

GStock8

RAPPORT DE PROJET

CORENTIN SANJUAN
BTS SIO SLAM - 2024

SOMMAIRE

01 - Contexte

02 - Expression des besoins

03 - Les objectifs

04 - Analyse fonctionnelle

05 - Deploiement

06 - Manuel utilisateur

07 - Conclusion

Contexte

Description du laboratoire GSB

Le secteur d'activité :

L'industrie pharmaceutique est un secteur très lucratif dans lequel le mouvement de fusion acquisition est très fort. Les regroupements de laboratoires ces dernières années ont donné naissance à des entités gigantesques au sein desquelles le travail est longtemps resté organisé selon les anciennes structures. Des déboires divers récents autour de médicaments ou molécules ayant entraîné des complications médicales ont fait s'élever des voix contre une partie de l'activité des laboratoires : la visite médicale, réputée être le lieu d'arrangements entre l'industrie et les praticiens, et tout du moins un terrain d'influence opaque.

L'entreprise :

Le laboratoire Galaxy Swiss Bourdin (GSB) est issu de la fusion entre le géant américain Galaxy (spécialisé dans le secteur des maladies virales dont le SIDA et les hépatites) et le conglomérat européen Swiss Bourdin (travaillant sur des médicaments plus conventionnels), lui même déjà union de trois petits laboratoires . En 2009, les deux géants pharmaceutiques ont uni leurs forces pour créer un leader de ce secteur industriel. L'entité Galaxy Swiss Bourdin Europe a établi son siège administratif à Paris. Le siège social de la multinationale est situé à Philadelphie, Pennsylvanie, aux Etats-Unis. La France a été choisie comme témoin pour l'amélioration du suivi de l'activité de visite.

Réorganisation :

Une conséquence de cette fusion, est la recherche d'une optimisation de l'activité du groupe ainsi constitué en réalisant des économies d'échelle dans la production et la distribution des médicaments (en passant par une nécessaire restructuration et vague de licenciement), tout en prenant le meilleur des deux laboratoires sur les produits concurrents. L'entreprise compte 480 visiteurs médicaux en France métropolitaine (Corse comprise), et 60 dans les départements et territoires d'outre-mer. Les territoires sont répartis en 6 secteurs géographiques (Paris-Centre, Sud, Nord, Ouest, Est, DTOM Caraïbes-Amériques, DTOM Asie-Afrique).

Expression des besoins

Gestion des stocks :

- Fournir une base de données permettant de stocker les informations relatives aux médicaments, matériels et autres ressources.
- Permettre l'ajout, la modification et la consultation des stocks de manière conviviale et intuitive.
- Garantir la quantité toujours positive d'un stock en mettant en place des mécanismes de contrôle appropriés.

Gestion des commandes :

- Fournir une base de données permettant de stocker les informations relatives aux commandes.
- Permettre l'ajout et la consultation des commandes de manière conviviale et intuitive.
- Assurer le contrôle des entrées et sorties de stocks lors du traitement des commandes.

Gestion des utilisateurs :

- Mettre en place une base de données d'utilisateurs pour gérer les accès au système.
- Permettre l'ajout, la modification et la consultation des utilisateurs avec des rôles appropriés (administrateur, utilisateur).

Sécurité et confidentialité :

- Mettre en place des mécanismes d'authentification des utilisateurs pour garantir l'accès sécurisé au système.
- Chiffrer les données sensibles, notamment les mots de passe des utilisateurs.
- Assurer la confidentialité des données en mettant en place des mesures de protection appropriées.

Les objectifs

Ce projet vise à permettre à un laboratoire pharmaceutique GSB de gérer ses stocks de médicaments, matériels ou autres. Ce logiciel devra comporter une base de données de stocks, de commandes afin pouvoir les entrées ou sortie du stock, il y aura également une base d'utilisateur à gérer. GStockB aura pour objectif d'assurer la quantité toujours positive d'un stock et doit permettre le contrôle des entrée et sortie avec la validation d'un "administrateur".

Pour réaliser ces fonctionnalités, le logiciel devra fournir une interface conviviale et intuitive permettant d'ajouter, modifier et consulter les commandes et les stocks. Il sera important d'assurer la sécurité et la confidentialité des données, en mettant en place des mesures de protection appropriées, telles que l'authentification des utilisateurs et le chiffrement des données sensibles tels que les mots de passe.

Analyse fonctionnelle

Cette application présente plusieurs fonctionnalités

- Connexion à partir d'un compte utilisateur unique.
- Tableau de bord pour visualiser rapidement les informations importantes.
- Création de stocks.
- Création de commandes contenant plusieurs stocks.
- Activation/désactivation d'un utilisateur.
- Interface interactive avec un système de "pop-up".
- Création de fichiers de logs.
- Hachage des mots de passe des utilisateurs en base de données.
- Architecture logicielle MVC.

Langages et technologies utilisés :

- PHP
- SQL
- CSS
- XAMPP (X, Apache (serveur Web), MariaDB (système de gestion de bases de données relationnelles), PHP, PERL)
- Git, GitHub
- Visual Studio Code

Liens des répertoires GitHub

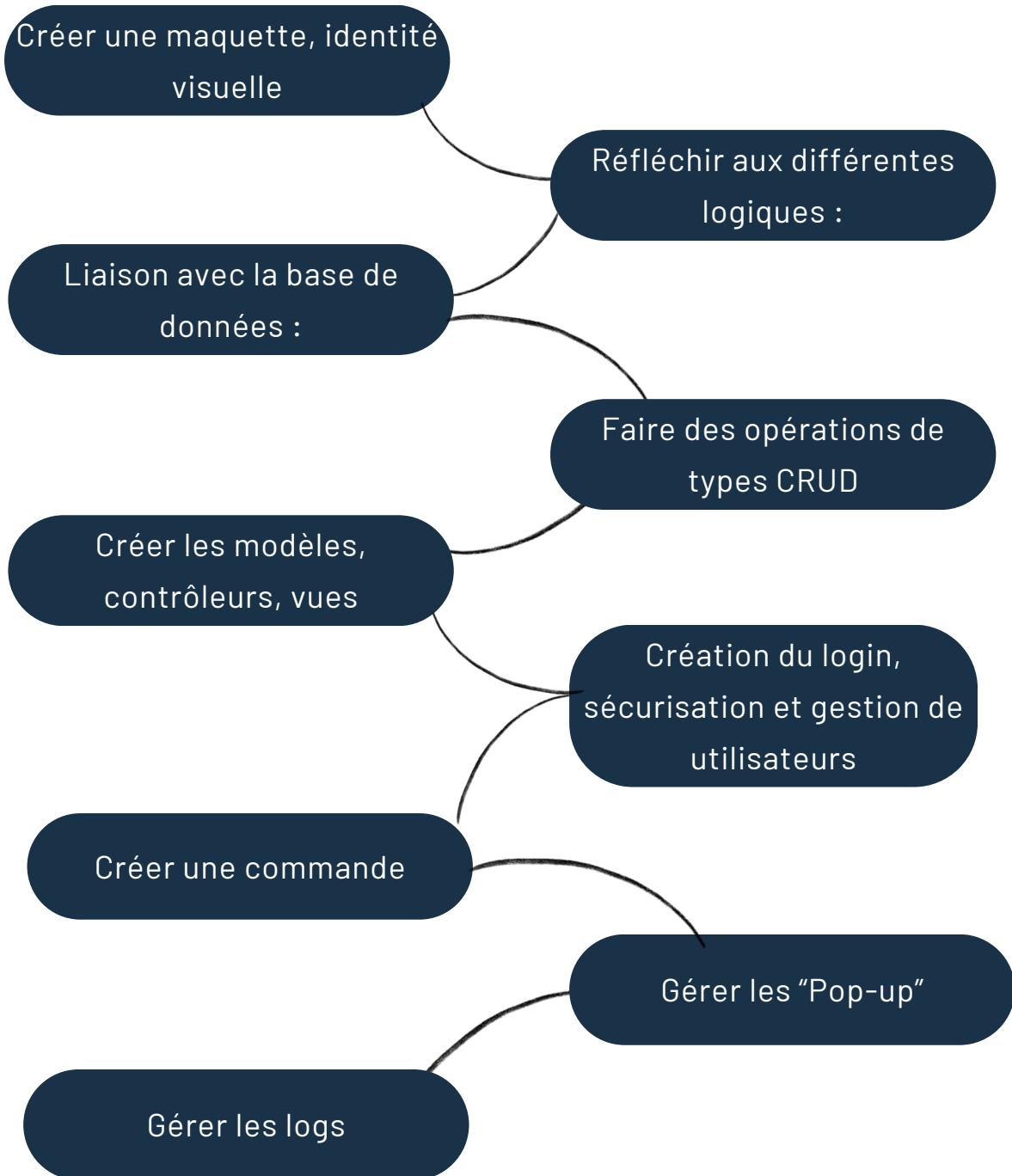
- Projet : <https://github.com/Crtnsj/GStockB>
- Base de données : <https://github.com/Crtnsj/BDD-GStockB>

Présentation des technologies utilisées :

- PHP (PHP: Hypertext Preprocessor) est un langage de programmation côté serveur populaire et largement utilisé pour le développement web. Il est principalement utilisé pour générer des pages web dynamiques et interagir avec des bases de données. PHP offre une grande flexibilité et une large gamme de fonctionnalités pour créer des applications web puissantes.
- SQL (Structured Query Language) est un langage de programmation spécialement conçu pour gérer les bases de données relationnelles. Il permet de créer, modifier et interroger des bases de données pour stocker, récupérer et manipuler des informations. SQL est essentiel dans le développement de systèmes de gestion de bases de données et joue un rôle clé dans la manipulation et la gestion des données.

- CSS : CSS (Cascading Style Sheets) est un langage de feuille de style utilisé pour décrire la présentation et l'apparence des documents HTML. Il permet de contrôler les aspects visuels d'une page web, tels que les couleurs, les polices, les marges, les mises en page, etc. CSS offre une séparation claire entre la structure HTML et le style, permettant ainsi une conception web plus modulaire et maintenable.
- XAMPP (X, Apache, MariaDB, PHP, PERL) : XAMPP est un ensemble de logiciels open source qui fournit un environnement de développement web complet. En utilisant XAMPP, les développeurs peuvent configurer un environnement de développement local pour tester et exécuter leurs applications web avant de les déployer sur un serveur en production.
- Git : Git est un système de contrôle de version distribué qui permet aux développeurs de gérer et de suivre les modifications de leur code source. Il permet de créer des branches, de fusionner des modifications et de revenir à des versions antérieures du code. Git facilite la collaboration et la gestion des projets de développement de logiciels.
- GitHub : GitHub est une plateforme de développement logiciel basée sur Git, où les développeurs peuvent héberger et partager leurs projets Git de manière centralisée. Il fournit également des fonctionnalités de suivi des problèmes, de gestion des demandes de fusion et de collaboration entre développeurs. GitHub est largement utilisé dans la communauté du développement logiciel open source et facilite le partage et la contribution aux projets.
- 6. Visual Studio Code : Visual Studio Code (VS Code) est un éditeur de code source léger et très populaire développé par Microsoft. Il prend en charge de nombreux langages de programmation et offre une variété de fonctionnalités telles que la coloration syntaxique, l'autocomplétion, le débogage, la gestion des extensions et l'intégration avec des outils de contrôle de version. VS Code est apprécié pour sa facilité d'utilisation, sa personnalisation et sa capacité à s'adapter à différentes technologies de développement.

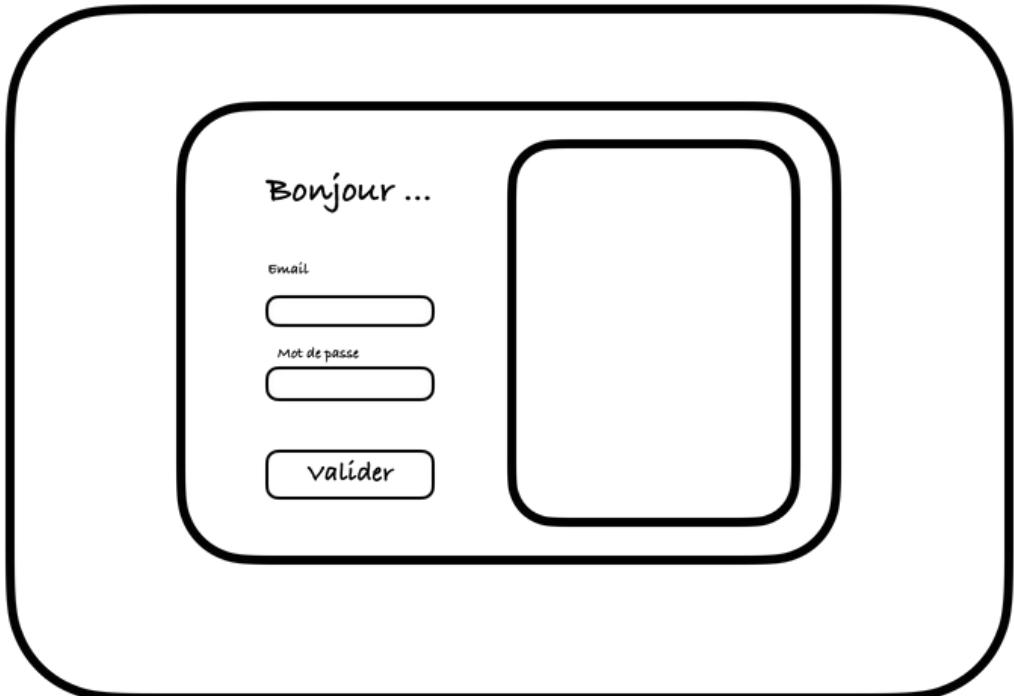
Les grandes étapes à réaliser :



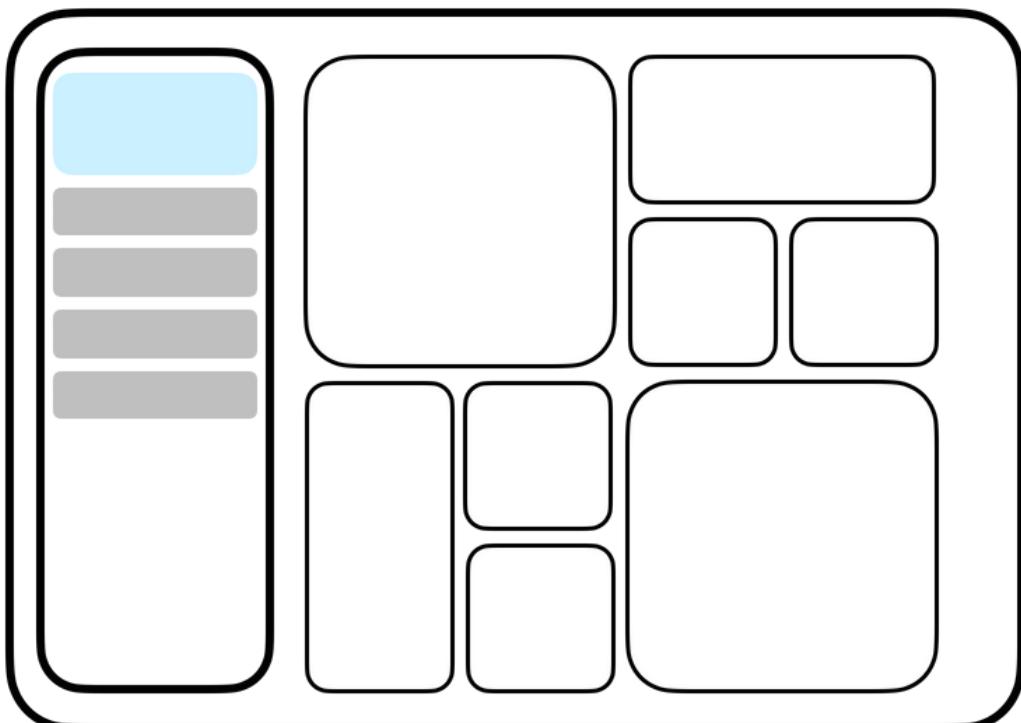
Créer une maquette, identité visuelle

Afin de réaliser ma maquette et mon identité visuelle, j'ai utilisé l'application FreeForm d'Apple. Cette application est simple d'utilisation et synchronisée avec tous mes appareils.

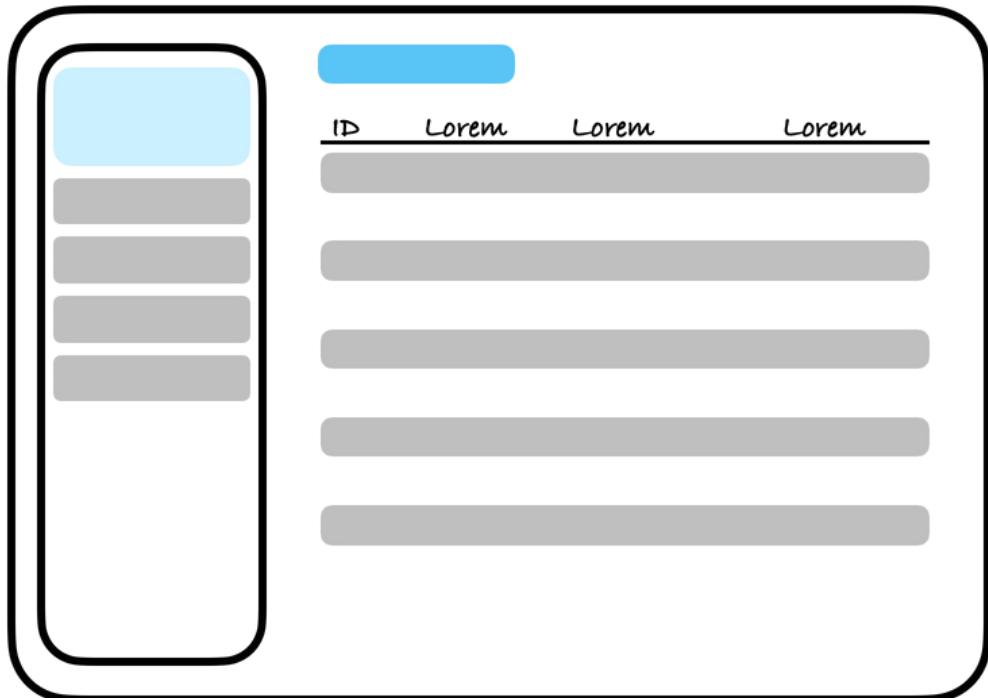
Pour commencer, j'ai conçu mes différentes pages. Voici, par exemple, ma maquette de la page de connexion :



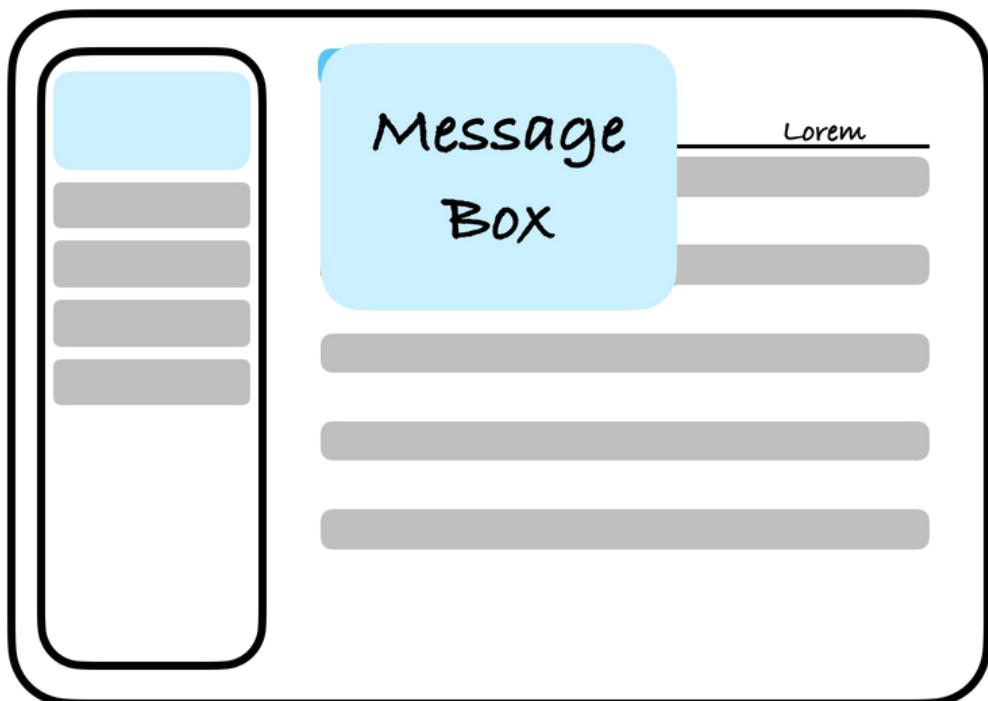
La page d'accueil avec le tableau de bord.



Le modèle qui sera appliqué aux autres pages :



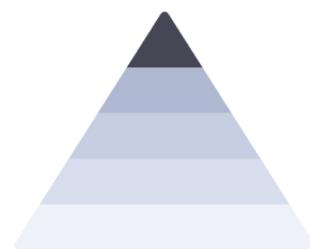
Et enfin, le modèle de "pop-up" pour les messages d'erreur, de succès ou pour les formulaires d'actions.



J'ai par la suite, créer les différents logos :



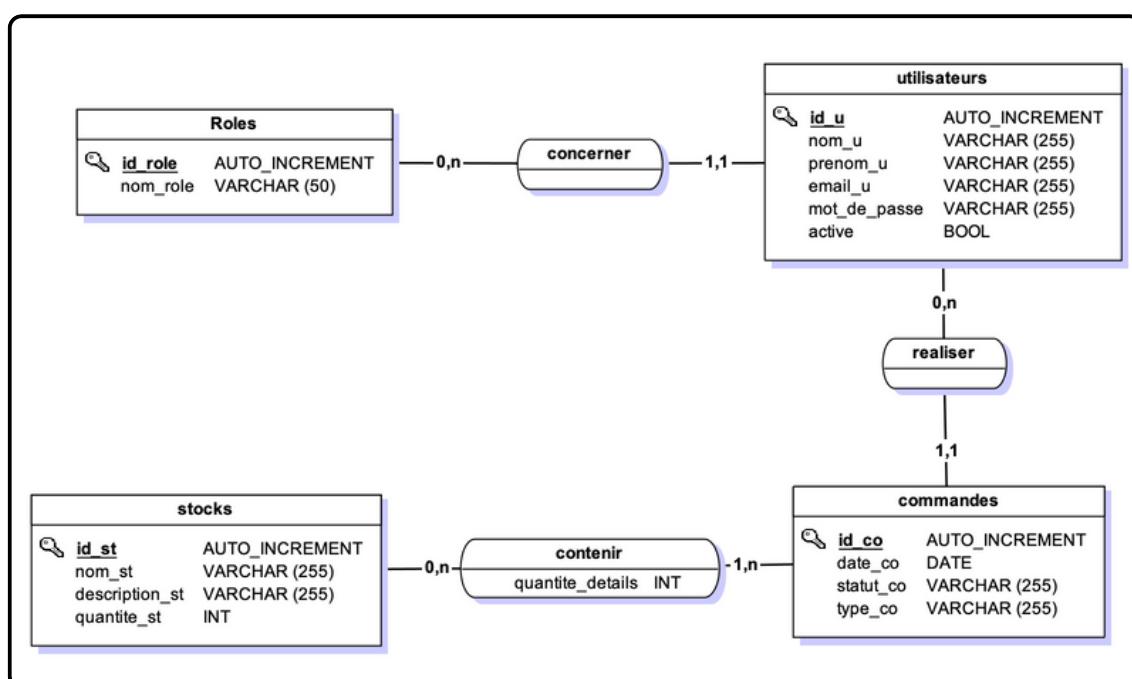
Rectangulaire



Carré

Réfléchir aux différentes logiques :

La conception de cette application a nécessité la création de plusieurs schémas afin de mieux comprendre et préparer la logique métier qui sera appliquée. Dans un premier temps, j'ai créé un Modèle Conceptuel de Données (MCD) pour représenter la base de données. Voici le MCD que j'ai utilisé :



Mon application comporte plusieurs niveaux de droits, ce qui permet de séparer les personnes habilitées à réaliser certaines actions de celles qui ne le sont pas. Voici un tableau résumant les différents niveaux de droits :

Super-Administrateur	Administrateur	Utilisateur
Gestion des utilisateurs	Validation des commandes Gestion des stocks	Création de commandes

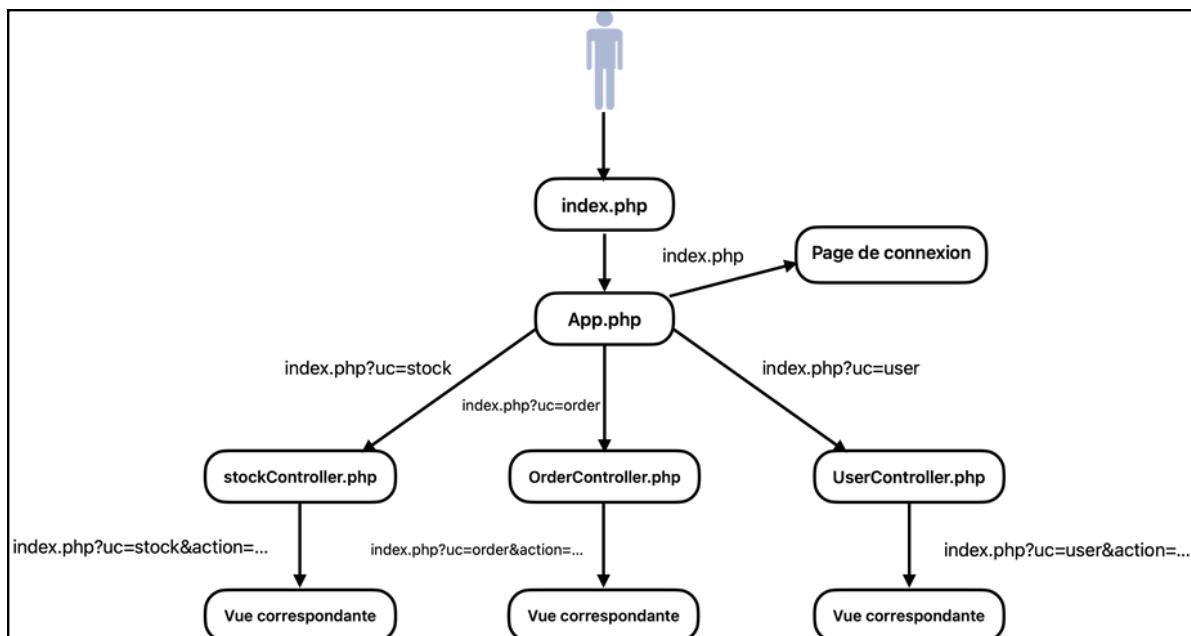
Dans le tableau précédent, il faut noter que le super administrateur possède également les droits des administrateurs et des utilisateurs, tandis que les administrateurs ont également les droits des utilisateurs.

Dans un second temps, j'ai préparé l'architecture applicative. Pour cela, j'ai utilisé le modèle MVC (Modèle-Vue-Contrôleur). Ce modèle d'architecture applicative favorise la séparation des préoccupations, ce qui facilite la maintenance et l'évolutivité du code.

MVC est composé de trois parties :

- Modèle (Model) : Le modèle représente les données et la logique métier de l'application. Il contient les structures de données, les méthodes et les algorithmes qui manipulent et gèrent les données. Le modèle est généralement indépendant de l'interface utilisateur et des autres composants de l'application.
- Vue (View) : La vue est responsable de l'affichage des données au sein de l'interface utilisateur. Elle représente la présentation visuelle des informations contenues dans le modèle. La vue reçoit les données du modèle et les organise de manière appropriée pour les présenter à l'utilisateur. Elle peut également être responsable de recueillir les entrées de l'utilisateur, telles que les clics de souris ou les saisies clavier.
- Contrôleur (Controller) : Le contrôleur agit comme un intermédiaire entre le modèle et la vue. Il reçoit les entrées de l'utilisateur à partir de la vue, traite ces entrées et met à jour le modèle en conséquence. Le contrôleur est responsable de la coordination des interactions entre la vue et le modèle, et il peut également déclencher des actions supplémentaires en fonction des événements survenant dans l'application.

Voici l'architecture de ce modèle dans mon application:



Voici également l'architecture de mes fichiers :

```
/GStockB
  /public
    index.php
    /pics
    /styles
      /sass
      css-compiled.css
  /src
    /core
      /App.php
      /Database.php
    /controllers
      userController.php
      orderController.php
      stockController.php
    /models
      User.php
      Order.php
      Stock.php
    /views
      v_head.php
      v_home.php
      v_foot.php
      /Order
        v_order.php
        ...
      /User
        v_user.php
        ...
      /Stock
        v_stock.php
        ...
```

Liaison avec la base de données :

Dans mon application, la connexion avec la base de données se fait à l'aide de PHP Data Object (PDO). PDO est une extension de PHP qui fournit une interface cohérente pour accéder aux bases de données. L'avantage de son utilisation est que nous pouvons travailler avec plusieurs systèmes de base de données différents sans changer la logique PHP. PDO permet également la création de requêtes préparées afin d'éviter les injections SQL. Voici le code correspondant à ma classe Database :

```
class Database
{
    private $host = 'localhost'; // Database server address
    private $dbname = 'GStockB'; // Database name
    private $username = 'root'; // Username for database connection
    private $password = ''; // Password for database connection
    private $dbHandler; // PDO connection handler
    private $statement; // PDO prepared statement
    private $error; // Error message in case of connection failure or query errors

    /**
     * Constructor method.
     * Establishes a connection to the database using PDO.
     */
    public function __construct()
    {
        //créer la chaîne de connexion
        $dsn = 'mysql:host=' . $this->host . ';dbname=' . $this->dbname;
        $options = array(
            PDO::ATTR_PERSISTENT => true,
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
        );

        try {
            //créer un nouvel objet PDO avec les informations de connexions et les paramètres
            //nécessaires
            $this->dbHandler = new PDO($dsn, $this->username, $this->password, $options);
        } catch (PDOException $e) {
            $this->error = $e->getMessage();
            echo $this->error;
        }
    }
}
```

Faire des opérations de types CRUD

Faire des opérations de type CRUD signifie être capable de créer, lire, modifier et supprimer un élément dans la base de données. Dans mon application, les opérations de type CRUD seront utiles pour, par exemple :

- Créer et voir les commandes
- Modifier les informations d'un utilisateur.
- Supprimer un stock.
- Récupérer les détails d'une commande.

L'utilisation de PDO pour établir la connexion entre l'application et la base de données facilite la préparation et l'exécution des requêtes, ainsi que la récupération des résultats de la base de données. Ces fonctionnalités sont mises en œuvre à l'aide des méthodes définies dans la classe Database.

Voici les méthodes qui permettent cela :

```
/**  
 * Executes a database query.  
 * Prepares the statement and returns it.  
 *  
 * @param string $sql The SQL query to execute.  
 * @return PDOStatement The prepared statement.  
 */  
public function query($sql)  
{  
    return $this->statement = $this->dbHandler->prepare($sql);  
}
```

Prépare la requête

Définit les paramètres
de requête

```
/**  
 * Binds a value to a parameter in the prepared statement.  
 *  
 * @param string $parameter The parameter placeholder in the SQL query.  
 * @param mixed $value The value to bind.  
 * @param int|null $type The data type of the parameter (default: PDO::PARAM_STR).  
 */  
public function bind($parameter, $value, $type = null)  
{  
    switch (is_null($type)) {  
        case is_int($value):  
            $type = PDO::PARAM_INT;  
            break;  
        case is_bool($value):  
            $type = PDO::PARAM_BOOL;  
            break;  
        case is_null($value):  
            $type = PDO::PARAM_NULL;  
            break;  
        default:  
            $type = PDO::PARAM_STR;  
    }  
    $this->statement->bindValue($parameter, $value, $type);  
}
```

Exécute la requête

```
/**  
 * Executes the prepared statement.  
 *  
 * @return bool True on success, false on failure.  
 */  
public function execute()  
{  
    return $this->statement->execute();  
}
```

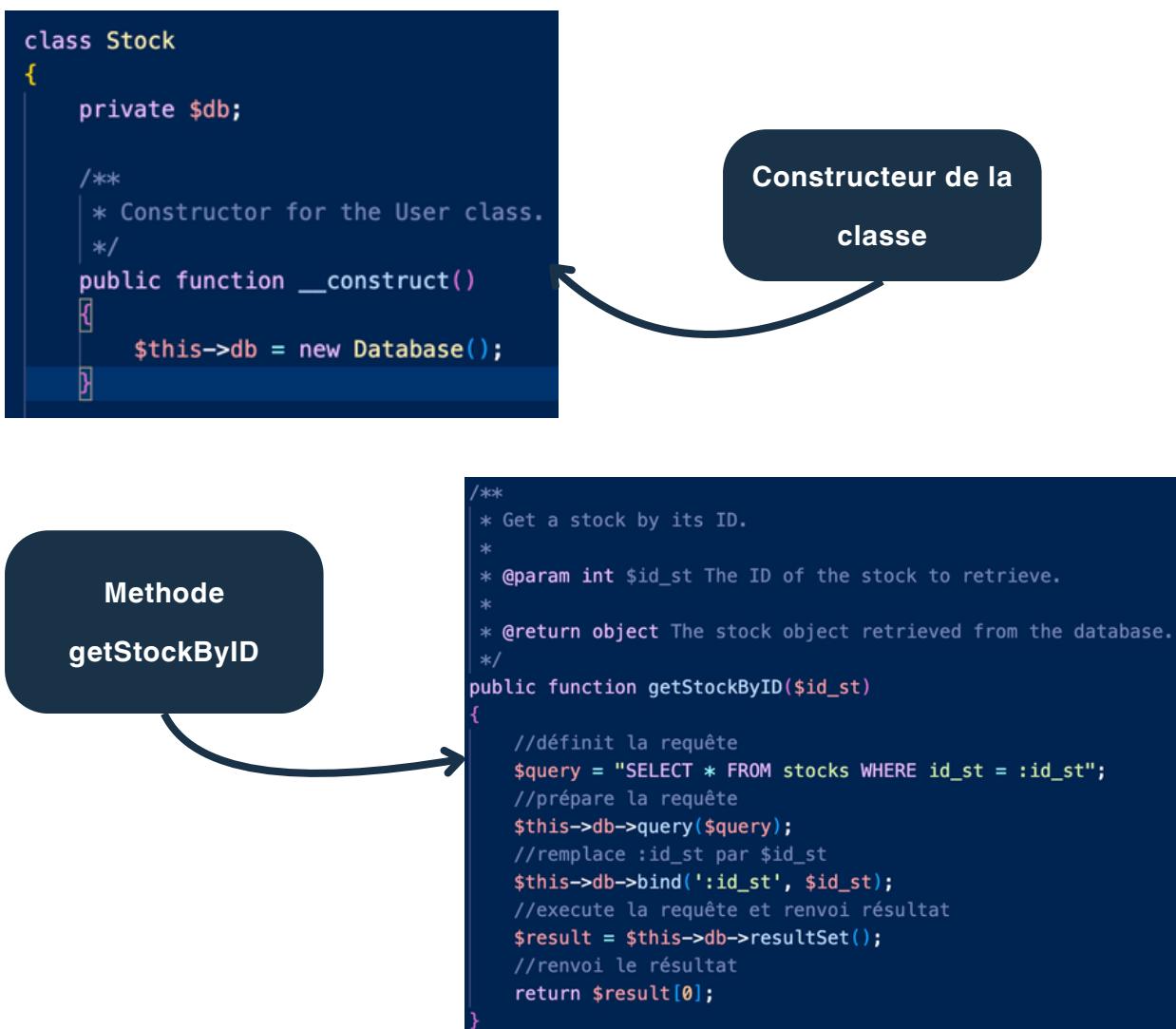
Exécute la requête et
renvoi la réponse

```
/**  
 * Fetches all rows from the result set of the executed query.  
 *  
 * @return array The result set as an array of objects.  
 */  
public function resultSet()  
{  
    $this->execute();  
    return $this->statement->fetchAll(PDO::FETCH_OBJ);  
}
```

Créer les modèles

En MVC, la notion de models correspond aux codes gérant la logique métier. L'objectif est que ces classes contiennent des méthodes prêtes à l'emploi et renvoient les informations nécessaires. Voici à quoi sont constitués ces fichiers PHP. Par exemple, voici ma classe User avec une méthode qui récupère les informations d'un stock via son ID :

Ensuite j'utilise ces méthodes dans les différents models de mon application. Par exemple voici ma classe User avec la méthode qui récupère les informations d'un stock via son ID :



J'ai donc créer une méthode pour chaque fonctions ou besoins que j'ai eu pour mon application.

Créer les contrôleurs

Mon application est en architecture MVC, ce qui veut dire par conséquent qu'elle comporte des contrôleurs. Mes contrôleurs fonctionnent avec des "switch case".

Voici l'exemple de mon contrôleur servant aux commandes :

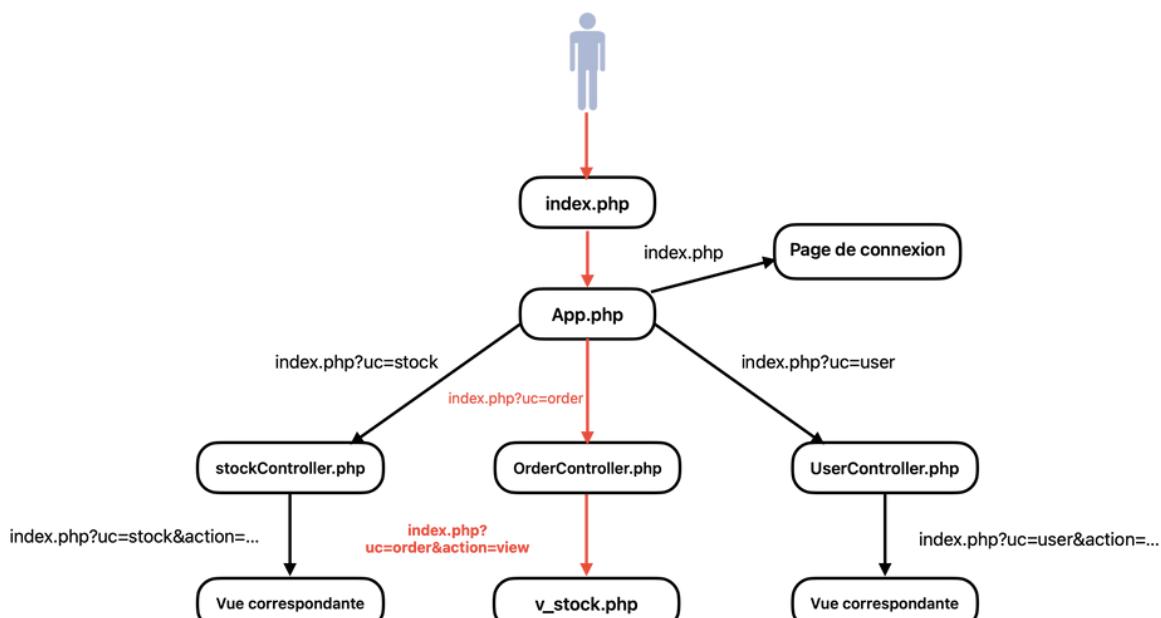
```
<?php
//récupère la valeur du paramètre de requête "action"
$action = $_GET["action"];

//remplit le tableau des commandes
if (empty($_GET["filter"])) {
    $orders = $orderDataAccess->handleFilter("id_co-DESC");
} else {
    $filter = $_GET["filter"];
    $orders = $orderDataAccess->handleFilter($filter);
    $column = explode("-", $filter)[0];
    $order = explode("-", $filter)[1];
}

switch ($action) {
    //si $action = view ...
    case "view":
        include "../src/views/order/v_order.php";
        break;
    //si $action = create ...
    case "create":
        $stocks = $stockDataAccess->getStocksNames();
        include "../src/views/order/v_createOrder.php";
        include "../src/views/order/v_order.php";
        break;

    case "viewDetails":
        $orderDetails = $orderDataAccess->getOrdersDetails(htmlspecialchars($_GET["id_co"]));
        include "../src/views/order/v_orderDetails.php";
        include "../src/views/order/v_order.php";
        break;
}
```

Dans chacun des contrôleurs, nous pouvons retrouver des "cases" qui correspondent à chaque action disponible. Concrètement, voici le chemin d'une requête vers l'URL : index.php?uc=order&action=view



J'ai également créé une "case" pour la validation des différents formulaires. Par exemple, pour la validation d'une commande, je sais que mon formulaire de validation renverra à l'adresse "index.php?uc=order&action=validForm" avec la méthode "POST" et les inputs nommés "valid" et "id". Pour mieux comprendre, voici le formulaire de validation :

```
src > views > order > v_validOrder.php > div.popUpBox--remove.popUpBox
1  <div class="popUpBox--remove popUpBox">
2    <button onclick="closePopUp()"><i class="ti ti-square-x"></i></button>
3    <form action="index.php?uc=order&action=validForm" method="POST">
4      <input type="hidden" name="id" value=<?php echo htmlspecialchars($_GET["id"]); ?>">
5      <input type="submit" name="valid" value="Êtes-vous sûr de valider ?">
6    </form>
7  </div>
```

Nous pouvons voir dans cette capture l'url de destination, la méthode utilisée et les champs "id" et "valid" qui seront envoyés. La récupération de cette requête ce fait de la manière suivante :

```
case "validForm":
    //for valid an order
    if (isset($_POST["valid"]) && isset($_POST["id"])) {
```

Ce code vérifie si les valeurs "id" et "valid" sont définies et exécute le code à l'intérieur des accolades en conséquence. Une vérification de type "if else" est effectuée pour chaque formulaire.

Créer les vues

Les vues dans le MVC sont les pages que l'utilisateur va voir. L'objectif est d'avoir le moins de traitements PHP possible sur ces pages. Voici un exemple de mon code qui affiche les commandes :

```
src > views > order > v_order.php > table.table
1  <a href=". /index.php?uc=order&action=create" class="btnAdd "><i class="ti ti-shopping-cart-plus"></i>Créer une commande</a>
2  <table class="table">
3    <thead>
4      <tr>
5        <th><a href=". /index.php?uc=order&action=view&filter=id_co-<?php echo $column === 'id_co' && $order === 'ASC' ? 'DESC' : 'ASC'; ?>"><?php echo $column === 'id_co' && $order === 'ASC' ? 'ASC' : 'DESC'; ?></a></th>
6        <th><a href=". /index.php?uc=order&action=view&filter=date_co-<?php echo $column === 'date_co' && $order === 'ASC' ? 'DESC' : 'DESC'; ?>"><?php echo $column === 'date_co' && $order === 'ASC' ? 'ASC' : 'DESC'; ?></a></th>
7        <th><a href=". /index.php?uc=order&action=view&filter=statut_co-<?php echo $column === 'statut_co' && $order === 'ASC' ? 'DESC' : 'DESC'; ?>"><?php echo $column === 'statut_co' && $order === 'ASC' ? 'ASC' : 'DESC'; ?></a></th>
8        <th><a href=". /index.php?uc=order&action=view&filter=type_co-<?php echo $column === 'type_co' && $order === 'ASC' ? 'DESC' : 'DESC'; ?>"><?php echo $column === 'type_co' && $order === 'ASC' ? 'ASC' : 'DESC'; ?></a></th>
9        <th><a href=". /index.php?uc=order&action=view&filter=id_u-<?php echo $column === 'id_u' && $order === 'ASC' ? 'DESC' : 'ASC'; ?>"><?php echo $column === 'id_u' && $order === 'ASC' ? 'ASC' : 'DESC'; ?></a></th>
10       <th>Actions</th>
11     </tr>
12   </thead>
13   <tbody>
14     <?php foreach ($orders as $order) { ?>
15       <tr>
16         <td><?php echo $order->id_co; ?></td>
17         <td><?php echo $order->date_co; ?></td>
18         <td><?php echo $order->statut_co; ?></td>
19         <td><i class="ti <?php echo $order->type_co == "sortie" ? 'ti-upload' : 'ti-download'; ?>"></i></td>
20         <td><?php echo $userDataAccess->translateIDtoEmail($order->id_u); ?></td>
21         <?php echo ($SESSION['id_role'] < 3 && $order->statut_co != "validee" && $order->statut_co != "invalidée") ? 
22           "<td>
23             <a href=". /index.php?uc=order&action=rejectOrder&id=$order->id_co"><i class='ti ti-shopping-cart-x'></i></a>
24             <a href=". /index.php?uc=order&action=validOrder&id=$order->id_co"><i class='ti ti-shopping-cart-check'></i></a>
25             <a href=". /index.php?uc=order&action=viewDetails&id_co=$order->id_co"><i class='ti ti-zoom-scan'></i></a>
26           </td>
27           :
28           "<td><a href=". /index.php?uc=order&action=viewDetails&id_co=$order->id_co"><i class='ti ti-zoom-scan'></i></a></td>";
29       </tr>
30     <?php } ?>
31   </tbody>
```

Le code précédent remplit dynamiquement le tableau avec les données de l'objet \$orders, qui provient de la base de données et est fourni par le contrôleur.

Création du login, sécurisation et gestion de utilisateurs

La première action à réaliser est la création d'utilisateurs. GStockB ne permet la création d'utilisateurs qu'au "Super-Administrateur". Cette création se fait à l'aide de ce formulaire, qui est traité par :

```
case "validForm":  
    //for create account  
    if (  
        isset($_POST["nom_u"]) && isset($_POST["prenom_u"]) && isset($_POST["id_role"]) && isset($_POST["email_u"])  
        && isset($_POST["mot_de_passe"]))  
    ) {  
        $nom_u = htmlspecialchars($_POST['nom_u']);  
        $prenom_u = htmlspecialchars($_POST['prenom_u']);  
        $id_role = htmlspecialchars($_POST['id_role']);  
        $email_u = htmlspecialchars($_POST['email_u']);  
        $mot_de_passe = htmlspecialchars($_POST['mot_de_passe']);  
        try {  
            $userCreation = $userDataAccess->createUser($nom_u, $prenom_u, $email_u, $mot_de_passe, $id_role);  
            if ($userCreation) {  
                setcookie("successMessage", "L'utilisateur a été créé avec succès", time() + (100000), "/");  
            } else {  
                setcookie("errorMessage", "L'email est déjà existant en base", time() + (100000), "/");  
            }  
            header("location: ./index.php?uc=user&action=view");  
        } catch (Exception $e) {  
            $userDataAccess->writeLog($e, 'userErrorLogs.log');  
        }  
    }  
}
```

Ensuite la création de l'utilisateur se fait par :

```
/**  
 * Create a new user with the given information.  
 *  
 * @param string $nom_u The last name of the user.  
 * @param string $prenom_u The first name of the user.  
 * @param string $email_u The email address of the user.  
 * @param string $mot_de_passe The password of the user.  
 * @param int $id_role The ID of the role for the user.  
 *  
 * @return void  
 */  
public function createUser($nom_u, $prenom_u, $email_u, $mot_de_passe, $id_role)  
{  
    $existingEmail = self::compareUsersEmails($email_u);  
    //si l'email n'est pas déjà existant...  
    if (! $existingEmail) {  
        //hash le mot de passe  
        $hash = password_hash($mot_de_passe, PASSWORD_DEFAULT);  
        $query = "INSERT INTO `utilisateurs` (`id_u`, `nom_u`, `prenom_u`, `email_u`, `mot_de_passe`, `id_role`)  
        $this->db->query($query);  
        $this->db->bind(':nom_u', $nom_u);  
        $this->db->bind(':prenom_u', $prenom_u);  
        $this->db->bind(':email_u', $email_u);  
        $this->db->bind(':id_role', $id_role);  
        $this->db->bind(':hash', $hash);  
        $this->db->execute();  
        return true;  
    } else {  
        return false;  
    }  
}
```

Ce code de création d'un utilisateur effectue un hachage du mot de passe avant de le stocker dans la base de données. Ce hachage rend le mot de passe illisible et sécurise ainsi les identifiants.

Pour se connecter, l'application demande à l'utilisateur de remplir un formulaire avec une adresse e-mail valide et le mot de passe correspondant. Après la validation de ce formulaire, les données sont envoyées à l'URL "./index.php?uc=validLoginForm", et ce code les récupère :

```
case "validLoginForm":  
    try {  
        $login = $userDataAdapter->login(htmlspecialchars($_POST["email"]), htmlspecialchars($_POST["password"]));  
        if ($login) {  
            header("location: ./index.php?uc=home");  
        } else {  
            $userDataAdapter->writeLog($_POST["email"] . " a échoué la connexion", 'loginErrorLogs.log');  
            setcookie("errorMessage", "Identifiants invalides", time() + (100000), "/");  
            header("location: ./index.php");  
        }  
    } catch (Exception $e) {  
        $userDataAdapter->writeLog($e, 'userErrorLogs.log');  
        setcookie("errorMessage", "Une erreur s'est produite", time() + (100000), "/");  
    }  
    break;
```

Le snippet ci-dessus utilise la méthode "login" de la classe User. Voici cette méthode:

```
/**  
 * Logs in a user with the given email and password.  
 *  
 * @param string $email The email address of the user.  
 * @param string $password The password of the user.  
 *  
 * @return bool True if the login is successful, false otherwise.  
 */  
public function login($email, $password)  
{  
    //récupère les informations de l'utilisateur  
    $query = "SELECT `id_u`, `mot_de_passe`, `id_role`, `active` FROM utilisateurs WHERE email_u = :email";  
    $this->db->query($query);  
    $this->db->bind(':email', $email);  
    $result = $this->db->resultSet();  
    //si un résultat est présent ->  
    if ($result) {  
        $user = $result[0];  
        //vérifie si l'utilisateur peut se connecter en comparant le hash en base et celui du formulaire  
        //et en validant que le compte est actif  
        if (password_verify($password, $user->mot_de_passe) && $user->active == '1') {  
            $_SESSION['id_u'] = $user->id_u;  
            $_SESSION['id_role'] = $user->id_role;  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```

Cette méthode renvoie true si l'utilisateur peut se connecter et initialise les variables de Session "id_u" et "id_role ou renvoi false si il ne peut se connecter.

GStockB ne permet pas l'accès à toutes les fonctionnalités à tous les utilisateurs. Cette restriction est réalisée grâce au champ "id_role" de chaque utilisateur. Après la connexion, l'"id_role" est stocké en tant que variable de session, ce qui le rend accessible à tout moment et assure que l'utilisateur possède les droits appropriés.

Voici un exemple qui vérifie le niveau de droits avec le code qui donne l'accès à la page de gestion des utilisateurs :

```
if ($_SESSION["id_role"] == 1) {
    include "../src/controllers/userController.php";
} else {
    setcookie("errorMessage", "Vous n'avez pas les droits suffisant", time() + (100000), "/");
    header("location: ./index.php?uc=home");
}
```

Créer une commande

La fonctionnalité principale de cette application est la création de commandes. Voici le code qui récupère les données du formulaire de création :

```
//for create an order
if (isset($_POST["type_co"]) && $_POST["stock1"] && isset($_POST["qte1"]) && isset($_POST["numberOfStocks"])) {
    $numberOfStocks = htmlspecialchars($_POST["numberofStocks"]); //nombre de stocks selectionnés
    $selectedStocks = array();

    //vérifie si un stock n'est pas rentré deux fois
    for ($i = 1; $i <= $numberOfStocks; $i++) {
        $stock[$i] = htmlspecialchars($_POST["stock" . $i]);
        $selectedStocks[] = $stock[$i];
    }
    /**
     * Create the details of an order
     *
     * @param int $id_co      The id of the order
     * @param int $id_st      The ID of the stock
     * @param int $qte         The quantity
     *
     * @return void
     */
    public function createOrderDetails($id_co, $id_st, $qte)
    {
        $query = "INSERT INTO `details_commande`(`id_co`, `id_st`, `quantite_details`) VALUES (:id_co, :id_st, :qte);";
        $this->db->query($query);
        $this->db->bind(':id_co', $id_co);
        $this->db->bind(':id_st', $id_st);
        $this->db->bind(':qte', $qte);
        $this->db->execute();
    }
    } else {
        setcookie("errorMessage", "Vous ne pouvez pas créer une commande qui contient deux fois le même stock", time() + (100000));
        header("location: index.php?uc=order&action=view");
    }
}
```

Ce code vérifie d'abord s'il n'y a pas de doublon de stock, puis il crée une commande avec la date, le type (entrée/sortie) et l'ID de l'utilisateur. Après avoir créé la commande, il récupère son ID et crée des entrées dans la table "commandes_details" avec l'ID de chaque stock, l'ID de la commande et les quantités souhaitées. Voici le code qui crée une commande :

```
/**
 * Create an order.
 *
 * @param string $type_co  The type of the order. Possible values: entrée, sortie .
 * @param string $date_co  The date of the order. The format is : Y-m-d H:i:s .
 * @param int $id_u        The ID of the order requestor.
 *
 * @return void
 */
public function createOrder($type_co, $date_co, $id_u)
{
    $query = "INSERT INTO `commandes`(`id_co`, `date_co`, `statut_co`, `type_co`, `id_u`) VALUES (NULL, :date_co, 'en_attente', :type_co, :id_u)";
    $this->db->query($query);
    $this->db->bind(':type_co', $type_co);
    $this->db->bind(':date_co', $date_co);
    $this->db->bind(':id_u', $id_u);
    $this->db->execute();
}
```

Celui qui récupère l'id de la commande via sa date et son créateur :

```
/**  
 * Get an order with her creation date. Need the id of the  
 * requestor order for a better precision  
 *  
 * @param string $date_co    The date of the order. The format is : Y-m-d H:i:s .  
 * @param int $id_u          The ID of the order requestor.  
 *  
 * @return void  
 */  
public function getOrderByDate($date_co, $id_u)  
{  
    $query = "SELECT id_co FROM commandes WHERE date_co = :date_co AND id_u = :id_u;";  
    $this->db->query($query);  
    $this->db->bind(':date_co', $date_co);  
    $this->db->bind(':id_u', $id_u);  
    $result = $this->db->resultSet();  
    return $result[0]->id_co;  
}
```

Et enfin celui qui créer les details de la commande :

```
/**  
 * Create the details of an order  
 *  
 * @param int $id_co    The id of the order  
 * @param int $id_st    The ID of the stock  
 * @param int $qte       The quantity  
 *  
 * @return void  
 */  
public function createOrderDetails($id_co, $id_st, $qte)  
{  
    $query = "INSERT INTO `details_commande` (`id_co`, `id_st`, `quantite_details`) VALUES (:id_co, :id_st, :qte);";  
    $this->db->query($query);  
    $this->db->bind(':id_co', $id_co);  
    $this->db->bind(':id_st', $id_st);  
    $this->db->bind(':qte', $qte);  
    $this->db->execute();  
}
```

Gérer les “Pop-up”

L'expérience utilisateur est un aspect fondamental de tout projet de développement. C'est pourquoi j'ai décidé de créer un système de "pop-up". Ce système intuitif et esthétique est, selon moi, un excellent choix. Tout d'abord, j'ai créé les messages d'erreur et de succès. J'ai utilisé les cookies des navigateurs web à cette fin. Voici un exemple de message d'erreur annonçant l'accès à l'application par un utilisateur non identifié :

```
else {  
    setcookie("errorMessage", "Vous n'êtes pas connecté", time() + (100000), "/");  
    header("location: ./index.php");
```

Le processus est le même pour les messages de succès hormis que le nom du cookie est successMessage

Ensuite ce cookie est traité par le fichier PHP v_head qui est inclus par l'index. Voici son fonctionnement :

```
<?php //si un cookie d'erreur est définit ...
if (isset($_COOKIE["errorMessage"])) :
?>
<div class="popUpBox--error popUpBox">
    <button onclick="closePopUp()"><i class="ti ti-square-x"></i></button>
    <span><?php echo $_COOKIE["errorMessage"]; ?></span>
</div>
```

Pour les pop-ups d'action ou de formulaire, le fonctionnement est différent. Ils sont directement inclus par le contrôleur. Voici un exemple de cette situation avec le formulaire de validation d'une commande (l'ajout de la vue des commandes est simulé pour montrer que la page n'est pas actualisée) :

```
case "validOrder":
    include "../src/views/order/v_validOrder.php";
    include "../src/views/order/v_order.php";
    break;
```

Ces pop-ups sont constitués de <div> qui contiennent toute un bouton en haut à droite servant à l'effacer. Il est également possible de faire disparaître le message en cliquant en dehors de celui-ci. Le processus de fermeture est réalisé par ce code JavaScript dans le fichier v_foot.php:

```
<script>
    //function for close a popUpBox
    function closePopUp() {
        //delete cookie errorMessage
        document.cookie = "errorMessage=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/;";
        document.cookie = "successMessage=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

        const urlParams = new URLSearchParams(window.location.search);
        const actionParam = urlParams.get('action');
        //return to view page if an other action is selected in url
        if (actionParam !== 'view' && actionParam !== null) {
            //update url
            urlParams.set('action', 'view');
            urlParams.delete('id');
            window.history.replaceState({}, '', `${window.location.pathname}?${urlParams}`);
        }
        //desable pop up
        const popUpDiv = document.querySelector(".popUpBox");
        popUpDiv.style.display = "none";
    }

    // listener for remove pop Up when click out off him
    document.addEventListener("click", function(event) {
        const popUpDiv = document.querySelector(".popUpBox");
        if (!popUpDiv.contains(event.target)) {
            // close pop up
            closePopUp();
        }
    });
</script>
```

Gérer les logs

Pour mon application, j'ai mis en place un système de fichiers de logs. Un fichier de log est un fichier texte généré par un programme, un système d'exploitation ou un service afin d'enregistrer des événements, des messages ou des actions significatives qui se produisent lors de l'exécution de ce programme, système ou service. Les fichiers de log sont utilisés pour diagnostiquer les problèmes, effectuer des analyses, suivre l'activité et fournir des informations sur le fonctionnement d'un système.

Dans mon projet, j'utilise principalement un fichier qui enregistre tous les échecs de connexion. Ce fichier contient la date et l'email utilisé. Voici des extraits de code qui me permettent de le réaliser :

```
{} else {}  
    //écriture du log  
    $userDataAccess->writeLog($_POST["email"] . " a échoué la connexion", 'loginErrorLogs.log');  
    //message d'erreur  
    setcookie("errorMessage", "Identifiants invalides", time() + (100000), "/");  
    header("location: ./index.php");  
}
```

Ce code utilise la méthode writeLog de ma classe User. Voici cette méthode :

```
/**  
 * Write a log message to the specified file.  
 *  
 * @param string $message The message to write to the log.  
 * @param string $filename The name of the log file.  
 *  
 * @return void  
 */  
public function writeLog($message, $filename)  
{  
    // !! important, you must have rights to the target directory  
    $logMessage = date('[Y-m-d H:i:s]') . ' ' . $message . PHP_EOL;  
    $outputFile = __DIR__ . '/../logs/' . $filename;  
    file_put_contents($outputFile, $logMessage, FILE_APPEND);  
}
```

Ce code remplit le contenu du fichier de sortie avec le contenu du message.

Deploiement

Afin de rendre ce projet complet et prêt pour une mise en production, j'ai décidé de le déployer en ligne sur un serveur web avec un nom de domaine. Voici les étapes que j'ai suivies :

Création d'un serveur

Le serveur que j'utilise est un serveur Linux Ubuntu hébergé. Il s'agit d'un serveur de type "VPS" (Virtual Private Server) que je loue auprès d'un hébergeur. Ce type de service VPS est pratique car il me fournit un serveur Ubuntu (ou autre) clé en main, avec une adresse IP publique et une connexion SSH.

Installation de apache2

Après avoir obtenu la connexion ssh et mis à jour le serveur j'ai installé Apache2 qui me servira à créer le serveur web. Pour installer Apache2 j'ai utilisé la commande suivante :

```
sudo apt install apache2 -y
```

J'ai également utilisé ces commandes pour démarré le service et pour qu'il démarre également au démarrage du serveur :

```
sudo systemctl start apache2  
sudo systemctl enable apache2
```

Installation de Mariadb

Pour ma base de données j'ai utiliser MariaDB. MariaDB est un Système de Gestion de Base de Données Relationnelle se basant sur SQL. Voici les commandes que j'ai utilisé pour installer le service :

```
sudo apt install mariadb-server -y
```

J'ai utilise la commande suivante pour apporter le minimum de sécurité à la base :

```
sudo mysql_secure_installation
```

Installation de PHP

Afin de permettre au serveur web de fonctionner, vous devez installer PHP avec les extensions pour Apache et MySQL. Voici la commande nécessaire :

```
sudo apt install php libapache2-mod-php php-mysql -y
```

Après l'installation il faut redémarrer le service apache

```
sudo systemctl restart apache2
```

Configurer Mariabd :

Dans les étapes suivantes, j'ai simplement installer MariaDB. Voici maintenant les commandes pour configurer la base :

Création de la base :

```
CREATE DATABASE GStockB;
```

Création d'un utilisateur :

```
GRANT ALL PRIVILEGES ON GStockB.* TO 'gstockb'@'localhost'  
IDENTIFIED BY 'motdepasse'; FLUSH PRIVILEGES;
```

Remplissage de la base :

```
source /chemin/vers/le/script.sql;
```

Configurer PHP

Mon application a été développée pour ne pas nécessiter l'écriture de session_start() dans chaque fichier PHP. Pour réaliser cela, vous devez modifier le fichier php.ini et définir la valeur de la directive session.auto_start à 1 :

```
session.auto_start = 1
```

Configurer Apache et SSL

J'ai ensuite configuré le serveur pour qu'il soit accessible en HTTPS et que le "root" renvoie vers mon dossier "public". Voici les étapes que j'ai réalisées :

Installation de certbot qui servira à générer un certificat SSL avec Let's Encrypt

```
sudo apt install certbot python3-certbot-apache -y
```

Génération du certificat :

```
sudo certbot --apache
```

Pour configurer apache j'ai créer un nouveau fichier de configuration gstockb.conf à l'adresse :

etc/apache2/sites-available/

Ce fichier de configuration comporte les lignes suivantes :

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/public/
    ServerName gstockb.sanjuan.fr
    ServerAlias www.gstockb.sanjuan.fr
    RewriteEngine on
    RewriteCond %{SERVER_NAME} =gstockb.sanjuan-01.fr
    RewriteRule ^ https:// %{SERVER_NAME}%{REQUEST_URI} [END,NE,R=permanent]
</VirtualHost>

<VirtualHost *:443>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www/html/public/
    SSLCertificateFile /etc/letsencrypt/live/gstockb.sanjuan-01.fr/fullchain.pem
    SSLCertificateKeyFile /etc/letsencrypt/live/gstockb.sanjuan-01.fr/privkey.pem
    Include /etc/letsencrypt/options-ssl-apache.conf
</VirtualHost>
```

Importer le code source de l'application

Pour récupérer le code source, j'utilise Git pour cloner mon dépôt distant. Git est installé par défaut sur mon serveur. J'utilise un dossier "tampon" pour intégrer mon code afin d'éviter d'appliquer des modifications en production. Voici la commande pour cloner le dépôt :

```
git clone https://github.com/Crtnsj/BDD-GStockB.git
```

J'ai ensuite une branche nommée "prod" dans ce dépôt, qui est préparée de manière minimale pour la production. La seule modification nécessaire sera les identifiants de connexion à la base de données, qui ne sont pas les mêmes que sur le dépôt distant. Voici la commande pour passer à la branche "prod" :

```
git checkout prod
```

Je copie ensuite ce dossier à la racine du serveur web avec cette commande

```
sudo cp /home/ubuntu/Pulls/GStockB/. /var/www/html/
```

Après un redémarrage du serveur pour être sûr que toutes les informations ont étaient prises en compte le serveur web est déployé. Seul la configuration des entrées A au niveau DNS est à réaliser au près du fournisseur.

Manuel Utilisateur

L'application est hébergé et accessible sur le web à l'url :
<https://gstockb.sanjuan-01.fr/>

Voici des identifiants de connexions :

Administrateur : demo@administrateur.fr

Utilisateur :demo@utilisateur.fr

Mot de passe des deux comptes : dem

Après la connexion, l'utilisateur arrive sur le "Dashboard" où il trouvera différentes informations importantes. Tous les éléments de ce tableau de bord sont cliquables, ce qui permet un accès rapide aux informations.

The screenshot shows the GStockB dashboard with the following sections:

- Voir les stocks**: Shows a list of stocks with the lowest levels. The table includes columns for #, Nom (Name), and quantité (Quantity). Items listed include Aspirine, Masques, Toplexil, Gaviscon, Daflon, Spasfon, Dafalgan, Pansements, Orthèse, Imodium, Xanax, Efferalgan, and Doliprane.
- Voir les commandes**: Shows a list of the last commands with columns for Commande (Command ID), Date (Date), and Utilisateur (User). Commands 155, 154, 153, 152, 151, 150, and 151 are listed with their respective dates and user IDs.
- Gérer les utilisateurs**: Shows a link to manage users.
- Se déconnecter**: Shows a link to log out.
- Stocks les plus populaires**: Shows a list of popular stocks with columns for Stock (Stock) and Nombre de commandes (Number of commands). Items listed are Dafalgan (2), Gaviscon (2), Pansements (2), and Aspirine (1).
- Nombre total de stocks**: Shows a large number 13 indicating the total number of stocks.
- Liste des commandes**: Shows a table of commands with columns for #, Date, Statut (Status), Type (Type), and Utilisateur (User). It lists validée (valid) entries and sortie (exit) entries for users 10 and 10.

Pour créer, modifier, supprimer un stock, il faut se rendre à l'onglet "Voir les stocks" avec ce bouton :



Ensuite un tableau avec les stocks apparaît.

Créer un stock						
#	Nom	Description	Quantité	Type	Actions	
6	Doliprane	Utilisé pour le traitement des douleurs, fièvre, allergies, symptômes du rhume ou état grippal	1738		Médicament	 
9	Aspirine	Acide acétylsalicylique, remède contre la douleur et la fièvre.	100		Médicament	 
24	Efferalgan	Traitemen symptomatique des douleurs d'intensité légère à modérée et/ou des états fébriles.	930		Médicament	 
25	Dafalgan	Traitemen symptomatique des douleurs légères à modérées et de la fièvre.	690		Médicament	 
26	Imodium	Traitemen symptomatique des diarrhées aiguës et chroniques.	840		Médicament	 
27	Spasfon	Traitemen symptomatique des douleurs liées aux troubles fonctionnels du tube digestif et des voies biliaires.	500		Médicament	 
28	Daflon	Traitemen des symptômes en rapport avec l'insuffisance veinolymphatique (jambes lourdes, douleurs, impatiences du primo-débutus),	500		Médicament	 
29	Gaviscon	Traitemen symptomatique du reflux gastro-oesophagienn.	380		Médicament	 
30	Topalexil	Traitemen symptomatique des toux non productives gênantes en particulier à prédominance nocturne.	240		Médicament	 
31	Xanax	Traitemen symptomatique des manifestations anxiuses sévères et/ou invalidantes,	900		Médicament	 
32	Pansements	Dispositif de protection permettant de recouvrir une plaie située sur la peau.	700		Matériel	 
33	Orthèse	Appareillage qui compense une fonction absente ou déficiente, assiste une structure articulaire ou musculaire dans son fonctionnement normal	700		Matériel	 

Pour créer un stock il faut cliquer sur **Créer un stock** :

 Créer un stock

⚠ Nom ⚠

6 Doliprane

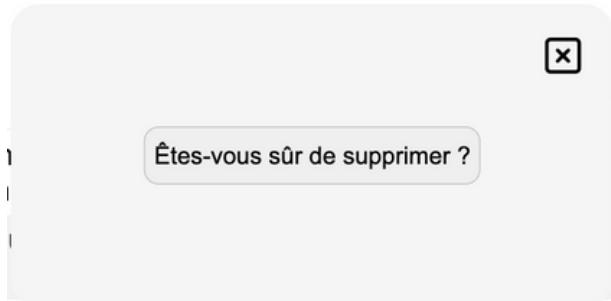
Pour modifier un stock cliquez sur le bouton :  dans la colonne “actions”.

Cette fenêtre apparaît ensuite :

Description	<input type="text" value="Doliprane"/>	<input type="button" value="X"/>
J'utilisé pour	Description	
symptômes	Utilisé pour le traitement des douleurs, fièvre, allergies, symptômes du rhume ou état grippal	
Acide acé		
ièreve.		
Traitement		
égère à n		
Traitement		
nodérées	Quantité : 1738	
Traitement		
chronique	Type	
Traitement		
roubles f	<input type="text" value="Médicament"/>	
iliaires.	<input type="button" value="Valider"/>	
Traitement des symptômes en rapport avec		380
insuffisance veinolymphatique (jambes lourdes, tâoures, impénétration du primaire, déshabitués)		

Pour supprimer un stock cliquez sur le bouton :  dans la colonne “actions”.

Cette fenêtre apparaît ensuite :



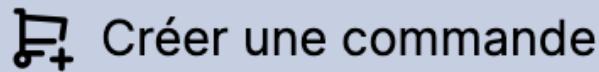
Il est également possible de trier en ascendant ou descendant le tableau en cliquant sur les intitulés de tableau



Pour créer, valider, refuser ou voir les détails d'une commande, il faut se rendre à l'onglet Voir les commandes avec ce bouton :



Une page similaire à la visualisation des stocks apparaît alors. Pour créer une commande il faut cliquer sur ce bouton :



⬇ **Date** ⬇

Ensuite cette fenêtre apparaît :

The screenshot shows a modal window titled "Entrée". It has a dropdown menu set to "Entrée". A button "Ajouter une ligne" is available. Below is a table with columns "Stock" and "Quantité". A row is present with "Dafalgan" in the Stock column and "0" in the Quantité column. At the bottom is a button "Valider la création".

Pour ajouter plusieurs stocks à la commande, il suffit de cliquer sur "Ajouter une nouvelle ligne". Veuillez noter que vous ne pouvez pas ajouter plusieurs fois le même stock à une commande. De plus, la possibilité de choisir si la commande est une entrée ou une sortie est également disponible via une liste déroulante.

Après avoir validé la création, un administrateur peut alors valider la commande avec ce bouton dans la colonne d'action :

Une erreur peut survenir lors de la validation. Cette erreur indique qu'un stock désigné par une commande de sortie est insuffisant pour être réalisé. Dans ce cas, il sera nécessaire de créer une commande d'entrée pour ce stock avant de pouvoir valider la sortie.

L'administrateur peut également refuser une commande avec ce bouton :

Après le refus d'une commande celle-ci sera alors supprimée

La possibilité de voir les détails de la commande se fait via ce bouton :

Cette fenêtre apparaît alors avec un tableau des stocks et leurs quantité :

The screenshot shows a modal window with a table. The columns are "Stock" and "Quantité". Two rows are listed: "Toplexil" with "10" and "Pansements" with "100". At the bottom are buttons "VALIDER" and "SUPPRIMER".

Stock	Quantité
Toplexil	10
Pansements	100

Pour gérer les utilisateurs seul le “super administrateur” a les droits. Voici donc à quoi ressemble l’interface de gestion :

The screenshot shows a table for managing users. The columns are #, Nom, Prénom, Email, Statut, Rôle, and Actions. Three users are listed: "Super Administrateur" (id 10), "Administrateur" (id 24), and "Utilisateur" (id 25). Each user has edit and delete icons in the Actions column. An "Ajouter un utilisateur" button is at the top left. The bottom part of the screenshot shows an edit form for a user with fields for Nom, Prénom, Email, and Rôle.

#	Nom	Prénom	Email	Statut	Rôle	Actions
10	Super	Administrateur	super@administrateur	✓	Super Administrateur	
24	Demo	Admin	demo@administrateur.fr	✓	Administrateur	
25	Demo	User	demo@utilisateur.fr	✓	Utilisateur	

Le super administrateur peut créer un utilisateur via le bouton mis à sa disposition. Ensuite voici la fenêtre qui s'affiche :

Créer un utilisateur

Nom Prenom

Role

Email

Mot de passe

Il doit alors saisir les informations de l'utilisateur. La seule restriction est que l'adresse mail ne peut être égale à un autre utilisateur.

La possibilité de modifier les informations d'un utilisateur est également possible grâce à ce bouton : 

La fenêtre de modification de l'utilisateur s'affiche ensuite:

Modifier un utilisateur

Nom Prenom

Role

Email

Changer le mot de passe

Ancien mot de passe

Mot de passe

Conclusion

Pour conclure, ce projet a été très intéressant à réaliser car il m'a permis de me familiariser avec le PHP. L'approche orientée objet du langage est à la fois formatrice et saine, car elle permet de structurer le code de manière modulaire, favorisant ainsi la réutilisabilité et la maintenabilité du projet.

De plus, ce projet m'a également permis de découvrir la puissance et la facilité d'utilisation des VPS. J'ai pu consolider mes connaissances en HTML, CSS, JavaScript et SQL, ce qui a été bénéfique pour le développement de l'application.

Après plusieurs semaines de travail, je suis très satisfait du résultat. J'ai réussi à créer une application entièrement fonctionnelle, atteignant tous les objectifs fixés.

Je suis enthousiaste à l'idée de proposer des versions futures de ce projet, avec comme objectifs une meilleure gestion des logs et des erreurs, l'ajout d'une partie de gestion des fournisseurs, l'amélioration de la responsivité et l'ajout de nouvelles fonctionnalités. J'aspire également à rendre cette application accessible, en veillant à ce que tous les utilisateurs puissent l'utiliser dans les meilleures conditions possibles.