette manière d'exécution permet plus de sécurité car le code PHP ne sera pas accessible par les utilisateurs. Il permet aussi de faire des requêtes à des bases de données afin de récupérer ces données et les afficher sur le navigateur ou encore d'insérer des informations renseignées par le client dans une *BDD*, au travers d'un formulaire.

PHP permet aussi de traiter des méthodes de requête *HTTP* (*Hypertext Transfer Protocol*). Ces méthodes, aussi appelées *verbes HTTP*, servent à traiter des informations en fonction de l'action de l'utilisateur.

Template vues PHP

```
switch ($uc) {
    case 'accueil':
        include 'app/views/v_accueil.php';
        break;
    case 'boutique':
        include "app/views/v_boutique.php";
        break;
    case 'produit':
        include "app/views/v_produit.php";
        break;
    case 'aPropos':
        include "app/views/v_aPropos.php";
        break;
    case 'espaceClient':
        include "app/views/v_espaceClient.php";
        break;
    case 'connexion':
        include "app/views/v_connexion.php";
        break;
    case 'panier':
        include "app/views/v_panier.php";
        break;
    case 'modifierInfos':
        include "app/views/v_modifInfos.php";
        break;
    case 'boutique':
        include "app/views/v_boutique.php";
        break;
    default:
        break;
}
```

Celles que j'ai utilisées sont :

- *GET* – méthode permettant de récupérer des données en lecture seule. Celle-ci ne doit jamais être utilisée dans le but d'insérer des données dans une *BDD*. Cela augmenterait grandement le risque d'injection *SQL*.
- *POST* – méthode servant à récupérer des informations saisies dans un formulaire au serveur afin d'ajouter ou modifier des données dans une *BDD*. Cette méthode peut malheureusement créer des effets de bord. Si celle-ci est envoyée successivement les effets seront additionnels.
- *PUT* – méthode ayant le même effet que *POST* à la seule différence que cette méthode est idempotente, c-à-d que même lorsqu'elle est envoyée successivement, ces effets ne seront pas additionnels et auront toujours le même effet.
- *DELETE* – méthode utilisée afin de supprimer des données dans une *BDD*

Langage HTML

HTML (*Hypertext Markup Language*) est un langage permettant de structurer une page web, en utilisant des balises telles que *header*, *body*, *main*, *footer*. Ces balises servent à englober du contenu et ont toute une sémantique précise. La balise *header* désignera l'en-tête de page tandis que *footer* le pied de page. La balise *body* contiendra le corps du contenu dans lequel on placera les 2 balises nommées précédemment et la balise *main* qui désignera la partie principale de la page.

Une dernière balise importante est *head*, celle-ci correspond à l'en-tête du document source HTML. Elle servira de conteneur pour le titre du site (via la balise *title*), les liens CSS, grâce à la balise *link*, permettant de styliser la page web. Les liens *CDN* (*Content Delivery Network*) qui permettent d'insérer des bibliothèques au projet, les balises *script* permettant d'y écrire du JavaScript ou bien de lier un fichier JavaScript au document. Ou encore la balise *meta* contenant les métadonnées du document via les attributs de cette balise comme le type d'encodage utilisé dans le document, en utilisant l'attribut *charset*, ou bien la valeur *Content-Security-Policy* contenue dans l'attribut *http-equiv* servant de protection contre les attaques XSS (*Cross-Site Scripting*).

```
<!-- balise doctype html signifiant au navigateur
que ce document est de type HTML -->
<!DOCTYPE html>
<!-- balise html marquant la racine du document
attribut lang définissant la langue de base du site -->
<html lang="fr">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta http-equiv="Content-Security-Policy" content="default-src 'self'
    https://fonts.cdnfonts.com/css/agile-jewelry-personal-use
    https://fonts.cdnfonts.com/css/heldane-text
    https://code.jquery.com/jquery-3.6.4.slim.min.js
    https://cdnjs.cloudflare.com/ajax/libs/jquery.spritely/0.6.8/jquery.spritely.min.js">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>LaFleur</title>
  <link rel="icon" href="public/img/logo-la-fleur.svg" />
  <link rel="stylesheet" href="public/css/main.css">
  <link href="https://fonts.cdnfonts.com/css/agile-jewelry-personal-use" rel="stylesheet">
  <link href="https://fonts.cdnfonts.com/css/heldane-text" rel="stylesheet">
  <script src="https://code.jquery.com/jquery-3.6.4.slim.min.js" integrity="sha256-a2yjHM4jnf9f54xUQakjZGaqYs/
    V1CYvWpoqZzC2/Bw=" crossorigin="anonymous"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery.spritely/0.6.8/jquery.spritely.min.js"></script>
</head>
```

balise head du site LaFleur

Langage CSS

Le langage *CSS* (*Cascading Style Sheets*), ou feuille de style en cascade en français, est le langage servant à gérer l'aspect graphique d'un projet web. Par le biais de sélecteur, il permet de cibler des balises HTML afin d'y appliquer des règles de style, appelé propriété CSS, comme une couleur de texte ou de fond, une police d'écriture particulière ou encore le positionnement visuel de cette balise lors du rendu dans un navigateur.

Il existe différents types de sélecteurs ayant chacun leur priorité d'application. Il est possible de sélectionner une balise juste en utilisant sa dénomination, (*header*, *main*, *footer*, etc..), en créant des classes que l'on ajoute en tant qu'attribut *class* dans une balise *HTML* puis que l'on sélectionne dans notre fichier *CSS* en utilisant la syntaxe *.nom-class* ou encore en utilisant l'attribut *id* dans une balise *HTML* puis en le sélectionnant *#nom-id*, contrairement à l'attribut *class*, la valeur de l'attribut *id* doit être unique pour tout le document *HTML*. Comme dit précédemment, ces différents sélecteurs ont un ordre de priorité d'application. Ordre de priorité pour ces 3 types de sélecteurs : *#nom-id* > *.nom-class* > *header*. Le terme *cascading* de l'acronyme *CSS* désigne le fait que la navigateur lira les fichiers *CSS* de haut en bas.

Lorsque 2 mêmes sélecteurs de même priorité seront appelés à différents endroits du fichier et qu'une ou plusieurs propriétés CSS avec différentes valeurs seront utilisées, les dernières valeurs de ces propriétés remplaceront les précédentes.

Pour ce qui est du design *responsive*, CSS intègre les *mediaqueries*. Celles-ci servent à appliquer des règles CSS sous conditions. On va les employer afin de modifier le style du site en fonction de la largeur de l'écran de l'appareil de l'utilisateur, tous ça dans le but d'avoir un site gardant le même style global tout en étant lisible depuis n'importe quel appareil.

```
main {
  display: flex;
  flex-direction: column;
  padding: 50px 0px;
  gap: 100px;
  min-height: calc(99.9vh - 280px);
}

img {
  border-radius: 10px;
}

.content-section {
  max-width: 50%;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding: 10px;
  gap: 20px;
}
```

CSS sélecteurs et propriétés

```
@media (max-width: 1100px) {
  .form-section {
    width: 60%;
  }
  .basic-section {
    max-width: 60%;
    flex-direction: column;
    padding: 10px 25px;
  }
  .content-section {
    max-width: 100%;
  }
  .card-accueil:last-child {
    display: none;
  }
  .section-panier {
    max-width: 80%;
  }
}
```

CSS mediaqueries

Langage JavaScript

Le langage *JavaScript* est utilisé afin d'améliorer l'interactivité du site et l'expérience utilisateur. Il permet d'écrire des fonctions servant à générer puis insérer du contenu *HTML* dans une page web coté client (contrairement à *PHP* qui le fait depuis le serveur), d'ajouter des *EventListener* (écouteurs d'évènements), permettant de lancer des scripts lors d'une interaction d'un utilisateur, ou encore de construire des fonctions appelant un script *PHP* qui ira faire une requête dans une *BDD* sans qu'il y ait de rechargement de page, que l'on nomme *requêtes AJAX*.

```
const passwordInputs = document.querySelectorAll(".password");

passwordInputs.forEach((passwordInput) => {
  passwordInput.addEventListener("change", () => {
    if (password.value !== password_verify.value) {
      if (
        !password_verify.classList.contains("border-error") &&
        !password.classList.contains("border-error")
      ) {
        password.classList.add("border-error");
        password_verify.classList.add("border-error");
        password.classList.remove("border-validate");
        password_verify.classList.remove("border-validate");
      }
    } else {
      password.classList.remove("border-error");
      password_verify.classList.remove("border-error");
      password.classList.add("border-validate");
      password_verify.classList.add("border-validate");
    }
  });
});
```

JavaScript EventListener

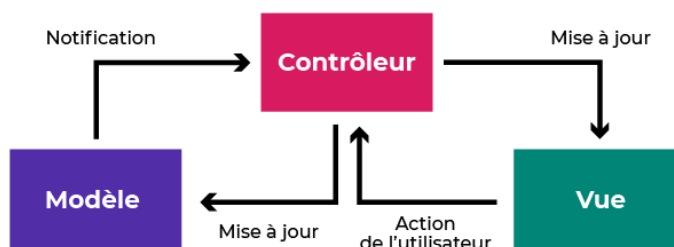
Framework Laravel

Basé sur *PHP*, *Laravel* respecte le modèle *MVC (Modèle-Vue-Contrôleur)* et est entièrement développé en *POO (Programmation Orienté Objet)*. Il utilise un *ORM (Object-Relational Mapping)* appelé *Eloquent* et s'installe via *Composer*.

La *Programmation Orienté Objet (POO)*, est un paradigme de programmation informatique basé sur ce que l'on appelle un objet. Le principe est de créer des classes qui feront office de moule afin de fabriquer ces objets. Ces classes devront contenir des attributs et/ou des méthodes (fonctions de ces classes) servant à stocker des données (attribut) et à manipuler ces objets (méthodes). L'avantage de ce paradigme est d'encapsuler ces attributs et méthodes à travers ces objets qui peuvent représenter ce que l'on souhaite.

L'architecture *MVC* est un modèle conceptuel servant à séparer toute application web en trois parties communicantes :

- Les modèles, servent à gérer les requêtes sur les *BDD*
- Les vues, affichent le contenu du site
- Les contrôleurs, servent de passerelle entre les modèles et les vues



Représentation graphique de l'architecture MVC

Généralement, ces 2 concepts vont être couplés. Les modèles seront des classes servant à modéliser les données de la *BDD* à travers des objets, les contrôleurs seront aussi des classes qui manipuleront ces objets afin de les envoyer aux vues. Les vues seront des fichiers *PHP* classiques générant la partie *HTML* en y intégrant les données renvoyées par le contrôleur.

Pour la gestion des modèles, *Laravel* utilise *Eloquent*, un *Object-Relational Mapping*. Un *ORM* se base sur la *POO* afin de gérer les requêtes à une *BDD* à notre place en nous permettant de manipuler des objets. Celui-ci se chargera ensuite de traduire les attributs de ces objets en requête lors d'interaction avec ces *BDD*.

Pour terminer, *Laravel* s'installe en utilisant *Composer*. Celui-ci est un logiciel de gestionnaire de dépendances. Il permet l'installation et la configuration de différentes librairies / paquets pour le développement de sites ou applications web.

Panneau administrateur

J'ai choisi de commencer le développement du projet par la partie back-end car cela m'a permis de construire la *BDD* et de créer des données depuis le framework grâce aux *Factories* et *DatabaseSeeder*. J'ai aussi trouvé ce choix plus judicieux car lors du développement de la boutique LaFleur, les données étant déjà présentes, cela permettra d'avoir le rendu attendu avec du contenu pertinent tout en pouvant modifier les données depuis cette application.

Concernant le développement de cette partie, j'ai démarré par la création de migrations en me basant sur mon *MLD* en vue de construire la *BDD*. J'ai utilisé ces commandes dans le terminal de VSCode :

1. `composer create-project --prefer-dist laravel/laravel backOfficeLaFleur` – créer le projet *Laravel*
2. `composer require barryvdh/laravel-debugbar --dev` – Ajouter la debug bar
3. `php artisan make:migration create_nomTable_table` – Création d'un fichier de migration
4. `php artisan migrate` – Construit la *BDD* à partir des fichiers de migration créés précédemment

```
<?php

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

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('clients', function (Blueprint $table) {
            $table->id();
            $table->string('nom_client', 190);
            $table->string('prenom_client', 50);
            $table->string('email', 190)->unique();
            $table->string('mdp', 80);
            $table->char('tel', 10);
            $table->timestamp('date_creation')->default(DB::raw('CURRENT_TIMESTAMP'));
            $table->timestamp('date_modif')->nullable();
            $table->unsignedBigInteger('adresse_id');
            $table->foreign('adresse_id')->references('id')->on('adresses');
            // $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('clients');
    }
};
```

fichier de migration de la table clients

Ensuite j'ai décidé d'ajouter le paquet *Breeze* au projet. Celui-ci implémente un système d'authentification accompagné du framework CSS *Tailwind* ainsi que de l'outil *Vite* permettant de lancer un serveur de développement qui implémentera *Tailwind* et rechargement la page du projet en développement à chaque sauvegarde enregistrée. J'ai utilisé pour cela les commandes suivantes :

1. composer require laravel/breeze --dev – Ajout des dépendances du paquet *Breeze*
2. php artisan breeze:install – Installation de *Breeze*
3. npm install – Installation des dépendances npm
4. npm run dev – Afin de lancer le serveur *Vite*

Pour continuer, j'ai créé les différents modèles correspondant à chacune des tables de ma *BDD*, à l'exception des tables « fleur_produit » et « commande_produit ». Pour ces deux tables j'ai choisi d'utiliser la méthode `belongsToMany` de *Laravel* créant une relation pivot entre les modèles représentant les tables reliées aux tables « fleur_produit » et « commande_produit ». J'ai donc utilisé la commande « php artisan make:model NomModel » puis ajouté les attributs et méthodes nécessaire.

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Fleur extends Model
{
    use HasFactory;
    protected $table = "fleurs";
    protected $primaryKey = "id";
    protected $fillable = [
        "nom_fleur",
        "quantite_stock",
        "date_inventaire"
    ];
    public $timestamps = false;

    public function unite()
    {
        return $this->belongsTo(Unite::class);
    }

    public function couleur()
    {
        return $this->belongsTo(Couleur::class);
    }

    public function produits()
    {
        return $this->belongsToMany(Produit::class)->withPivot('quantite_fleur');
    }
}
```

Model Fleur

L'étape suivante fut la création des données de test, en vue d'hydrater la *BDD* pour pouvoir passer à la gestion des vues et des contrôleurs. Pour cela j'ai créé des *Factories*, en utilisant la commande « `php artisan make:factory NomFactory --model=NomModel` », ce sont des classes permettant de générer du contenu pour chaque attribut des modèles correspondant pour ensuite insérer ces données dans la *BDD* via le fichier *DatabaseSeeder* et la commande « `php artisan db:seed` ». J'ai également créé 2 comptes administrateur dans le fichier *DatabaseSeeder*, les fleuristes pourront toujours modifier leur mot de passe ou faire un autre compte grâce au paquet *Breeze*. L'utilisateur 'Test User' sera supprimé en production.

```
<?php

namespace Database\Factories;

use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Facades\Hash;

/**
 * @extends \Illuminate\Database\Eloquent\Factories\Factory<\App\Models\Client>
 */
class ClientFactory extends Factory
{
    /**
     * Define the model's default state.
     *
     * @return array<string, mixed>
     */
    public function definition(): array
    {
        return [
            "nom_client" => ucwords($this->faker->words(1, true)),
            "prenom_client" => ucwords($this->faker->words(1, true)),
            "email" => $this->faker->unique()->safeEmail(),
            "mdp" => Hash::make($this->faker->password()),
            "tel" => $this->faker->numerify('06#####'),
            "date_creation" => $this->faker->dateTime(),
            "adresse_id" => $this->faker->unique()->numberBetween(1, 10),
        ];
    }
}
```

ClientFactory

```
class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     */
    public function run(): void
    {
        $faker = new Faker();

        // \App\Models\User::factory(10)->create();

        \App\Models\User::factory()
            ->create([
                'name' => 'Test User',
                'email' => 'test@example.com',
            ])
            ->create([
                'name' => 'Guillaume Cholet',
                'email' => 'guillaume.cholet@lafleur.com',
                'password' => 'lafleur'
            ])
            ->create([
                'name' => 'Sabine Cholet',
                'email' => 'sabine.cholet@lafleur.com',
                'password' => 'lafleur'
            ]);

        Client::factory(10)->create();
    }
}
```

DatabaseSeeder

Je suis ensuite passé à la création des contrôleurs. Pour cela j'ai utilisé la commande « `php artisan make:controller NomController --resource` ». L'option « *resource* » sert à créer la définition des méthodes *CRUD* (*create, read, update, delete*). J'ai ensuite construit le corps de ces méthodes de la manière suivante :

- *index* – exécute une requête de lecture à la *BDD* afin de récupérer la liste des données correspondantes au modèle pour les envoyer à la vue.
- *create* – sert à renvoyer une vue munie de champs de saisie permettant à l'utilisateur d'ajouter des données. Cette méthode nécessite parfois de faire une requête de lecture à la *BDD* dans le but d'associer des données d'autres modèle à celui qui sera conçu.
- *store* – récupère et traite les données saisies dans la vue renvoyé par la méthode *create* par le biais d'un formulaire, en utilisant la méthode *PUT*, afin d'ajouter ces données à la *BDD*.
- *show* – requête les informations d'une entité particulière dans la *BDD* afin de les renvoyer dans la vue correspondante.
- *edit* – exécute une requête de lecture dans la *BDD* afin de récupérer les informations d'une entité précise dans le but de la modifier au travers d'un formulaire dans la vue correspondante.

- update – récupère les informations, par la méthode *PUT*, saisie dans le formulaire de la vue renvoyé par la méthode edit et les traitent afin de mettre a jour les données de l'entité correspondante.
- destroy - permet de supprimer les données de l'entité correspondante dans la *BDD* en utilisant la méthode *DELETE*.

```
/**
 * Display a listing of the resource.
 */
public function index()
{
    //
    $fleurs = Fleur::orderBy('quantite_stock', 'ASC')->get();
    return view('fleurs.index', [
        'fleurs' => $fleurs,
    ]);
}
```

index modèle fleur

```
/**
 * Show the form for creating a new resource.
 */
public function create()
{
    //
    $couleurs = Couleur::all();
    $unites = Unite::all();
    return view('fleurs.create', [
        'couleurs' => $couleurs,
        'unites' => $unites,
        'fleurs' => Fleur::all()
    ]);
}
```

create modèle fleur

```
/**
 * Store a newly created resource in storage.
 */
public function store(Request $request)
{
    //
    if ($request->validate([
        "nom-fleur" => "required|string|min:3|max:190",
        "quantite-stock-fleur" => "required|int",
        "couleur-fleur" => "required|int",
        "unite-fleur" => "required|int",
    ])) {
        $nom_fleur = $request->input('nom-fleur');
        $quantite_stock = $request->input("quantite-stock-fleur");
        $couleur_fleur = $request->input("couleur-fleur");
        $unite_fleur = $request->input("unite-fleur");
        $fleur = new Fleur();
        $fleur->nom_fleur = $nom_fleur;
        $fleur->quantite_stock = $quantite_stock;
        $fleur->couleur_id = $couleur_fleur;
        $fleur->unite_id = $unite_fleur;
        $fleur->date_inventaire = now();
        $fleur->save();
        return redirect()->route("fleurs.index");
    } else {
        return redirect()->back();
    }
}
```

store modèle fleur

```
/**
 * Display the specified resource.
 */
public function show(string $id)
{
    //
    $fleur = Fleur::find($id);
    return view('fleurs.show', [
        'fleur' => $fleur,
        'fleurs' => Fleur::all()
    ]);
}
```

show modèle fleur

```
/**
 * Show the form for editing the specified resource.
 */
public function edit(string $id)
{
    //
    $fleur = Fleur::find($id);
    $couleurs = Couleur::all();
    $unites = Unite::all();
    return view('fleurs.edit', [
        'fleur' => $fleur,
        'couleurs' => $couleurs,
        'unites' => $unites,
        'fleurs' => Fleur::all()
    ]);
}
```

edit modèle fleur

```
/**
 * Update the specified resource in storage.
 */
public function update(Request $request, string $id)
{
    //
    if ($request->validate([
        "nom-fleur" => "required|string|max:45",
        "quantite-stock-fleur" => "required|int",
        "couleur-fleur" => "required|int",
        "unite-fleur" => "required|int",
    ])) {
        $nom_fleur = $request->input('nom-fleur');
        $quantite_stock = $request->input("quantite-stock-fleur");
        $couleur_fleur = $request->input("couleur-fleur");
        $unite_fleur = $request->input("unite-fleur");
        $fleur = Fleur::find($id);
        $fleur->nom_fleur = $nom_fleur;
        $fleur->quantite_stock = $quantite_stock;
        $fleur->couleur_id = $couleur_fleur;
        $fleur->unite_id = $unite_fleur;
        $fleur->date_inventaire = now();
        $fleur->save();
        return redirect()->route("fleurs.index");
    } else {
        return redirect()->back();
    }
}
```

update modèle fleur

```

/**
 * Remove the specified resource from storage.
 */
public function destroy(string $id)
{
    //
    $fleur = Fleur::find($id);
    $id_fleurs = [];
    foreach ($fleur->produits as $produit) {
        $id_fleurs[] = $produit->pivot->produit_id;
    }
    if (in_array($id, $id_fleurs)) {
        // Stocker un message dans la session
        session()->flash('message', 'La fleur est rattachée a un produit et ne peut donc pas être supprimée.');
```

destroy modèle fleur

Pour améliorer l'expérience utilisateur, j'ai ajouté la librairie *Select2* fonctionnant avec *JQuery*. Cette librairie sert à rendre plus simple d'utilisation les inputs de type « select » en ajoutant une barre de recherche et en améliorant l'interface utilisateur de base. Afin d'associer les contrôleurs aux vues, il faut pour cela configurer les routes. Elles vont servir à lier les chemins des contrôleurs avec les vues renvoyées par ses méthodes. Ces routes sont déclarés dans un fichier commun appelé *web.php*. Grâce à l'option « resource » utilisée lors de la création de nos contrôleurs nous pouvons utiliser une fonction qui s'occupera de lier chaque méthode du contrôleur avec chaque vue correspondante. Sans cela il faudrait créer une route spécifique pour chaque méthode et chaque vue. Ici j'ai créé des méthodes supplémentaires, j'ai donc dû préciser leurs routes associées avec les *verbes HTTP* utilisés par ces méthodes.

```

Route::get('/', function () {
    return view('welcome');
});

Route::get('/dashboard', function () {
    return view('dashboard');
})->middleware(['auth', 'verified'])->name('dashboard');

Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])->name('profile.destroy');
});

require __DIR__ . '/auth.php';

Route::resource('fleurs', FleurController::class);
Route::resource('produits', ProduitController::class);
Route::resource('commandes', CommandeController::class);
Route::resource('villes', VilleController::class);
Route::post('produits/{id}/attach', [ProduitController::class, 'attach'])->name('produits.attach');
Route::get('produits/{id_produit}/detach/{id_fleur}', [ProduitController::class, 'detach'])->name('produits.detach');
```

routes du fichier web


Les routes pour le dashboard et auth ont été créées automatiquement par le paquet *Breeze*.

Pour les vues, *Laravel* utilise *Blade*, c'est un moteur de template permettant d'écrire du code *PHP* ainsi que de créer des composants. Les composants sont des fichiers *Blade* que l'on intègre dans les vues générales. Ils permettent d'éviter de dupliquer du code. Avec *Blade*, il est possible de créer des templates, c-à-d que l'on va utiliser une architecture principale dans laquelle on injectera nos vues.

```
<x-app-layout>
  <x-slot name="header">
    <h1 class="font-semibold text-xl text-gray-800 leading-tight">
      {{ __('Modifier la fleur') }} {{ $fleur->nom_fleur }}
    </h1>
  </x-slot>
  <x-stock-alert :fleurs="$fleurs"></x-stock-alert>
  <div class="max-w-7xl w-10/12 mx-auto mt-6 py-6 px-4 sm:px-6 lg:px-8 bg-white rounded shadow">
    <form action="{{ route('fleurs.update', $fleur->id) }}" method="POST">
      @method('PUT')
      @csrf
      <div class="flex flex-col">
        {{-- Input nom fleur --}}
        <label for="nom-fleur" class="text-gray-800 mb-1 font-bold text-center sm:text-left">
          {{ __('Nom') }}
        </label>
        <input type="text" name="nom-fleur" id="nom-fleur" value="{{ $fleur->nom_fleur }}"
          class="rounded border-gray-400 mb-1">
        @error('nom-fleur')
          <div class="text-red-500">{{ $message }}</div>
        @enderror
        {{-- Input quantite stock --}}
        <label for="quantite-stock-fleur" class="text-gray-800 mb-1 font-bold text-center sm:text-left">
          {{ __('Quantité en stock') }}
        </label>
        <input type="text" name="quantite-stock-fleur" id="quantite-stock-fleur"
          value="{{ $fleur->quantite_stock }}" class="rounded border-gray-400 mb-1">
        @error('quantite-stock-fleur')
          <div class="text-red-500">{{ $message }}</div>
        @enderror
        {{-- Select couleur --}}
        <div class="flex flex-col gap-1 items-center mt-4 sm:flex-row">
          <label for="couleur-fleur" class="text-gray-800 mb-1 font-bold self-center">
            {{ __('Couleur') }}
          </label>
          <select name="couleur-fleur" id="couleur-fleur" class="rounded border-gray-400 select2">
            <option value="{{ $fleur->couleur_id }}">{{ $fleur->couleur->nom_couleur }}</option>
            @foreach ($couleurs as $couleur)
              <option value="{{ $couleur->id }}">{{ $couleur->nom_couleur }}</option>
            @endforeach
          </select>
          {{-- Select unite --}}
          <label for="unite-fleur" class="text-gray-800 ml-0 sm:ml-5 mb-1 font-bold self-center">
            {{ __('Unite') }}
          </label>
          <select name="unite-fleur" id="unite-fleur" class="rounded border-gray-400 select2">
            <option value="{{ $fleur->unite_id }}">{{ $fleur->unite->nom_unite }}</option>
            @foreach ($unites as $unite)
              <option value="{{ $unite->id }}">{{ $unite->nom_unite }}</option>
            @endforeach
          </select>
        </div>
        <div class="mt-4 max-w-fit sm:max-w-none mx-auto flex flex-col justify-end sm:flex-row sm:mx-0">
          <x-buttons.submit></x-buttons.submit>
          <x-buttons.reset></x-buttons.reset>
          <x-buttons.back :href="route('fleurs.index')"></x-buttons.back>
        </div>
      </div>
    </form>
  </div>
</x-app-layout>
```

vue edit modèle fleur

Pour finir, après avoir développé la gestion des fleurs et produits, je me suis occupé de la gestion des commandes. En suivant le même schéma que pour les tables fleurs et produits, il est possible de consulter les commandes et de modifier leur état de paiement et de livraison. Lorsque le stock d'une fleur est inférieur ou égal à 100, un message apparaît dans toutes les vues pour informer les fleuristes accompagné de boutons leur permettant de modifier, voir ou supprimer la fleur en question. Une modale de confirmation s'ouvre lorsque l'utilisateur tente de supprimer un élément. Les administrateurs peuvent modifier la quantité de stock d'une fleur et ajouter, modifier ou supprimer des fleurs et produits.

 [Tableau de bord](#) [Fleurs](#) [Produits](#) [Commandes](#) [Villes](#) Test User ▾


Toutes les commandes

Le stock de la fleur Rose est bientôt épuisé, il reste actuellement 100 fleurs.

[Modifier](#) [Voir](#) [Supprimer](#)

Date commande	Date livraison	Suivi paiement	Suivi livraison	Prix total	ACTIONS
07/05/2023 18:46	08/05/2023	Validé	Validé	190.88 €	Voir
07/05/2023 18:46	08/05/2023	Annulé	Annulé	269.69 €	Voir
07/05/2023 18:48	08/05/2023	Validé	Attente	28.98 €	Modifier Voir
07/05/2023 18:49	08/05/2023	Attente	Attente	12.99 €	Modifier Voir

consultation des commandes

 [Tableau de bord](#) [Fleurs](#) [Produits](#) [Commandes](#) [Villes](#) Test User ▾

Modifier la commande 4


Date de livraison
08/05/2023

Client
Prenom
Nom

Adresse
14 rue du pré
Lourmarin
84360

Etat du paiement
Validé

Etat de la livraison
Attente

Date livraison: 08/05/2023 

Suivi paiement: Validé ▾

Suivi livraison: Annulé ▾

[Sauvegarder](#) [Retour](#)

modification d'une commande

Modifier la fleur rose

Le stock de la fleur Rose est bientôt épuisé, il reste actuellement 100 fleurs.

Modifier

Voir

Supprimer

Nom

rose

Quantité en stock

100

Couleur

rouge

Unité

tige

Sauvegarder

Effacer

Retour

modification d'une fleur

Ajouter un produit

Le stock de la fleur Rose est bientôt épuisé, il reste actuellement 100 fleurs.

Modifier

Voir

Supprimer

Prix de vente

10.99

Nom du produit

Bouquet de fleur

Catégorie

remerciement

Fleur

rose rouge

rose blanc

Quantité de fleurs

5

Sauvegarder

Effacer

Retour

ajout d'un produit

Boutique

Pour la boutique en ligne j'ai suivie l'architecture *MVC*. J'ai créé un modèle d'accès aux données qui ne s'occupe que de la connexion et des requêtes à la *BDD* ainsi qu'un modèle par table. Chaque modèle utilise les méthodes du modèle d'accès aux données afin de gérer les requêtes. Pour les contrôleurs, j'ai développé un contrôleur général appelé « *index.php* » qui s'occupe de gérer les requêtes *HTTP* et fournir les données et vues nécessaires en fonction des variables récupérées par les méthodes *POST* et *GET*. Celui-ci appelle les différents contrôleurs qui accèdent aux modèles afin de communiquer avec la *BDD*. Pour la gestion des vues, j'ai développé un fichier nommé « *template.php* » qui contient l'entête du document *HTML*, le *header* et *footer* du site, ainsi qu'une boucle « *switch* » qui renverra la vue demandé.

Afin de construire le modèle d'accès aux données, j'ai utilisé *PDO (PHP Data Objects)*, fournit de base avec *PHP* il permet d'exécuter des requêtes *SQL* depuis un script *PHP*. Premièrement il faut établir la connexion à la *BDD*, *PDO* étant un objet, il faut instancier cette classe en lui fournissant l'adresse et le nom de la *BDD* ainsi que l'identifiant et le mot de passe d'un utilisateur de cette *BDD*. J'ai donc déclaré des attributs privés et statiques ainsi qu'une méthode statique utilisant ses attributs afin d'instancier un *PDO*.

J'ai utilisé des attributs privés car ces informations sont sensibles et de cette manière seule la classe « *M_AccesDonnees* » aura accès à ces attributs. De plus n'ayant aucun besoin d'instancier cette classe j'ai déclaré mes attributs et méthodes en statique.

```
class M_AccesDonnees
{
    private static String $serveur = 'mysql:host=localhost';
    private static String $bdd = 'dbname=lafleur';
    private static String $user = 'root';
    private static String $mdp = '';

    /**
     *
     * @var PDO
     */
    private static $monPdo;

    /**
     * Fonction statique qui crée l'unique instance de la classe
     * retourne l'unique objet de la classe
     * @return PDO
     */
    public static function getPdo()
    {
        if (M_AccesDonnees::$monPdo == null) {
            M_AccesDonnees::$monPdo = new PDO(
                M_AccesDonnees::$serveur . ';' . M_AccesDonnees::$bdd,
                M_AccesDonnees::$user,
                M_AccesDonnees::$mdp
            );
            M_AccesDonnees::$monPdo->query("SET CHARACTER SET utf8");
        }
        return M_AccesDonnees::$monPdo;
    }
}
```

modèle d'accès aux données

J'ai ensuite déclaré des méthodes afin de construire des requêtes préparées dans les modèles destinés à gérer ces requêtes en utilisant les méthodes « *prepare* », « *bindParam* » et « *execute* » de *PDO*.

```
50  /**
51  * Préparation d'une requete de lecture
52  * @param string $requete_sql
53  * @return PDOStatement
54  */
55  public static function prepare(String $requete_sql)
56  {
57      return M_AccesDonnees::getPdo()->prepare($requete_sql);
58  }
59
60  /**
61  * @param PDOStatement $statement
62  * @param String $marque
63  * @param mixed $valeur
64  * @param int $pdo_param
65  * @return void
66  */
67  public static function bindParam(PDOStatement $statement, String $marque, $valeur, int $pdo_param)
68  {
69      $statement->bindParam($marque, $valeur, $pdo_param);
70  }
71
72  /**
73  * Execute une requete préparé
74  * @return void
75  */
76  public static function execute(PDOStatement $statement)
77  {
78      $statement->execute();
79  }
```

méthodes pour requête SQL préparé

Et enfin j'ai construit d'autres méthodes permettant de récupérer l'id du dernier élément ajouté à la *BDD* ainsi que de gérer des transactions lorsque plusieurs requêtes sont exécutées. En déclarant une « transaction » à la *BDD* on indique qu'une suite de requêtes *SQL* va être exécutée, enfin on appelle un « commit » pour marquer la fin de cette suite. Les requêtes s'exécutent et si l'une d'elle se solde par une erreur, c'est toute la suite qui est annulée. Cette façon de faire est indispensable lors d'entrée de données sur plusieurs tables afin d'éviter des problèmes d'intégrité de *BDD*.

```

85  /**
86   * Récupère l'id du dernier insert
87   * @return int|false
88   */
89   public static function lastInsertId(): int | false
90   {
91       return M_AccesDonnees::getPdo()->lastInsertId();
92   }
93
94   /**
95   * Lance une transaction PDO
96   * @return void
97   */
98   public static function beginTransaction()
99   {
100      M_AccesDonnees::getPdo()->beginTransaction();
101  }
102
103  /**
104   * Exécute un commit
105   * @return void
106   */
107  public static function commit()
108  {
109      M_AccesDonnees::getPdo()->commit();
110  }

```

méthodes pour récupérer le dernier id inséré et
géré les transactions *SQL*

```

8  class M_Commande
9  {
10
11     /**
12      * Crée une commande
13      *
14      * Crée une commande à partir des arguments validés passés en paramètre;
15      * crée les lignes de commandes dans la table commande_produit à partir du
16      * tableau d'idProduit passé en paramètre
17      * @param int $client_id
18      * @param Array $listeIdproduits
19      * @param Array $quantites_ventes
20      * @param bool $frais_livraison
21      * @return void
22      */
23     public static function creerCommande(
24         $client_id,
25         $listeIdproduits,
26         $quantites_ventes,
27         $frais_livraison
28     ): void {
29         M_AccesDonnees::beginTransaction();
30         $date = new DateTime();
31         $req_commande = "INSERT INTO
32             commandes ( date_livraison,
33                       etat_paiement,
34                       etat_livraison,
35                       frais_livraison,
36                       client_id )
37             VALUES
38             (:date_livraison, 'W', 'W', :frais_livraison, :client_id)";
39         $res = M_AccesDonnees::prepare($req_commande);
40         M_AccesDonnees::bindParam(
41             $res,
42             ':date_livraison',
43             $date->add(DateInterval::createFromDateString('1 day'))->format("Y-m-d"),
44             PDO::PARAM_STR
45         );
46         M_AccesDonnees::bindParam($res, ':frais_livraison', $frais_livraison, PDO::PARAM_BOOL);
47         M_AccesDonnees::bindParam($res, ':client_id', $client_id, PDO::PARAM_INT);
48         M_AccesDonnees::execute($res);
49         $commande_id = M_AccesDonnees::lastInsertId();
50         $i = 0;
51         foreach ($listeIdproduits as $produit_id) {
52             // Ajout d'une ligne de données dans la table commande_produit
53             $req_commande_produit = "INSERT INTO
54                 commande_produit (commande_id, produit_id, quantite_vente)
55                 VALUES
56                 (:commande_id, :produit_id, :quantite_vente)";
57             $res = M_AccesDonnees::prepare($req_commande_produit);
58             M_AccesDonnees::bindParam($res, ':commande_id', $commande_id, PDO::PARAM_INT);
59             M_AccesDonnees::bindParam($res, ':produit_id', $produit_id, PDO::PARAM_INT);
60             M_AccesDonnees::bindParam($res, ':quantite_vente', $quantites_ventes[$i], PDO::PARAM_INT);
61             M_AccesDonnees::execute($res);
62             $i++;
63         }
64         M_AccesDonnees::commit();
65     }
66 }

```

Pour ce qui est des modèles représentant les tables de la *BDD*, chaque méthode gère soit une entrée, soit une lecture. J'utilise les méthodes construites dans mon modèle d'accès aux données. Les paramètres fournis à ces méthodes sont tous filtrés lorsqu'ils parviennent d'entrées utilisateurs via les *verbes HTTP*.

méthode permettant de créer
une commande

L'étape suivante fut la gestion des contrôleurs. Comme dit précédemment j'ai créé un contrôleur général (« index.php ») qui m'a servi à récupérer et filtrer les variables des méthodes *GET* et *POST*. Puis par l'utilisation d'une boucle « switch », j'ai géré les différents cas de retour des *verbes HTTP* afin de faire appel aux contrôleurs nécessaire qui me permettent de requêter ma *BDD*. À la suite de cette boucle, j'ai inclus mon fichier « template.php » qui s'occupe de la gestion des vues.

À l'entrée de ce contrôleur principal, j'ai instancié une « session ». Une session est une « superglobale », c'est une variable globale disponible dans tous les contextes du script, doté d'un identifiant unique, elle permet de stocker des informations sur la visite d'un utilisateur et persiste même après sa sortie du site. Cette session m'a permis de stocker les produits qu'un potentiel client voudrait acheter ainsi que des informations le concernant, comme son id lors de sa connexion me permettant de récupérer ses informations de compte client dans la *BDD*. J'ai décidé d'encapsuler la session au travers d'un contrôleur.

J'ai aussi ajouté un « autoloader » pour mes modèles et contrôleurs. Cela consiste en une fonction allant chercher les fichier requis lors d'un appel à une méthode.

```

1  <?php
2  session_start();
3
4  // Pour afficher les erreurs PHP
5  error_reporting(E_ALL);
6  ini_set("display_errors", 1);
7
8  // Attention : A supprimer en production !!!
9
10 require 'util/autoload.php';
11 require 'util/fonctions.inc.php';
12 require 'util/validateurs.inc.php';
13 autoload();
14
15 $session = new C_Session;
16
17 $uc = filter_input(INPUT_GET, 'uc'); // Use Case
18 $action = filter_input(INPUT_GET, 'action'); // Action
19 $categorie = filter_input(INPUT_GET, 'categorie'); // ID de categorie
20 $couleur = filter_input(INPUT_GET, 'couleur'); // ID de couleur
21 $idProduit = filter_input(INPUT_GET, 'produit'); // ID de produit
22
23 $session->initPanier();
24
25 $formulaireRecu = filter_input(INPUT_POST, "valider");
26 if (isset($formulaireRecu)) {
27     switch ($formulaireRecu) {
28         case "Connexion":
29             $mail = trim(filter_input(INPUT_POST, 'mail_connexion'));
30             $password = filter_input(INPUT_POST, 'password_connexion');
31             break;
32         case "S'inscrire":
33             $nom = trim(strtolower(filter_input(INPUT_POST, 'nom')));
34             $prenom = trim(strtolower(filter_input(INPUT_POST, 'prenom')));
35             $rue = trim(filter_input(INPUT_POST, 'rue'));
36             $ville = trim(strtolower(filter_input(INPUT_POST, 'ville')));
37             $cp = preg_replace('/\s+/', '', filter_input(INPUT_POST, 'cp'));
38             $mail = trim(filter_input(INPUT_POST, 'mail'));
39             $password = filter_input(INPUT_POST, 'password');
40             $password_verify = filter_input(INPUT_POST, 'password_verify');
41             $phone = preg_replace('/\s+/', '', filter_input(INPUT_POST, 'phone'));
42             break;
43         case "modifierInfos":
44             $nom = trim(strtolower(filter_input(INPUT_POST, 'nom')));
45             $prenom = trim(strtolower(filter_input(INPUT_POST, 'prenom')));
46             $rue = trim(filter_input(INPUT_POST, 'rue'));
47             $ville = trim(strtolower(filter_input(INPUT_POST, 'ville')));
48             $cp = preg_replace('/\s+/', '', filter_input(INPUT_POST, 'cp'));
49             $mail = trim(filter_input(INPUT_POST, 'mail'));
50             $password = filter_input(INPUT_POST, 'password');
51             $password_verify = filter_input(INPUT_POST, 'password_verify');
52             $phone = preg_replace('/\s+/', '', filter_input(INPUT_POST, 'phone'));
53             break;
54         case "confirmerCommande":
55             $date_livraison = filter_input(INPUT_POST, "date-livraison");
56             $quantites_ventes = [];
57             foreach ($_POST as $input_name => $quantite_vente) {
58                 $quantite_input = trim(filter_input(INPUT_POST, $input_name));
59                 if ($quantite_input > 0 and estEntier($quantite_input)) {
60                     $quantites_ventes[] = $quantite_input;
61                 }
62             }
63             break;
64         default:
65             break;
66     }
67 }

```

récupération et filtrage du retour des verbes HTTP

```

86 switch ($uc) {
87     case 'accueil':
88         $controleur = new C_Consultation;
89         $lesProduits = $controleur->derniersProduitsSortis();
90         break;
91     case 'boutique':
92         $controleur = new C_Consultation();
93         if ($action == 'voirCategorie') {
94             $lesProduits = $controleur->trouveLesProduitsDeCategorie($categorie);
95         } elseif ($action == 'voirCouleur') {
96             $lesProduits = $controleur->trouveLesProduitsDeCouleur($couleur);
97         } else {
98             $lesProduits = $controleur->tousLesProduits();
99         }
100         $lesCategories = $controleur->toutesLesCategories();
101         $lesCouleurs = $controleur->toutesLesCouleurs();
102         break;
103     case 'produit':
104         $controleur = new C_Consultation;
105         $produit = $controleur->trouveLeProduit($idProduit);
106         $produitDispo = $controleur->produitEstDisponible($idProduit);
107         if (!$produitDispo) {
108             $message = afficheErreur("Désolé, ce produit est en rupture de stock.");
109         }
110         break;

```

switch de index.php

À l'exception de « index.php », tous les contrôleurs sont encapsulés dans des classes. Ces classes sont sans attributs et ne contiennent que des méthodes. Les méthodes appelant des modèles pour exécuter des requêtes de lecture retourne simplement le résultat du modèle appelé. Dans le cas d'écriture dans la *BDD*, ces contrôleurs effectuent des traitements sur les données destinées à l'écriture et envoyées au modèle. Par exemple le contrôleur Client se chargera de la partie espace client du site, il devra questionner à plusieurs reprises les modèles et traiter des données en fonction. Celui-ci devra également « hasher » le mot de passe inséré par un utilisateur lors de la création d'un compte. C'est à dire qu'il va crypter le mot de passe afin de rendre illisible sa lecture dans la *BDD*. Sans cela une simple injection SQL pourrait donner accès à tous les compte client du site. Ce contrôleur se chargera également de vérifier si un utilisateur créant un compte a renseigné une adresse, ville ou code postal déjà existant afin de lier son compte à ces données ou dans le cas contraire les ajouter à la *BDD*.

contrôleur session

```

3 class C_Session
4 {
5     /**
6      * Fonction qui vérifie si le mdp est bon lors de la connexion en fonction du mail
7      * Retourne l'id de l'utilisateur en cas de réussite
8      * Retourne false en cas d'échec
9      * @return int|false
10     */
11     public function verifyMotDePasse(String $mail, String $mdp): int | false
12     {
13         $data = M_Client::getInfoClientParMail($mail);
14         if (!is_array($data)) {
15             return false;
16         } else if (password_verify($mdp, $data['mdp']) and estEntier($data['id'])) {
17             $_SESSION['id'] = $data['id'];
18             return $data['id'];
19         } else {
20             return false;
21         }
22     }
23
24     /**
25      * Fonction qui renvoie l'id de l'utilisateur connecté via la session
26      * Retourne false en cas d'échec
27      * @return int|bool
28     */
29     public static function getIdClient(): int | false
30     {
31         if (isset($_SESSION['id']) and !empty($_SESSION['id'])) {
32             return $_SESSION['id'];
33         } else {
34             return false;
35         }
36     }
37
38     /**
39      * Initialise le panier
40      * Crée une variable de type session dans le cas
41      * où elle n'existe pas
42      * @return void
43     */
44     public function initPanier(): void
45     {
46         if (!isset($_SESSION['produits'])) {
47             $_SESSION['produits'] = array();
48         }
49     }
50 }

```

```

6 class C_Client
7 {
8     /**
9      * Appelle le modele pour inscrire un nouveau client dans la bdd
10     * en fonction des données rentré dans le formulaire d'inscription
11     * @param String $nom
12     * @param String $prenom
13     * @param String $rue
14     * @param String $ville
15     * @param String $cp
16     * @param String $mail
17     * @param String $password
18     * @param String $password_verify
19     * @param String $phone
20     * @return bool
21     */
22     public function inscription($nom, $prenom, $rue, $ville, $cp, $mail, $password, $password_verify, $phone) {
23         if ($password != $password_verify || M_Client::getInfoClientParMail($mail) != false) {
24             return false;
25         } else {
26             // Démarre une transaction
27             M_AccesDonnees::beginTransaction();
28
29             // Vérifie si la ville existe déjà dans la bdd
30             // Si oui ne l'ajoute pas et récupère son id
31             // Si non l'ajoute dans la bdd
32             if (M_Ville::trouveLaVille($ville) == false) {
33                 $livrable = 0;
34                 $ville_id = M_Ville::creerVille($ville, $livrable);
35             } else {
36                 $ville_id = M_Ville::trouveLaVille($ville)['id'];
37             }
38
39             // Vérifie si le code postal existe déjà dans la bdd
40             // Si oui ne l'ajoute pas et récupère son id
41             // Si non l'ajoute dans la bdd
42             if (M_CodePostal::trouveLeCodePostal($cp) == false) {
43                 $cp_id = M_CodePostal::creerCodePostal($cp);
44             } else {
45                 $cp_id = M_CodePostal::trouveLeCodePostal($cp)['id'];
46             }
47
48             // Vérifie si l'adresse existe déjà dans la bdd
49             // Si oui ne l'ajoute pas et récupère son id
50             // Si non l'ajoute dans la bdd
51             if (M_Adresse::trouveAdresse($rue, $ville_id, $cp_id) == false) {
52                 $adresse_id = M_Adresse::creerAdresse($rue, $ville_id, $cp_id);
53             } else {
54                 $adresse_id = M_Adresse::trouveAdresse($rue, $ville_id, $cp_id)['id'];
55             }
56             $password = password_hash($password, PASSWORD_BCRYPT);
57             M_Client::creerCompteClient(
58                 $nom,
59                 $prenom,
60                 $mail,
61                 $password,
62                 $phone,
63                 $adresse_id
64             );
65
66             // Commit la transaction
67             M_AccesDonnees::commit();
68             return true;
69         }
70     }
71 }

```

méthode inscription du controleur client

Comme le mot de passe est hasher, il va falloir utiliser la fonction « password_verify » de *PHP* qui va s'occuper de comparer le mot de passe entré par l'utilisateur, lors de sa connexion au site, au mot de passe récupéré dans la *BDD*. J'ai décidé de faire cela dans le contrôleur s'occupant de la session car j'ai besoin d'y stocker l'id de l'utilisateur quand il se connecte. Faire cela m'évitera de requêter l'id de l'utilisateur connecté à la *BDD* lorsque j'en aurai besoin. Il me suffira de le récupérer via la session.

Comme dit précédemment, je vais placer les id des produits ajoutés au panier par l'utilisateur dans la session, pour cela je passe par le contrôleur Session afin d'y placer un tableau qui contiendra ces id.

Concernant l'affichage des produits, j'ai construit un contrôleur Consultation, qui communiquera principalement avec le modèle Produit (modèle représentant la table produit de la BDD). J'y ai développé des méthodes permettant d'afficher les produits selon leur catégorie ou bien leur couleur, une méthode permettant l'ajout d'un produit au panier, celle-ci appellera une méthode du contrôleur Session, ainsi qu'une méthode vérifiant si un produit est disponible ou non.

L'affichage des produits se fait ensuite dans la vue boutique par une boucle foreach.

```

14 /**
15  * Renvoie les produits de la catégorie demandé
16  * @return Array
17  */
18 public function trouveLesProduitsDeCategorie(int $categorie)
19 {
20     return M_Produit::trouveLesProduitsDeCategorie($categorie);
21 }
22
23 /**
24  * Renvoie les produits de la couleur demandé
25  * @return Array
26  */
27 public function trouveLesProduitsDeCouleur(int $couleur)
28 {
29     return M_Produit::trouveLesProduitsDeCouleur($couleur);
30 }
31
32 /**
33  * Ajoute le produit demandé au panier
34  * @return Array
35  */
36 public function ajouterAuPanier(C_Session $session, int $idProduit)
37 {
38     // Ajoute le produit au panier s'il n'y est pas
39     if (!$session->ajouterProduitSession($idProduit) and $idProduit > 0) {
40         return "Ce produit est déjà dans le panier.";
41     } else {
42         return "Ce produit a été ajouté au panier.";
43     }
44 }
45
46 /**
47  * Vérifie la disponibilité d'un produit
48  * @param int
49  * @return bool
50  */
51 public function produitEstDisponible(int $idProduit): bool
52 {
53     $produit = M_Produit::trouveLeProduit($idProduit);
54     foreach ($produit as $ligneProduit) {
55         if ($ligneProduit["quantite_stock"] < $ligneProduit["quantite_fleur"]) {
56             return false;
57         }
58     }
59     return true;
60 }

```

contrôleur consultation

```

92 /**
93  * Retourne sous forme d'un tableau associatif tous les produits
94  * de la catégorie passée en argument
95  *
96  * @param $idCategorie
97  * @return Array un tableau associatif
98  */
99 public static function trouveLesProduitsDeCategorie($idCategorie)
100 {
101     $req = "SELECT
102         nom_produit, produits.id AS produit_id, prix_vente
103     FROM
104         produits
105     JOIN
106         categories ON categorie_id = categories.id
107     WHERE
108         categorie_id = '$idCategorie'";
109     $res = M_AccesDonnees::prepare($req);
110     M_AccesDonnees::execute($res);
111     $lesLignes = $res->fetchAll(PDO::FETCH_ASSOC);
112     return $lesLignes;
113 }
114
115 /**
116  * Retourne sous forme d'un tableau associatif tous les produits
117  * de la catégorie passée en argument
118  *
119  * @param int $idCouleur
120  * @return Array un tableau associatif
121  */
122 public static function trouveLesProduitsDeCouleur(int $idCouleur)
123 {
124     $req = "SELECT
125         nom_produit, produit_id, prix_vente
126     FROM
127         produits
128     JOIN
129         fleur_produit ON produit_id = produits.id
130     JOIN
131         fleurs ON fleur_id = fleurs.id
132     JOIN
133         couleurs ON couleur_id = couleurs.id
134     WHERE
135         couleur_id = '$idCouleur'";
136     $res = M_AccesDonnees::prepare($req);
137     M_AccesDonnees::execute($res);
138     $lesLignes = $res->fetchAll(PDO::FETCH_ASSOC);
139     return $lesLignes;
140 }

```

```

43 <section class="section-boutique">
44     <?php
45     foreach ($lesProduits as $unProduit) :
46         $idProduit = $unProduit['produit_id'];
47         $nomProduit = $unProduit['nom_produit'];
48         $prixVente = $unProduit['prix_vente'];
49         $produitDispo = $controleur->produitEstDisponible($idProduit);
50     ?>
51     <article class="card">
52         <a href="index.php?uc=produit&produit=?= $idProduit ?>">
53             
56         </a>
57         <span class="info-produit-card">
58             <p class="nom-produit-card"><?= $nomProduit; ?> </p>
59             <p class="text-center"><?= $prixVente; ?> €</p>
60             <?php if ($produitDispo) : ?>
61                 <img data-id=?= $idProduit ?>
62                 class="logo-panier add-panier"
63                 src="public/img/panier.svg"
64                 alt="logo de panier">
65             <?php else : ?>
66                 <img data-id=?= $idProduit ?>
67                 class="logo-panier add-panier produit-indisponible"
68                 src="public/img/panier.svg"
69                 alt="logo de panier">
70             <?php endif; ?>
71         </span>
72     </article>
73     <?php endforeach ?>
74 </section>

```

affichage des produits dans la vue boutique

modèle produit

Afin d'améliorer l'expérience utilisateur, j'ai décidé de développer une requête *AJAX* concernant l'ajout au panier d'un produit. Lorsqu'un utilisateur cliquera sur le bouton prévu à cet effet, une modale s'ouvrira pour l'avertir si le produit a bien été ajouté au panier, s'il est déjà dans le panier ou alors si celui-ci est en rupture de stock. J'ai donc utilisé la fonction « fetch » via la méthode *POST* dans laquelle j'ai fourni l'id du produit. J'ai créé un fichier *PHP* que la requête *AJAX* va cibler. Ce fichier *PHP* va traiter la demande d'ajout au panier puis retourner un message sous forme de chaîne de caractères traduit au format *JSON* en réponse à la requête *AJAX*. Enfin, en fonction du retour, le message s'affichera dans une modale.

```

10 // récupération des données JSON envoyées
11 $request_data = json_decode(file_get_contents('php://input'));
12
13 // vérification de l'action demandée
14 if ($request_data->action === 'get_data') {
15     $controleur = new C_Consultation();
16     $idProduit = $request_data->parameter1;
17     $produitDispo = $controleur->produitEstDisponible($idProduit);
18
19     if ($produitDispo) {
20         // appel de la fonction get_data avec les paramètres fournis
21         $message = $controleur->ajouterAuPanier($session, $idProduit);
22
23         // retourner les données encodées en JSON
24         header('Content-Type: application/json');
25         echo json_encode($message);
26     } else {
27         // retourner les données encodées en JSON
28         header('Content-Type: application/json');
29         echo json_encode("Désolé ce produit est en rupture de stock.");
30     }
31 }

```

fichier PHP destiné à la requête AJAX

```

1 const addPanier = document.querySelectorAll(".add-panier");
2 addPanier.forEach((element) => {
3     element.addEventListener("click", (event) => {
4         fetch("util/fetch.php", {
5             method: "POST",
6             body: JSON.stringify({
7                 action: "get_data",
8                 parameter1: event.target.dataset.id,
9             }),
10            headers: {
11                "Content-Type": "application/json",
12            },
13        })
14        .then((response) => {
15            if (!response.ok) {
16                throw new Error("Erreur HTTP, code : " + response.status);
17            }
18            return response.json();
19        })
20        .then((message) => {
21            console.log(message);
22            alert(message);
23        })
24        .catch((error) => {
25            alert("La requête a échoué, veuillez réessayer ou recharger la page.");
26            console.error(error);
27        });
28    });
29 });

```

requête AJAX

Afin de gérer l'affichage, j'ai créé un template en *php* qui selon la variable « uc » (pour Use Case), récupéré dans l'URL via la méthode *GET*, s'occupera de renvoyer le bon affichage demandé. Dans ce template est présent l'entête du document comprenant la balise *head* (cf page 14). Celle-ci contient le titre du site, l'encodage des caractères, une balise forçant la compatibilité des navigateurs « internet explorer » par celle du navigateur « Edge », une autre ajustant l'affichage des éléments en fonction de la taille de l'écran, les liens des fichiers *CSS* et *JavaScript* ainsi qu'une balise de sécurité limitant les attaques *XSS* (*Cross-Site Scripting*). Ce fichier « template.php » contient également l'en-tête et le pied de page du site LaFleur.

J'ai utilisé une boucle « switch » pour la gestion de la variable contenue dans l'URL. En fonction de sa valeur, qui est une chaîne de caractères, la boucle inclura le bon fichier de vue. (cf page 13)

Pour terminer, j'ai adapté mes vues en fonction de la maquette et les utilisateurs seront prévenus par un message sur les vues suivantes :

- panier : lorsque ceux-ci ne seront pas connectés, si le panier est vide ou contient des articles et si leur ville n'est pas desservie
- espace client : lorsque leur ville n'est pas desservie, lorsque leur commande est invalide (ville non desservie, informations renseignées non valide), tant qu'il n'ont passé aucune commande
- connexion : lorsqu'une ou plusieurs informations renseignées sont invalides et leur précise pourquoi

Il faut saisir le champ Nom.
Il faut saisir le champ Prénom.
Il faut saisir le champ Rue.
Il faut saisir le champ Ville.
Erreur de code postal. Format attendu : "34000".
Erreur de mail. Format attendu : "exemple@domaine.com".
Les mots de passe ne correspondent pas.
Erreur de téléphone. Format attendu : "0615273849".

messages page connexion / inscription

Connectez-vous pour passer commande.

Votre panier est vide.

Venez visiter notre boutique !

Boutique

messages panier

Blog

Pour la réalisation du blog, j'ai utilisé *WordPress*, c'est un *CMS (Content Management System)*. C'est une application web servant à créer des sites simplement et rapidement. Cela fonctionne sous forme de bloc sémantique à placer (paragraphes, images, liens, liste, etc..) puis il ne reste plus qu'à y insérer du contenu. Des thèmes graphiques sont proposés afin de gagner du temps et d'économiser de l'argent pour les personnes ayant besoin d'un site simple avec peu de ressources. Ces thèmes peuvent être gratuits ou payants et sont presque entièrement personnalisables. Il est aussi possible d'y intégrer des « plugins », ce sont des extensions permettant d'ajouter des fonctionnalités. Grâce à cela il est possible d'augmenter le contenu d'un site *CMS* et de passer de la réalisation de sites très simple, comme un blog, à un site e-commerce complet et plus complexe. En bref un *CMS* est une sorte de logiciel permettant de créer un site rapidement et reste accessible même à des non développeurs.

J'ai fais le choix de prendre un thème gratuit mais esthétique mettant en avant les photos du blog sur la page d'accueil. J'ai rédigé quelques articles basiques faisant office de démonstration des possibilités de *WordPress* aux clients. C'est eux-même qui rajouteront du contenu par la suite. J'ai rédigé un article sur le prochain évènement accompagné d'une publicité pour la boutique en ligne ainsi qu'une présentation de la loterie. J'ai aussi développé un plugin, nommé « *SEO Highlighter* », leur permettant d'insérer des mots dans un champ de saisie, ces mots seront ajoutés dans une table de la base de donnée du blog. Ceux-ci passeront alors automatiquement en gras dans toutes les pages et tout les articles du blog. Cela leur permettra d'augmenter leur référencement naturel, c-à-d que le site deviendra mieux classé par les moteurs de recherche.

SEO Highlighter

Mots-clefs

Ce plugin sert à marquer les mots que vous penserez important pour vos articles et/ou votre site dans sa globalité.

Ces mots ainsi marqués contribueront à améliorer la visibilité de votre site en le faisant remonter dans la liste de résultats de recherche lorsque des internautes chercheront ces mots dans leur navigateur.

Ils passeront aussi automatiquement en gras sur votre site.

Infos : ne fonctionne pas sur les mots avec apostrophes, ex: aujourd'hui

Voici les mots déjà enregistrés :

Lafleur, LaFleur, fleur, fleurs, rose, roses, tulipe, tulipes, pivoine, pivoines

Entrer une liste de mots à mettre en valeur, séparés par une virgule.

Save Changes

SEO Highlighter pugin

Ce plugin fonctionne comme suit :

1. Il ajoute un lien dans l'onglet réglage du panneau administrateur de *WordPress* menant au menu de gestion de l'extension.
2. Il crée une table dans la base de données du site *WordPress* contenant 2 colonnes nommé « id » et « keyword ». La colonne « keyword » est de type « VARCHAR(255) ».
3. Lorsqu'un mot est entré par l'admin, celui-ci est ajouté dans la table. Les mots enregistrés dans cette table sont affichés dans la page de gestion de l'extension.
4. Afin de passer les mots en gras, le plugin va récupérer les données de la table dans un tableau puis à l'aide d'une *regex*, il va les encadrer d'une balise *strong* accompagné de la propriété *bold* de CSS permettant de mettre en gras ce mot.

Une *regex*, ou expression régulière, est une chaîne de caractères particulière décrivant un ensemble de chaînes de caractères. Elle se construit en utilisant des caractères spéciaux ayant une sémantique logique. Cela sert à vérifier n'importe quelle chaîne de caractères afin de savoir si elle correspond bien à ce qu'on attends d'elle dans le développement. On va utiliser des *regex* afin de vérifier si une chaîne de caractères correspond bien à une adresse mail, numéro de téléphone, nombre, date, etc..

En utilisant une expression régulière j'ai fais en sorte que l'utilisateur ne puisse pas entrer de nombre ou de caractères spéciaux. Il est possible d'ajouter plusieurs mots a la fois en les séparant d'une virgule. Un mot déjà présent dans la base de données ne pourra pas être ajouté une deuxième fois.

```
51 // Valide ou non les mots rentrée par l'utilisateur en n'acceptant uniquement les
    lettres, virgule, accents et caractères alphabétique spéciaux
52 function validateVariable($variable)
53 {
54     // regex match lettre min ou maj, avec accent, caractère alphabétique spéciaux
    (ex: œ), virgule seulement si entouré de lettres ou lettres + espace
55     $regex = "/^[a-zA-ZÀ-ÖØ-öø-ÿœ]+(?:,[\s?][a-zA-ZÀ-ÖØ-öø-ÿœ]+)*$/";
56     return preg_match($regex, trim($variable));
57 }
```

regex utilisé dans mon plugin

4. Sécurité

Voici une liste des failles pour lesquelles j'ai implémenté des protections :

- *Cross-Site Request Forgery (CSRF)* : cette attaque consiste à usurper l'identité d'un utilisateur connecté au travers d'une requête *HTTP* falsifié afin d'utiliser ses droits de façon malveillante sur le site, contournant ainsi le système de sécurité d'identification. Afin de se protéger contre ce type d'attaque il est nécessaire d'associer des *token*, c-à-d des numéros d'identification unique, aux requête *HTTP* d'un site. Cela permet, lors de la réception de cette requête, de vérifier si celle-ci est sûr à travers ce jeton d'identification unique. Ces *token* sont créé par chiffrement d'un identifiant utilisateur.

- *Cross-Site Scripting (XSS)* : cette attaque vise à envoyer un code client malveillant sur un serveur web afin d'attaquer un utilisateur en lançant ce script côté navigateur lorsque celui-ci se connecte sur un site. Afin de limiter ce type d'attaque il est possible d'interdire à tout autre site extérieur d'accéder à notre site web, à l'exception une sélection de site de confiance.
- *Injection SQL* : ce type d'attaque vise les bases de données, le but étant d'injecter une requête *SQL* non voulu par le système afin de compromettre la sécurité de la *BDD* ou de voler des informations. Afin de se prévenir de cela, il faut impérativement créer ses requête avec des requêtes préparées et paramétrées.

```

9      <form action="{ route('commandes.update', $commande->id) }}" method="POST">
10      @method('PUT')
11      @csrf

```

Jeton CSRF dans le panneau admin

Afin d'augmenter la sécurité j'ai également créé des validateurs d'input, afin que la valeurs des champs de saisie correspondent bien aux données demandées.

```

69 function infosValide(String $nom, String $prenom, String $rue, String $ville, String $cp,
String $mail, String $password, String $password_verify, String $phone): String
70 {
71     $erreurs = "";
72     if ($nom == "") {
73         $erreurs = $erreurs . "Il faut saisir le champ Nom.<br/>";
74     } else if (strlen($nom) > 190) {
75         $erreurs = $erreurs . "Le champ Nom ne peut contenir que 190 caractères.<br/>";
76     }
77     if ($prenom == "") {
78         $erreurs = $erreurs . "Il faut saisir le champ Prénom.<br/>";
79     } else if (strlen($prenom) > 50) {
80         $erreurs = $erreurs . "Le champ Prénom ne peut contenir que 50 caractères.<br/>";
81     }
82     if ($rue == "") {
83         $erreurs = $erreurs . "Il faut saisir le champ Rue.<br/>";
84     } else if (strlen($rue) > 190) {
85         $erreurs = $erreurs . "Le champ Rue ne peut contenir que 190 caractères.<br/>";
86     }
87     if ($ville == "") {
88         $erreurs = $erreurs . "Il faut saisir le champ Ville.<br/>";
89     } else if (strlen($ville) > 190) {
90         $erreurs = $erreurs . "Le champ Ville ne peut contenir que 190 caractères.<br/>";
91     }
92     if ($cp == "") {
93         $erreurs = $erreurs . "Il faut saisir le champ Code postal.<br/>";
94     } else if (!estUnCp($cp)) {
95         $erreurs = $erreurs . "Erreur de code postal. Format attendu : \"34000\".<br/>";
96     }
97     if ($mail == "") {
98         $erreurs = $erreurs . "Il faut saisir le champ Email.<br/>";
99     } else if (!estUnMail($mail)) {
100         $erreurs = $erreurs . "Erreur de mail. Format attendu : \"exemple@domaine.com\".<br/>";
101     } else if (strlen($mail) > 190) {
102         $erreurs = $erreurs . "Le champ Email ne peut contenir que 190 caractères.<br/>";
103     }
104     if ($password == "") {
105         $erreurs = $erreurs . "Il faut saisir le champ Mot de passe.<br/>";
106     }
107     if ($password != $password_verify) {
108         $erreurs = $erreurs . "Les mots de passe ne correspondent pas.<br/>";
109     }
110     if ($phone == "") {
111         $erreurs = $erreurs . "Il faut saisir le champ Téléphone.<br/>";
112     } else if (!estUnTel($phone)) {
113         $erreurs = $erreurs . "Erreur de téléphone. Format attendu : \"0615273849\".<br/>";
114     }
115     return $erreurs;

```

```

20 * teste si une chaîne est un entier
21 *
22 * Teste si la chaîne ne contient que des chiffres
23 * @param String $valeur : la chaîne testée
24 * @return : vrai ou faux
25 */
26 function estEntier(String $valeur)
27 {
28     return preg_match("/^[0-9]/", $valeur) == 0;
29 }

```

```

32 * Teste si une chaîne a le format d'un mail
33 *
34 * Utilise les expressions régulières
35 * @param String $mail : la chaîne testée
36 * @return : vrai ou faux
37 */
38 function estUnMail(String $mail)
39 {
40     return preg_match("#^[\\w.-]+@[\\w.-]+\\.
[a-zA-Z]{2,6}$#", $mail) == 1;
41 }

```

```

44 * teste si une chaîne a un format de téléphone
français
45 *
46 * Teste le nombre de caractères de la chaîne
et le type entier (composé de chiffres)
47 * @param String $phone : la chaîne testée
48 * @return : vrai ou faux
49 */
50 function estUnTel(String $phone)
51 {
52     return strlen($phone) == 10 && estEntier
($phone) && $phone != "0000000000";
53 }

```

VI - Tests

1. Tests unitaire

Les tests unitaire servent à vérifier si les fonctions développées renvoient bien les bons type de variables suivant les différents cas que cette fonction retourne. J'ai passé mes test unitaires avec *phpunit*, pour cela j'ai utilisé *Composer* afin d'installer *phpunit*.

```
1  <?php
2
3  declare(strict_types=1);
4
5  use PHPUnit\Framework\TestCase;
6
7  require 'util/autoload.php';
8  autoload();
9  session_start();
10
11 final class C_SessionTest extends TestCase
12 {
13     public function testSiIdClientNestPasDansSession(): void
14     {
15         $session = new C_Session();
16
17         $verifMdp = $session->getIdClient();
18
19         $this->assertFalse($verifMdp);
20     }
21
22     public function testSiPasswordCorrespondEmail(): void
23     {
24         $session = new C_Session();
25
26         $verifMdp = $session->verifMotDePasse('test@test.com', '123456');
27
28         $this->assertIsNumeric($verifMdp);
29     }
30
31     public function testSiPasswordNeCorrespondPasEmail(): void
32     {
33         $session = new C_Session();
34
35         $verifMdp = $session->verifMotDePasse('test@test.com', '654321');
36
37         $this->assertFalse($verifMdp);
38     }
39
40     public function testSiIdClientEstDansSession(): void
41     {
42         $session = new C_Session();
43
44         $verifMdp = $session->getIdClient();
45
46         $this->assertIsNumeric($verifMdp);
47     }
48 }
```

test sur contrôleur session

```
1  <?php
2
3  declare(strict_types=1);
4
5  use PHPUnit\Framework\TestCase;
6
7  require 'util/validateurs.inc.php';
8
9
10 final class ValideurTest extends TestCase
11 {
12     public function testEstUnCp(): void
13     {
14         $cp = "34090";
15
16         $estUnCp = estUnCp($cp);
17
18         $this->assertTrue($estUnCp);
19     }
20
21     public function testNestPasUnCp(): void
22     {
23         $cp = "00000";
24
25         $estUnCp = estUnCp($cp);
26
27         $this->assertFalse($estUnCp);
28     }
29
30     public function testEstUnMail(): void
31     {
32         $mail = "test@test.com";
33
34         $estUnMail = estUnMail($mail);
35
36         $this->assertTrue($estUnMail);
37     }
38
39     public function testNestPasUnMail(): void
40     {
41         $mail = "test@123.j";
42
43         $estUnMail = estUnMail($mail);
44
45         $this->assertFalse($estUnMail);
46     }
47 }
```

test sur fonction de validation

```
PS D:\Logiciels\Xampp\htdocs\www\la_fleur\lafleur> ./vendor/bin/phpunit tests
PHPUnit 10.0.0 by Sebastian Bergmann and contributors.

Runtime:      PHP 8.1.10

.....
8 / 8 (100%)

Time: 00:00.558, Memory: 6.00 MB

OK (8 tests, 8 assertions)
```

résultats des tests

2. Validation W3C et test de performance

Press the Message Filtering button to collapse the filtering options and error/warning/info counts.

Message Filtering

Errors (12) · [Hide all errors](#) · [Show all errors](#)

- 1 Element `<div>` not allowed as child of element `<div>` in this context. (Suppressing further errors from this subtree.) (11) · [Hide all](#) · [Show all](#)
 - 1.1 ☒ Element `<a>` not allowed as child of element `<u>` in this context. (Suppressing further errors from this subtree.)
 - 1.2 ☒ Element `<p>` not allowed as child of element `` in this context. (Suppressing further errors from this subtree.) (10)
- 2 ☒ Duplicate attribute `target`

Warnings (5) · [Hide all warnings](#) · [Show all warnings](#)

- 1 ☒ Article lacks heading. Consider using `<h2>` or `<h6>` elements to add identifying headings to all articles. (5)

Info messages (5) · [Hide all info messages](#) · [Show all info messages](#)

- 1 ☒ Trailing slash on void elements has no effect and interacts badly with unquoted attribute values. (5)

validation W3C non acceptable

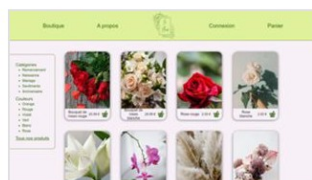
Press the Message Filtering button to collapse the filtering options and error/warning/info counts.

Message Filtering

Info messages (5) · [Hide all info messages](#) · [Show all info messages](#)

- 1 ☒ Trailing slash on void elements has no effect and interacts badly with unquoted attribute values. (5)

validation W3C réussie



Latest Performance Report for:

https://verlyck.needemand.com/projets_web/lafleur/index.ph...

Report generated: Wed, May 10, 2023 11:36 AM -0700
Test Server Location: Vancouver, Canada
Using: Chrome (Desktop) 103.0.5060.134, Lighthouse 9.6.4

GTmetrix Grade ?

A	Performance ? 92%	Structure ? 87%
----------	-----------------------------	---------------------------

Web Vitals ?

Largest Contentful Paint ? 1.5s	Total Blocking Time ? 0ms	Cumulative Layout Shift ? 0.07
---	-------------------------------------	--

test de performance

VII - Veilles technologiques

1. Tailwind

Tailwind est un framework « utility-first » CSS open source. Il offre une grande variété de classes prédéfinies représentant un unique couple de propriété et valeur CSS. Ces classes préfabriquées peuvent être directement écrites dans les balises *HTML* afin d'y appliquer le style voulu.

L'avantage ce framework est qu'il n'y a plus de code CSS à développer, celui-ci est déjà écrit. Cela permet de gagner du temps de développement car il n'y a plus besoin de penser à un bon nommage de classes ni de les coder. *Tailwind* permet aussi de « purger » le CSS, cela va analyser le code pour savoir quelles classes sont utilisées et supprimer celles non utilisées afin d'alléger l'application lors de la mise en production. Il est aussi possible de créer des classes personnalisables à partir des éléments de base du framework, ce qui en fait une technologie flexible.

L'inconvénient est qu'il alourdi le code *HTML*, celui-ci devient difficile à lire et si l'on ne maîtrise pas bien le CSS cela peut devenir fastidieux d'appliquer le style voulu avec le minimum de règles CSS. En bref c'est un outil très intéressant mais il demande un certain temps d'apprentissage pour le maîtriser et en sortir le plein potentiel.

Ce framework peut être installé via un lien *CDN* ou par un gestionnaire de paquet comme *npm*.

2. Select2 et JQuery

Select2 est une librairie JavaScript basé sur *JQuery*. Cette librairie permet de modifier l'apparence, les fonctionnalités de la balise *HTML* « select » et y ajoute un champ de recherche. Pour les balises contenant l'attribut « multiple », elle améliore grandement l'expérience utilisateur en donnant la possibilité de rechercher et ajouter les choix un par un. Ces options s'affichent accompagnées d'un bouton servant à retirer l'option sélectionnée.

JQuery est aussi une librairie JavaScript. Elle ajoute des fonctionnalités afin de faciliter la navigation dans le *DOM* (*Document Object Model*), des *EventListener* ou encore des effets d'animations. Le *DOM* est une interface de programmation pour les documents *HTML*, il représente les éléments de ce type de document comme un arbre constitué de nœuds et d'objets ayant chacun des propriétés et méthodes.

JQuery fût très populaire à l'époque où la navigation dans le *DOM* avec du pur *JavaScript* était compliqué. Depuis de nouvelles fonctionnalités natives à JavaScript ont simplifié cette navigation ce qui rends *JQuery* moins populaire.

```
1  $(document).ready(function () {  
2      $(".select2").select2();  
3  });  
4
```

application de *select2* aux balises *select* contenant la classe *select2*

VIII - Documentation anglophone

Mes recherches sur sites anglophones les plus importantes ont été faites pour *Laravel*. J'ai eu des difficultés pour comprendre comment définir correctement la relation *ManyToMany* et le fonctionnement des tables dites « pivot » et leurs modèles associés.

```
# Model Structure

Many-to-many relationships are defined by writing a method that returns the
result of the belongsToMany method. The belongsToMany method is provided by the
Illuminate\Database\Eloquent\Model base class that is used by all of your
application's Eloquent models. For example, let's define a roles method on our
User model. The first argument passed to this method is the name of the related
model class:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

class User extends Model
{
    /**
     * The roles that belong to the user.
     */
    public function roles(): BelongsToMany
    {
        return $this->belongsToMany(Role::class);
    }
}

Once the relationship is defined, you may access the user's roles using the roles
dynamic relationship property:

use App\Models\User;

$user = User::find(1);

foreach ($user->roles as $role) {
    // ...
}
```

Comme vous l'avez appris, manipuler des relations plusieurs-à-plusieurs nécessite la présence d'une table intermédiaire. *Eloquent* offre des solutions d'interactions très pratique entre ces tables. Pour l'exemple, supposons que notre modèle « User » est relié à de nombreux modèles « Role ». Après avoir accédé à cette relation, on peut parcourir la table intermédiaire en utilisant l'attribut « pivot » sur ces modèles : *exemple de code*

Remarquez que pour chaque modèle « Role » que nous parcourons, un attribut « pivot » y est automatiquement assigné. Cet attribut contient un modèle représentant la table intermédiaire. Par défaut, seul les clés primaires des modèles seront présentes dans le modèle « pivot ». Si votre table intermédiaire contient des attributs supplémentaires, vous devez les spécifier lorsque vous définissez la relation.

Les relations plusieurs-à-plusieurs sont définies par l'écriture d'une méthode retournant le résultat de la méthode « `belongsToMany` ». Cette méthode est fournie par la classe « `Illuminate\Database\Eloquent\Model` » utilisée par toute les applications basées sur les modèles d'*Eloquent*. Pour l'exemple, définissons une méthode « `roles` » dans notre modèle « `User` ». Le premier argument passé dans cette méthode est le nom de la classe de son modèle associé : *exemple de code*

Une fois la relation définie, vous pouvez accéder aux rôles des utilisateurs en passant par la propriété de relation dynamique « `roles` » : *exemple de code*

```
# Retrieving Intermediate Table Columns

As you have already learned, working with many-to-many relations requires the
presence of an intermediate table. Eloquent provides some very helpful ways of
interacting with this table. For example, let's assume our User model has many
Role models that it is related to. After accessing this relationship, we may access
the intermediate table using the pivot attribute on the models:

use App\Models\User;

$user = User::find(1);

foreach ($user->roles as $role) {
    echo $role->pivot->created_at;
}

Notice that each Role model we retrieve is automatically assigned a pivot
attribute. This attribute contains a model representing the intermediate table.

By default, only the model keys will be present on the pivot model. If your
intermediate table contains extra attributes, you must specify them when defining
the relationship:

return $this->belongsToMany(Role::class)->withPivot('active', 'created_by');
```