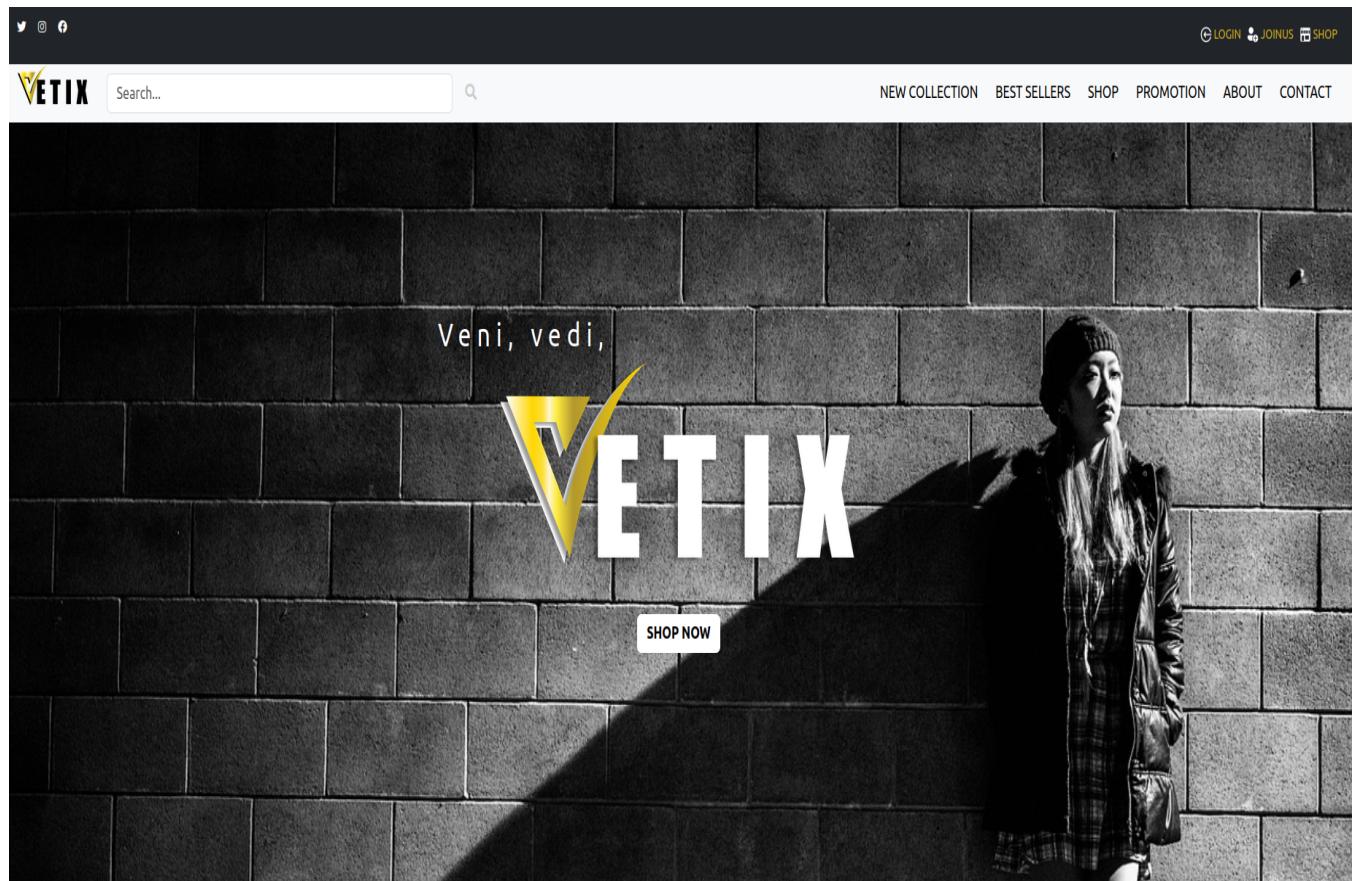


Titre : Développeur Web et Web Mobile



Jérémie Nowak

Table des matières

1.Introduction.....	2
2.Compétences du référentiel couverte par le projet.....	3
3.Fonctionnalités.....	3
1.Front end.....	3
2.Back end.....	4
4.Réalisations.....	4
1.Conception de la base de donnée.....	4
-MCD:.....	4
-MLD:.....	5
-MPD:.....	6
-Charte graphique:.....	7
-Maquette basse fidélité:.....	8
-Maquette haute fidélité:.....	9
1.Authentification.....	10
2.Favoris.....	15
3.Commentaire.....	19
4.Panel admin.....	21
6.Conclusion.....	23

1. Introduction

Durant ma formation j'ai eu l'occasion de travailler pour un projet de groupe sur le thème de la boutique en ligne avec deux de mes camarades Thomas SPINEC et Nadia HAZEM . Ce projet a duré 1 mois et nous a permis d'apprendre à travailler à plusieurs sur un projet "long".

Pour notre projet nous avons décidé de créer un site de prêt à porter pour femme. Sur ce site, vous pouvez y trouver plusieurs types de vêtements . Cela va des robes en passant par des pantalons, des pulls et même du loungewear. Avec cela vous trouverez aussi des accessoires, quelques sacs , paire de lunettes et même des montres.

2. Compétences du référentiel couverte par le projet

Ce projet m'a permis de valider ces compétences :

Pour l'activité 1, **"Développer la partie front-end d'une application web et web mobile en intégrant les recommandations de sécurité"**:

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

Pour l'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.

3.Fonctionnalités

1.Front end

- Menu burger: Le menu burger apparaîtra à partir de 766 px afin d'offrir une expérience utilisateur sur tablette et téléphone plus intéressante.

- Accès au panier afin de visualiser les articles qui ont été sélectionnés, leurs quantité, leurs prix et avoir la possibilité de supprimer ces mêmes articles.

- Affichage de la section "New collection" qui sera gérée par un booléen en base de données.

2.Back end

- Identification / inscription : Cela permet de créer ou de vérifier les données de connexion afin de pouvoir accéder au panier ainsi qu'aux favoris.

- Le favoris: La possibilité d'ajouter un produit en favoris lorsque l'utilisateur est connecté et qui pourra, plus tard, être ajouté au panier via la partie "profile".

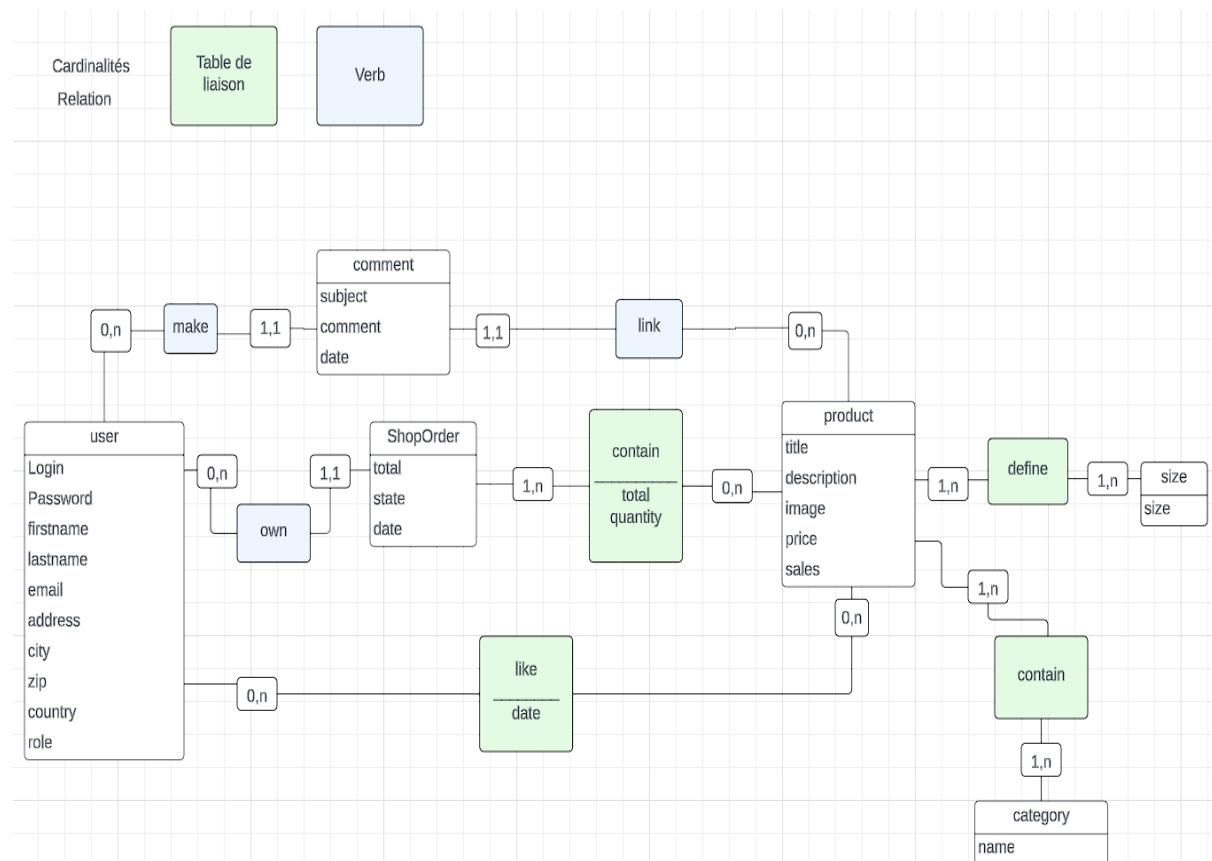
- Le panel administrateur: Qui permet de gérer les rôles ou même supprimer des utilisateurs. Gérer les produits par l'ajout de quantité, de taille ou d'images (jusqu'à 3) de stock, ainsi que les prix. Et pour finir l'ajout ou suppression de catégories.

4. Réalisations

1. Conception de la base de donnée

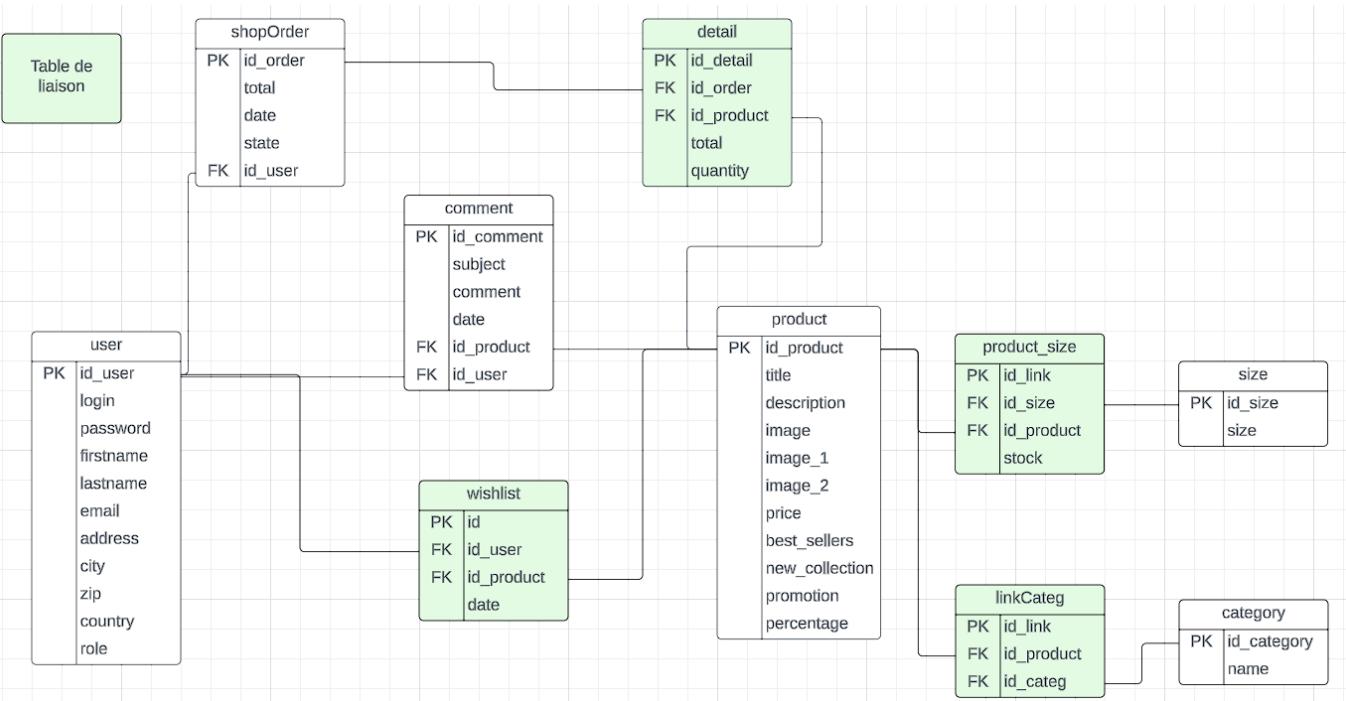
-MCD:

Sur le model conceptuel de donnée vous trouverez les cardinalités, les verbes et le nom des tables



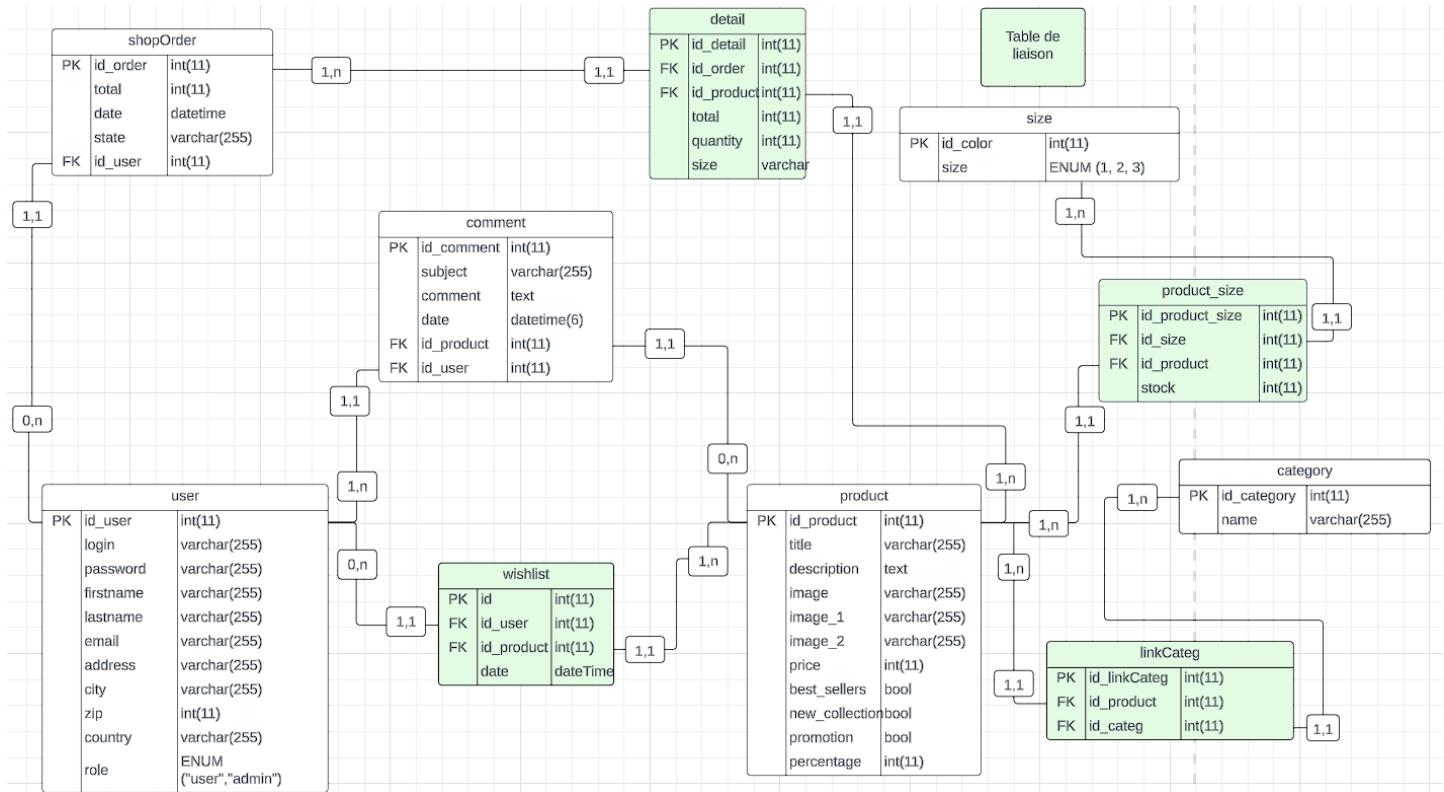
-MLD:

Sur le modèle logique de donnée vous trouverez les clés primaire et secondaire ainsi que les tables de liaisons ici représenté par la couleur verte.



-MPD:

Sur le modèle physique de données vous trouverez l'ensemble des informations comprises dans le MCD et le MLD. Ce modèle représente la base de données. On y retrouve donc les verbes, les cardinalités, les clefs primaires, les clés étrangères ainsi que le type (int, varchar, text....)



Ici la table user et la table product sont les deux entités principales. La table user permettra de faire le lien entre le panier, ici shopOrder, les commentaires avec la table comment ainsi que la table wishlist qui gérera les favoris.

Pour la table product sera lié aux différentes informations importantes afin de gérer son affichage. Il y aura donc la table category lié par une table de liaison appelé linkCateg, la table size qui sera lié par product_size et pour finir shopOrder lié elle par la table detail.2. Conception de la charte graphique

-Charte graphique:

Une étape très importante car elle permet d'harmoniser le travail de chacune des personnes travaillant sur le projet. On y retrouvera donc des variables de couleurs afin de gagner du temps, les différentes typographies, les logos à utiliser, la taille des polices ainsi que la normalisation des bouton

TYPOGRAPHY

Style Guide

BUTTONS

TYPOGRAPHY

Montserrat: Default font for the entire communication. It is a sans-serif font with a rounded, friendly feel. It is clean and modern, making it suitable for both digital and print applications. It has a consistent weight across all characters, making it easy to read at different sizes.

ROBOTO: Secondary font. It is a clean, sans-serif font with a modern and minimalist feel. It is well-suited for digital applications like web pages, mobile apps, and emails. It has a consistent weight across all characters, making it easy to read at different sizes.

HEADING

Font	Font Style	Heading
Montserrat	Normal	Heading
Montserrat	Oblique	Heading
Montserrat	Normal	Heading
Montserrat	Oblique	Heading
Montserrat	Normal	Heading
Montserrat	Oblique	Heading
Montserrat	Normal	Heading
Montserrat	Oblique	Heading

BUTTONS

DESKTOP

MOBILE

LOGO

Section 1

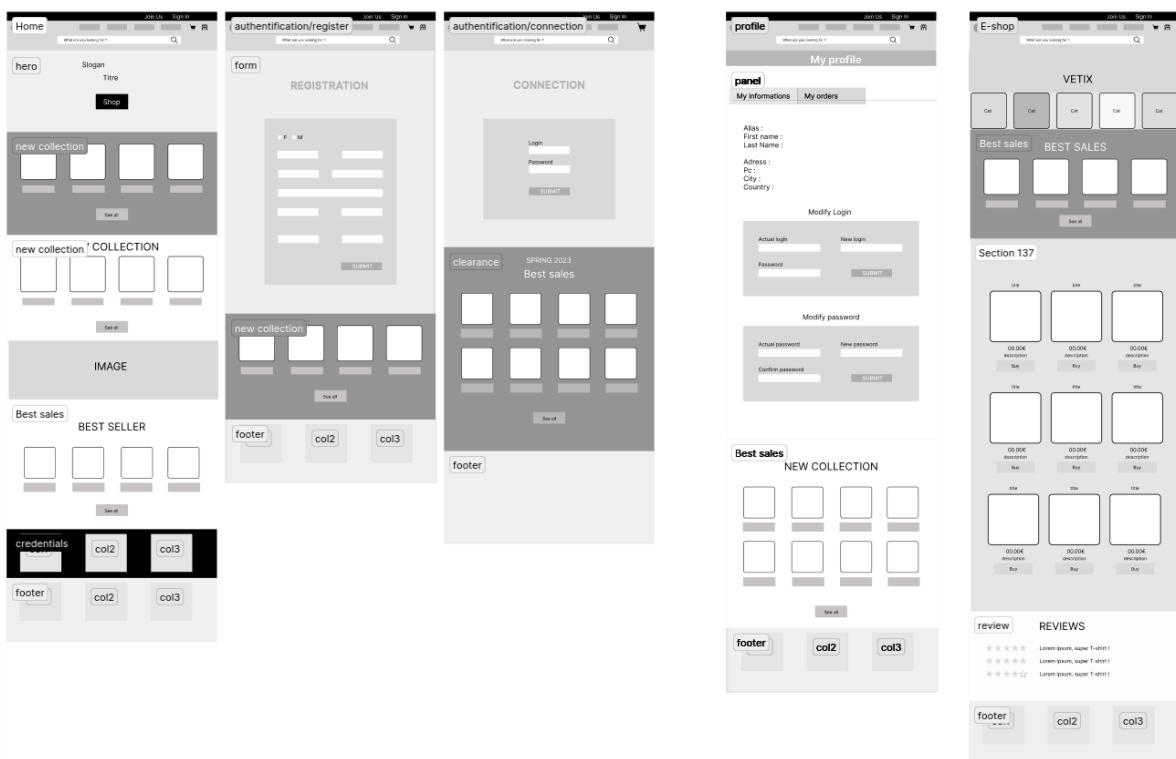
Section 2

LOGO

s.

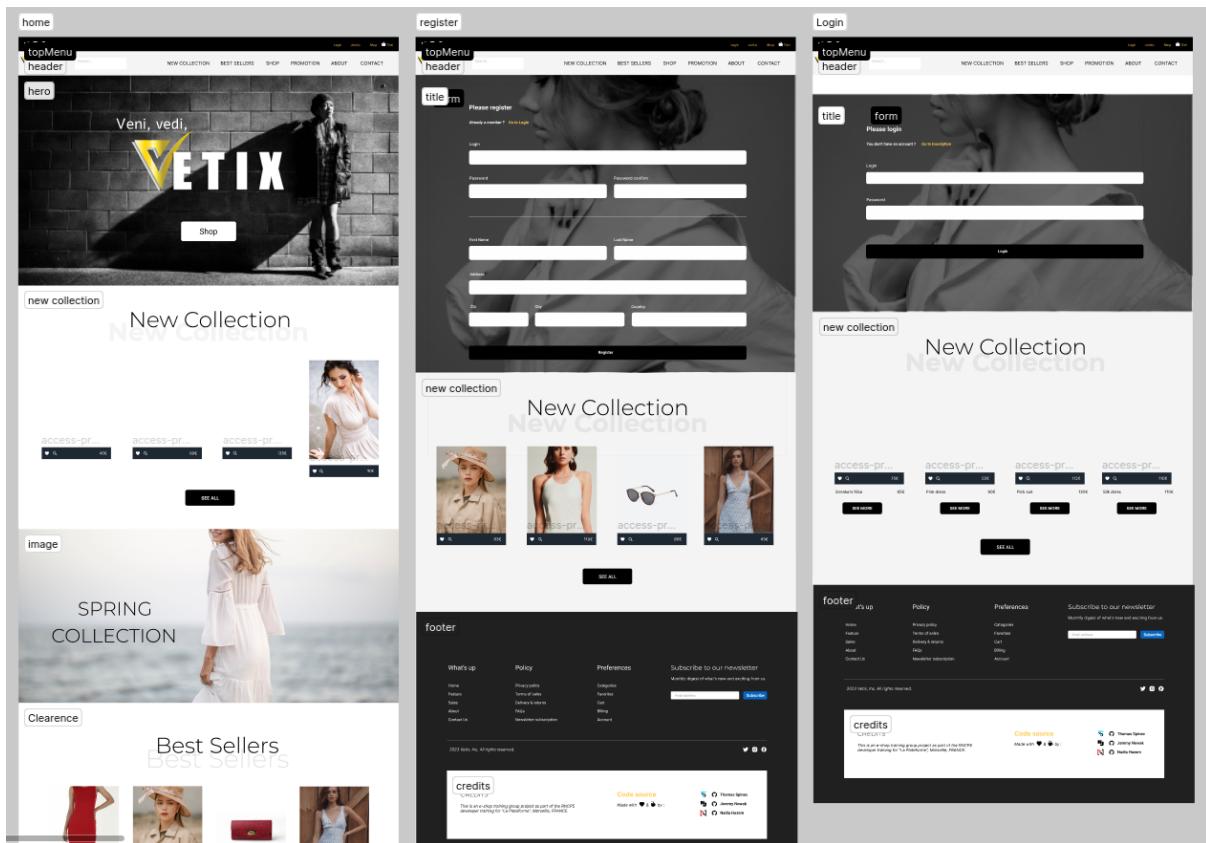
-Maquette basse fidélité:

La maquette basse fidélité est très utile afin d'avoir rapidement un support visuel du projet. Son manque de détail permet à la fois d'être rapidement créé mais aussi modifier sans avoir passé trop de temps dessus.



-Maquette haute fidélité:

Les maquettes haute fidélité a l'inverse des maquettes basses fidélités permettent de montrer un projet fini. Cela permet de valider ou non le design final ainsi que de remarquer les incohérences visuelles ainsi que les erreurs de conception et tout cela avant de passer à la conception.



5.Code

1.Authentification

Voici un screenshot de ma fonction JavaScript. Cette fonction vérifiera les champs et s'ils ne sont pas remplis un message d'erreur apparaîtra et la couleur du champ changera.

Une fois les champs remplis nous utiliserons `FormData()` qui permet de construire facilement un ensemble de paires clé/valeur représentant les champs du formulaire et leurs valeurs. Après quoi nous utilisons `l'API fetch()` afin d'envoyer les données en `POST`.

Ces données seront récupérées grâce à notre page de traitement que vous retrouverez sur l'image suivante.

```
function verifLogin() {
  let loginValue = login_insc.value;
  // verif if login is empty
  if (loginValue === "") {
    login_insc.nextElementSibling.innerHTML = "Please enter a login";
    // change border color and background
    login_insc.style.borderColor = "red";
    login_insc.style.backgroundColor = "#ff000033";

    validation = false;
  } else {
    login_insc.nextElementSibling.innerHTML = "";
    // change border color to default
    login_insc.style.borderColor = "initial";
    login_insc.style.backgroundColor = "#fafafa";
    // verif if login is available
    let dataLogin = new FormData();
    dataLogin.append("verifLogin", loginValue);
    // console.log(dataLogin);
    fetch("inc/php/auth.php", {
      method: "POST",
      body: dataLogin,
    })
      .then((response) => response.text())
      .then((response) => {
        response = response.trim();
        // console.log(response);
        if (response === "indispo") {
          login_insc.nextElementSibling.innerHTML = "Login unavailable";
          // change border color and background
          login_insc.style.borderColor = "red";
        }
      })
  }
}
```

Sur cette image vous trouverez la fonction `isUserExist()` qui prend en paramètre `$login`. Celle-ci sera utilisée lors du remplissage du champ avec comme `addEventListener()` en “blur” pour vérifier si le login existe déjà .

```
public function isUserExist($login)
{
    $request = "SELECT * FROM $this->tablename WHERE login = :login";
    $select = $this->bdd->prepare($request);
    $select->execute([
        ':login' => $login
    ]);
    $result = $select->fetch(PDO::FETCH_ASSOC);
    if ($result) {
        echo "indispo";
    } else {
        echo "dispo";
    }
    $this->bdd = null;
}
```

Ici vous trouverez notre page de traitement qui détectera si un fetch a été effectué. Dans cet exemple, nous recherchons si un fetch en méthode POST avec un nom tel que “verifLogin”, “insc” ou “conn”.

Lorsqu'un de ces fetch est détecté, on récupérera les données contenues dans l'objet de formData() pour les mettre dans des variables. Une fois récupéré nous utiliserons une méthode de la classe User que nous avons déjà instancié au dessus grâce à “new” User()

```
require_once "../class/User.php";

$user = new User();

// is user exist
if (isset($_POST['verifLogin'])) {
    $login = $_POST['verifLogin'];
    $user->isUserExist($login);
}

// register
if (isset($_POST['insc'])) {
    $login = $_POST['login'];
    $password = $_POST['password'];
    $firstname = $_POST['firstname'];
    $lastname = $_POST['lastname'];
    $mail = $_POST['email'];
    $address = $_POST['address'];
    $city = $_POST['city'];
    $zip = $_POST['zip'];
    $country = $_POST['country'];
    $user->register($login, $password, $firstname, $lastname, $mail, $address, $city, $zip, $country);
}

// connection
if (isset($_POST['conn'])) {
    $login = $_POST['login'];
    $password = $_POST['password'];
    $user->connect($login, $password);
}
```

Et pour terminer je vous présente la fonction `register()` qui prend en paramètre toutes les données dont nous aurons besoin pour identifier un utilisateur de notre site.

Vous remarquerez que j'ai utilisé “`htmlspecialchars()`” afin de se protéger des failles XSS (Cross-Site Scripting) ce qui échappe les caractères en entité HTML. Par exemple, le caractère "<" est remplacé par "<".

J'utilise aussi “`password_hash()`” afin de crypter le mot de passe et je le fais avec la méthode “`PASSWORD_DEFAULT`”. Celui-ci utilisera l'algorithme “`bcrypt`” (il est utilisé depuis PHP 5.5.0) . Je l'utilise donc car il est conçu pour changer dans le temps, au fur et à mesure que des algorithmes plus récents et plus forts sont ajoutés à PHP.

Après quoi je passerai par 3 étapes. La construction de la requête, cette étape consiste à créer la requête SQL qui sera exécutée pour insérer les données dans la table.

La préparation de la requête, après avoir construit la requête SQL, vous préparez la requête en utilisant la méthode `prepare()` en passant la requête en argument. La préparation de requête est une sécurité contre les injections SQL et si une erreur se produit lors de l'exécution de la requête préparée, la base de données n'a pas besoin de recevoir les informations à chaque fois.

Et pour finir l'exécution, une fois la requête préparée, vous exécutez la requête en utilisant la méthode `execute()`. J'associe les valeurs des paramètres à l'aide d'un tableau associatif, où les clés correspondent aux paramètres définis dans la requête préparée. Cela permet d'insérer les valeurs réelles dans la requête à exécuter.

```
public function register($login, $password, $email, $firstname, $lastname, $address,
$city, $zip, $country) thomas-spinec, 2 months ago * correction de l'affichage en d
{
    // special characters
    $login = htmlspecialchars($login);
    $password = htmlspecialchars($password);
    $email = htmlspecialchars($email);
    $firstname = htmlspecialchars($firstname);
    $lastname = htmlspecialchars($lastname);
    $address = htmlspecialchars($address);
    $city = htmlspecialchars($city);
    $country = htmlspecialchars($country);
    $zip = htmlspecialchars($zip);

    // hash password
    $password = password_hash($password, PASSWORD_DEFAULT);

    $register = "INSERT INTO $this->tablename (login, password, email, firstname,
lastname, address, zip, city, country) VALUES
(:login, :password, :email, :firstname, :lastname, :address, :zip, :city, :country)
";
    // préparation de la requête
    $insert = $this->bdd->prepare($register);
    // exécution de la requête avec liaison des paramètres
    $insert->execute([
        ':login' => $login,
        ':password' => $password,
        ':email' => $email,
        ':firstname' => $firstname,
        ':lastname' => $lastname,
        ':address' => $address,
        ':zip' => $zip.
```

Si tout s'est bien passé nous afficherons un message de validation

```
echo "Successfully registered!";
```

2. Favoris

Dans notre projet nous avons mis en place une liste de favoris. Il faudra être connecté pour avoir accès à cette feature.

En bas à gauche nous ne sommes pas connectés mais sur l'image de droite nous le sommes.



Ici vous trouverez le avant après du click sur le cœur représentant le favoris .



83€



83€

Afin de déclencher la fonction PHP nous passerons par un fetch. En effet a chaque appui sur le bouton nous récupérons le `data-id` que nous mettons dans une variable. Celle-ci transitera vers le PHP grâce à la création d'un objet `FormData()` que nous passerons dans le corps du fetch en méthode `POST` adresser à la page `addToWishList.php`.

```
love.addEventListener('click', event => {
  if (event.target.classList.contains('heart')) {
    event.preventDefault();
    const productId = event.target.getAttribute('data-id');

    let data = new FormData();
    data.append('productId', productId);

    fetch('inc/php/addToWishlist.php', {
      method: 'POST',
      body: data,
    })
    .then((response) => response.text())
    .then((response) => {
      response = response.trim();
      // If the request was successful, toggle the heart icon state
      if (response === "ok") {
        if(event.target.classList.contains('clicked')){
          event.target.classList.remove('clicked');
        }
        else{
          event.target.classList.add('clicked');
        }
      } else {
        console.log(response);
      }
    })
    .catch(error => {
      console.error(error);
    });
});
```

Si l'utilisateur est bel est bien connecté, on récupère l'id de l'utilisateur dans la super global `$_SESSION` à l'index "user" et et sa valeur "id". Après cela nous lançons donc la fonction `addToWishList()` avec les informations adéquates en paramètre.

```
<?php
session_start();
require_once '../class/User.php';
require_once '../class/Product.php';
require_once '../class/Wishlist.php';

$user = new User();
$product = new Product();
$wishlist = new Wishlist();

if ($user->isLogged()) {
    $user_id = $_SESSION['user']['id'];
    $product_id = $_POST['productId'];

    $wishlist->addToWishlist($user_id, $product_id);
} else {
    header('Location: login.php');
    exit();
}
```

Voici la fonction qui va nous permettre de mettre à jour l'état du favoris. Cette fonction vérifiera si l'article est déjà en favori. Si il l'est on utilisera la fonction `removeFromWishlist()`, sinon on ajoutera l'article.

```
public function addToWishlist($user_id, $product_id) {
    $user_id = htmlspecialchars($user_id);
    $product_id = htmlspecialchars($product_id);

    // Check whether the product is already in the user's wishlist
    if ($this->isFavorite($user_id, $product_id)) {
        $this->removeFromWishlist($user_id, $product_id);
    }
    else {
        // Insert the product into the wishlist table
        $request = "INSERT INTO $this->tablename (id_user, id_product, date) VALUES (:id_user,
:id_product, NOW())";
        $select = $this->bdd->prepare($request);
        $select->execute([':id_user' => $user_id, ':id_product' => $product_id]);

        if ($select){
            echo "ok";
        } else {
            echo "error";
        }
    }
}
```

La fonction `isFavorite()` comptera le nombre de ligne correspondantes à la requête.

```
public function isFavorite($user_id, $product_id) {
    $user_id = htmlspecialchars($user_id);
    $product_id = htmlspecialchars($product_id);

    // Check whether the product is already in the user's wishlist
    $request = "SELECT COUNT(*) FROM $this->tablename WHERE id_user = :id_user AND id_product = :id_product";
    $select = $this->bdd->prepare($request);
    $select->execute([':id_user' => $user_id, ':id_product' => $product_id]);

    $result = $select->fetchColumn();
    return $result > 0;
}
```

Après que cette requête ait été préparée et exécutée, on utilisera `fetchColumn()` afin de récupérer le résultat de la requête. On utilisera le return de la variable contenant le résultat et si, celui-ci est supérieur à 0 la fonction donc enverra un booléen true ou false dans le cas contraire.

Maintenant je vous montre la fonction qui permet de supprimer . Rien de bien spécial ici c'est simplement un `DELETE` qui va supprimer dans la table wishlist la ligne qui contiendra l'id de l'utilisateur ainsi que l'id du produit.

```
public function removeFromWishlist($user_id, $product_id) {
    $user_id = htmlspecialchars($user_id);
    $product_id = htmlspecialchars($product_id);

    // Delete the product from the wishlist table
    $request = "DELETE FROM $this->tablename WHERE id_user = :id_user AND id_product = :id_product";
    $select = $this->bdd->prepare($request);
    $select->execute([':id_user' => $user_id, ':id_product' => $product_id]);

    if ($select){
        echo "ok";
    } else {
        echo "error";
    }
}
```

3. Commentaire

Comme pour le favoris, l'utilisateur doit être connecté afin de poster un commentaire. L'affichage de commentaire se fera grâce à une requête qui ira chercher tous les commentaire via l'ID du produit comme indiqué ci-dessous

```
public function getComments($id)
{
    $id = (int)$id;
    $request = "SELECT comment.* , DATE_FORMAT(comment.date, '- %d %m %Y %H:%i -') as date, user.
    login as author
    FROM comment
    INNER JOIN user ON comment.id_user = user.id_user
    WHERE comment.id_product = :id
    ORDER BY date DESC
    ";
    // request
    $select = $this->bdd->prepare($request);
    // exec with params
    $select->execute([
        'id' => $id,
    ]);

    // get results
    $comments = $select->fetchAll(PDO::FETCH_ASSOC);
    // if $result produce an error, we return null
    if (!$comments) {
        return null;
    } else {
        // else we return the result
        return $comments;
    }
}
```

Le résultat de la requête sera traitée avec **FETCH_ASSOC** ce qui nous permet de récupérer la réponse sous forme d'un tableau associatif. Si il n'y a pas de commentaire on utilisera un **return null** Sinon nous utiliserons **return** afin retourner la valeur contenu dans **\$comments**.

Après avoir montré la fonction d'affichage des commentaires lié à l'id d'un article, je vais vous montrer la fonction qui permettra d'ajouter des commentaires ci-dessous.

```

public function addComment($subject, $comment, $id_product, $id_user)
{
    // html special chars to avoid injections
    $subject = htmlspecialchars($subject);
    $comment = htmlspecialchars($comment);
    $id_product = htmlspecialchars($id_product);
    $id_user = htmlspecialchars($id_user);

    // request
    $request = "INSERT INTO $this->tablename (subject, comment, date, id_product, id_user)
VALUES (:subject, :comment, NOW(), :id_product, :id_user)";

    $insert = $this->bdd->prepare($request);

    // exec with params
    $insert->execute([
        'subject' => $subject,
        'comment' => $comment,
        'id_product' => $id_product,
        'id_user' => $id_user,
    ]);

    // echo "ok" if the request went well
    if ($insert) {
        echo "ok";
    } else {
        echo "erreur";
    }
    $this->bdd = null;
}

```

La fonction `addComment()` prend en paramètre toutes les informations indispensables à l'identification du commentaire ainsi qu'à son contenu.

Nous récupérons tous les champs et les passons dans la fonction `htmlspecialchars()` afin d'échapper tous caractères qui permettrait d'exécuter un script malveillant (faille XSS).

Vous pouvez voir que dans la requête qui est un `INSERT` nous utilisons la fonction native SQL `NOW()` afin d'insérer la date et l'heure du commentaire.

4.Panel admin

Pour le panel admin je vais vous présenter comment abaisser ou éléver les privilèges.

```
<form method="post" id="formRole">
  <select name="role" id="catchRole" data-id=<?= $value
  ['id_user'] ?> class="role">
    <option data-id="user" value="user">user</option>
    <option data-id="admin" value="admin">admin</option>
  </select>
  <input type="submit" class="changeDroit btn btn-sm btn-dark"
  id="changeDroit" data-id=<?= $value['id_user'] ?>
  value="Change">
</form>
```

Dans ce formulaire en méthode **POST** nous y avons mis un select avec deux **option**. Celles-ci seront récupérées grâce au **data-id** de ces mêmes **option** ainsi que l'id de l'utilisateur.

Notre fonction JavaScript ici identifie le formulaire et instancie un nouvel objet de **FormData()** afin d'y stocker les données.

```
function changeRole(id) {
  const formRole = document.querySelector("#formRole");

  let data = new FormData(formRole);
  data.append("id", id);
  data.append("changeRole", "ok");
  fetch("inc/php/adminGestion.php", {
    method: "POST",
    body: data,
  })
  .then((response) => response.text())
  .then((response) => {
    response = response.trim();
    if (response === "ok") {
      displayUsers();
    } else {
      display.nextElementSibling.innerHTML = "Error, role didn't change";
      setTimeout(() => {
        display.nextElementSibling.innerHTML = "";
      }, 2000);
    }
  });
}
```

Notre page de traitement va détecter ce fetch en méthode `POST` du nom de 'changeRole' et exécuter la méthode `changeRole()` de la classe `User.php` qui aura été instanciée au préalable avec les informations en paramètre.

```
if (isset($_POST['changeRole'])) {
    $newRole = $_POST['role'];
    $id = $_POST['id'];
    $user->changeRole($newRole, $id);
}
```

Et pour terminer voici la méthode `changeRole()` qui passe les données extérieur dans la fonction `htmlspecialchars()` dans un souci de sécurité . Après quoi nous utiliserons un simple `UPDATE` afin de changer le rôle de l'utilisateur. Sans oublier la préparation de cette dernière.

```
public function changeRole($newRole, $id)
{
    $newRole = htmlspecialchars($newRole);
    $id = htmlspecialchars($id);
    $request = "UPDATE $this->tablename SET `role` = :role WHERE id_user=:id";

    $updateRole = $this->bdd->prepare($request);

    $updateRole->execute([
        ":role" => $newRole,
        ":id" => $id,
    ]);

    if ($updateRole) {
        echo "ok";
    } else {
        echo "error";
    }
}
```

6. Conclusion

Nous n'avons pas pu implémenter toutes les fonctionnalités que nous voulions, telles que la pagination, la vérification du stock à l'ajout dans un panier afin d'éviter que le produit soit en fait en rupture.

Ce projet réalisé à l'école La Plateforme, nous a permis (en plus d'autres projets) de valider toutes les compétences du référentiel.

Malgré cela, notre site possède plusieurs fonctionnalités, telles que:

- une page de boutique permettant de trier les produits par catégorie.
- un panier
- une page profil permettant également d'avoir accès à sa wishlist
- un panel administrateur pour gérer le contenu
- une page accueil mettant en avant quelques produits

En conclusion, ce projet a été le fruit d'un mois de travail en équipe. J'ai été épaulé par Nadia Hazem ainsi que mon très cher camarade Thomas Spinec.