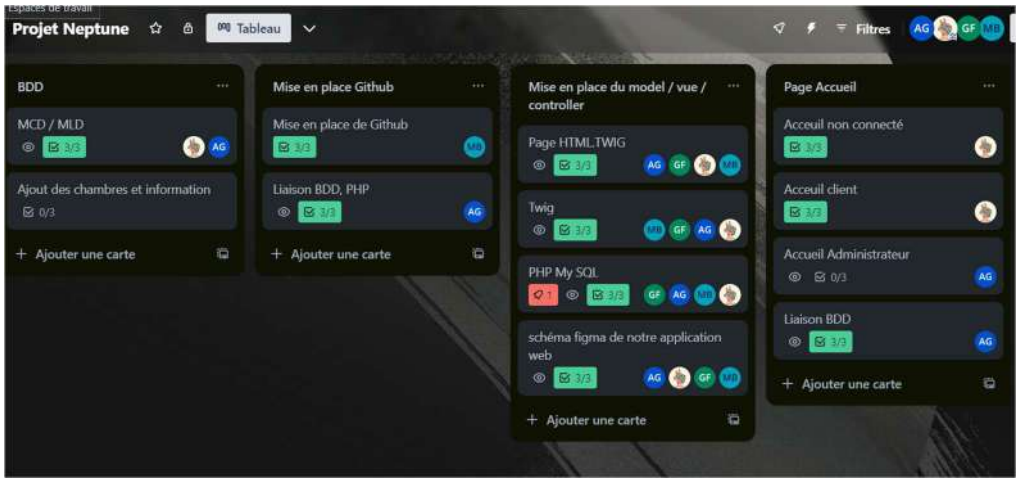
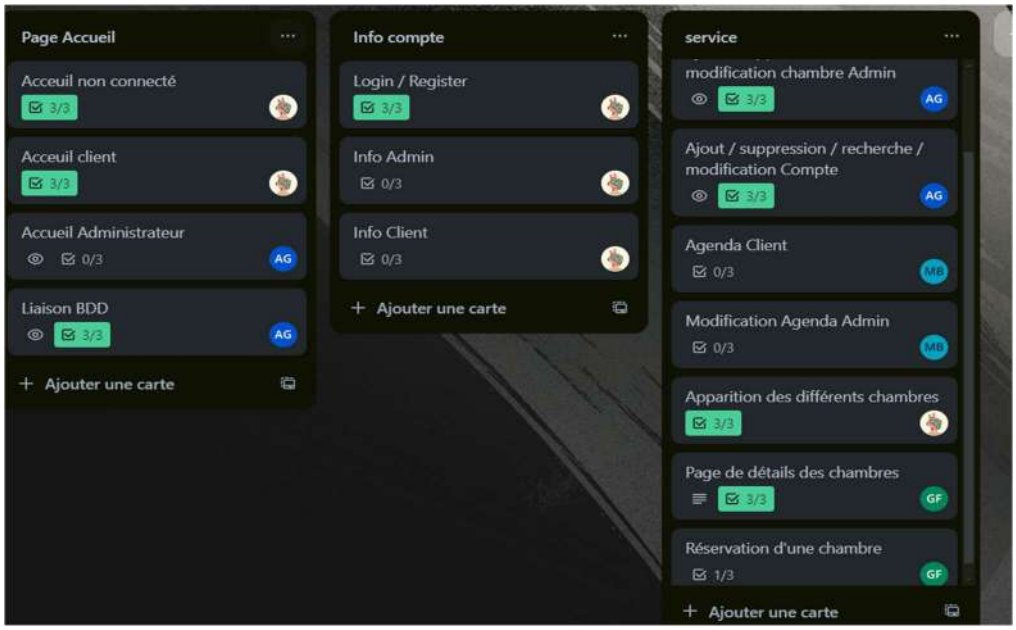
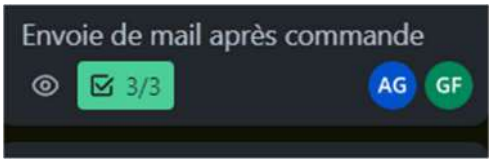
Projet Transversal d’un site pour l’Hôtel Neptune à Arras

Sommaire :

1/ Organisation pour le projet

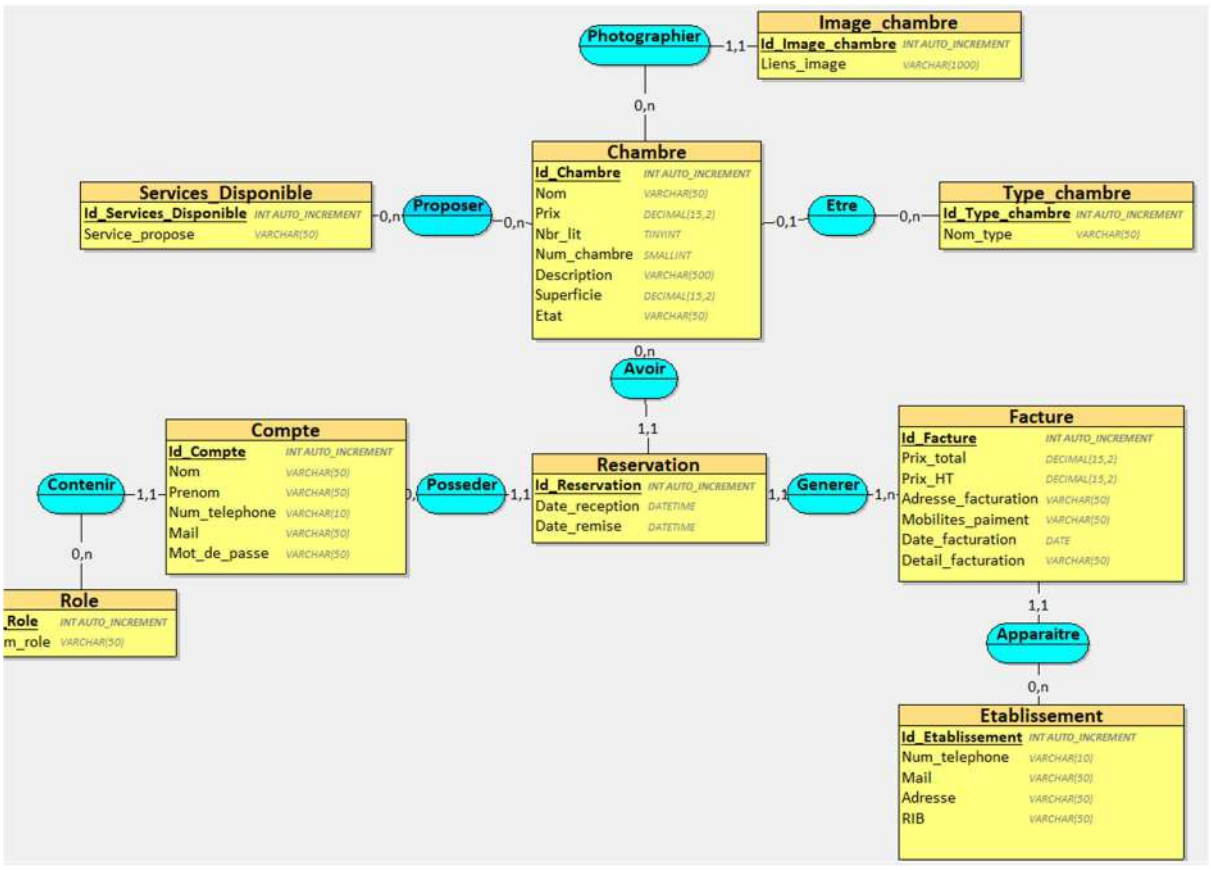






2/ Modèle MCP & connexion Base de données

Tout d'abord nous avons mis en place un modèle conceptuel de base de donnée sur looping afin de voir toutes les tables utiles pour le projet, les jointures, et les différents attributs :

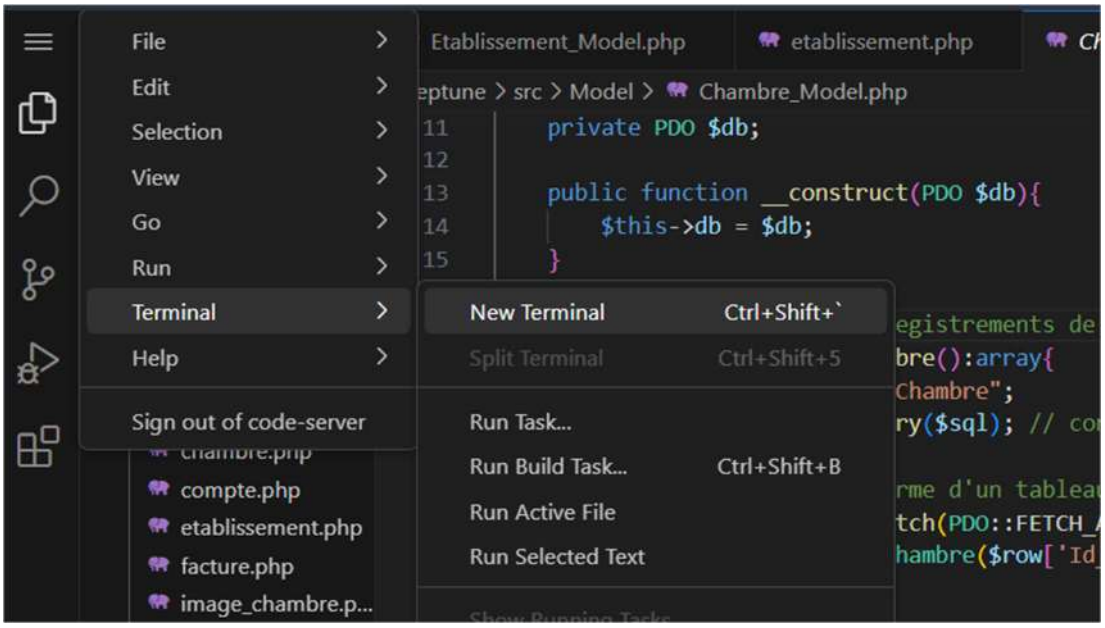


Dans un deuxième temps, dans notre projet VScode :

On ouvre un nouveau terminal, on reste en root :

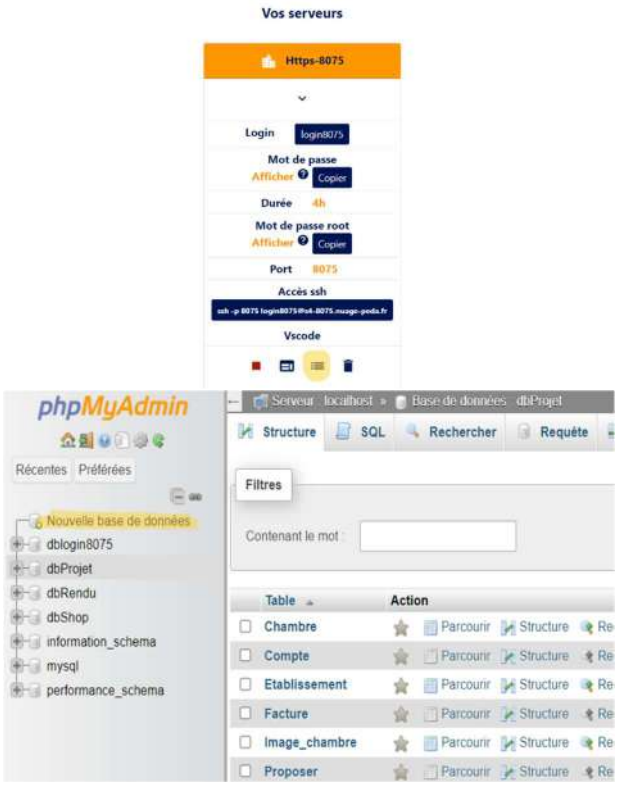
On donne tous les droits à toutes les bases de données de notre compte PhPMysql :

- ☐ Mysql
- ☐ grant all privileges on *.* to ‘login8075’@’localhost’ ; # donne les pleins pouvoir à un user quand il se connecte à cette machine local0
- ☐ flush privileges ; # je confirme mes modifications

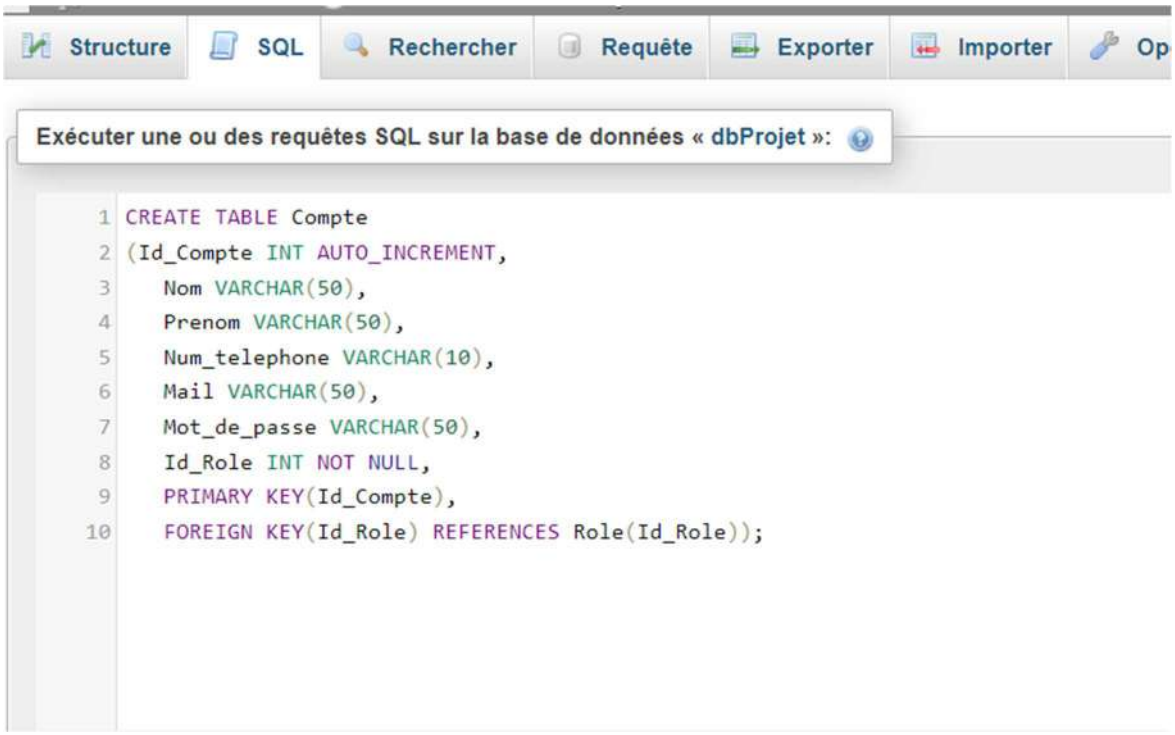


Une fois toutes les commandes effectuées, on peut créer notre base de donnée sur PhPMysql avec tous les droits de modifications, de suppressions etc.

Après avoir donné tous les privilèges on peut aller dans phpMyAdmin, créer une nouvelle base de données et créer nos tables :



Dans « SQL » on écrit notre code SQL pour créer les différentes tables de notre base de données avec les attributs, les clés primaires et étrangères… Par exemple pour la table « Compte » :




```
CREATE TABLE Role(
  Id_role INT AUTO_INCREMENT,
  Nom_role VARCHAR(50),
  PRIMARY KEY(Id_role)
);

CREATE TABLE Type_chambre(
  Id_type_chambre INT AUTO_INCREMENT,
  Nom_type VARCHAR(50),
  PRIMARY KEY(Id_type_chambre)
);

CREATE TABLE Services_Disponible(
  Id_services_disponible INT AUTO_INCREMENT,
  Service_propose VARCHAR(50),
  PRIMARY KEY(Id_services_disponible)
);

CREATE TABLE Etablissement(
  Id_etablissement INT AUTO_INCREMENT,
  Num_telephone VARCHAR(10),
  Mail VARCHAR(50),
  Adresse VARCHAR(50),
  RIB VARCHAR(50),
  PRIMARY KEY(Id_etablissement)
);

CREATE TABLE Compte(
```

```
  Id_compte INT AUTO_INCREMENT,
  Nom VARCHAR(50),
  Prenom VARCHAR(50),
  Num_telephone VARCHAR(10),
  Mail VARCHAR(50),
  Mot_de_passe VARCHAR(50),
  Id_role INT NOT NULL,
  PRIMARY KEY(Id_compte),
  FOREIGN KEY(Id_role) REFERENCES Role(Id_role)
);

CREATE TABLE Chambre(
  Id_chambre INT AUTO_INCREMENT,
  Nom VARCHAR(50),
  Prix DECIMAL(15,2),
  Nbr_lit TINYINT,
  Num_chambre SMALLINT,
  Description VARCHAR(500),
  Superficie DECIMAL(15,2),
  Etat VARCHAR(50),
  Id_type_chambre INT,
  PRIMARY KEY(Id_chambre),
  FOREIGN KEY(Id_type_chambre) REFERENCES Type_chambre(Id_type_chambre)
);

CREATE TABLE Facture(
  Id_facture INT AUTO_INCREMENT,
  Prix_total DECIMAL(15,2),
  Prix_HT DECIMAL(15,2),
  Adresse_facturation VARCHAR(50),
  Mobilites_paiment VARCHAR(50),
  Date_facturation DATE,
  Detail_facturation VARCHAR(50),
  Id_etablissement INT NOT NULL,
  PRIMARY KEY(Id_facture),
  FOREIGN KEY(Id_etablissement) REFERENCES Etablissement(Id_etablissement)
);

CREATE TABLE Image_chambre(
  Id_image_chambre INT AUTO_INCREMENT,
  Liens_image VARCHAR(1000),
  Id_chambre INT NOT NULL,
  PRIMARY KEY(Id_image_chambre),
  FOREIGN KEY(Id_chambre) REFERENCES Chambre(Id_chambre)
);

CREATE TABLE Reservation(
```

```
  Id_reservation INT AUTO_INCREMENT,
  Date_reception DATETIME,
  Date_remise DATETIME,
  Id_facture INT NOT NULL,
  Id_chambre INT NOT NULL,
  Id_compte INT NOT NULL,
  PRIMARY KEY(Id_reservation),
  FOREIGN KEY(Id_facture) REFERENCES Facture(Id_facture),
  FOREIGN KEY(Id_chambre) REFERENCES Chambre(Id_chambre),
  FOREIGN KEY(Id_compte) REFERENCES Compte(Id_compte)
);

CREATE TABLE Proposer(
  Id_chambre INT,
  Id_services_disponible INT,
  PRIMARY KEY(Id_chambre, Id_services_disponible),
  FOREIGN KEY(Id_chambre) REFERENCES Chambre(Id_chambre),
  FOREIGN KEY(Id_services_disponible) REFERENCES
Services_Disponible(Id_services_disponible)
);
```

Une fois la Base de données entièrement construite, on peut retourner dans notre projet VScode. Notre projet étant sous la forme **Model/Vue/Contrôleur**, nous allons donc l'organiser de la même manière pour implémenter la base de données.

Tout d'abord nous devons connecter la base de donnée au projet, pour faire cela :

Dans VisualCode répertoire :

- Service/DependencyContainer.php

On crée une Nouvelle Méthode de la classe pour établir une connexion entre le code et la BDD :

```php
    private function createPDOInstance(){
        try{
            $pdo = new PDO('mysql:host='.$_ENV['DB_HOST']. ';dbname='.$_ENV['DB_NAME'].';charset=utf8', $_ENV['DB_USER'],$_ENV['DB_PASS']);
            $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            return $pdo;
        }
        catch(PDOException $e){
            throw new \Exception("PDO connection error".$e->getMessages);
        }
    }

}
?>
```

On peut faire cela grâce à PDO que l'on vient importer tout en haut de notre code et qui permet de créer des instances de base de données

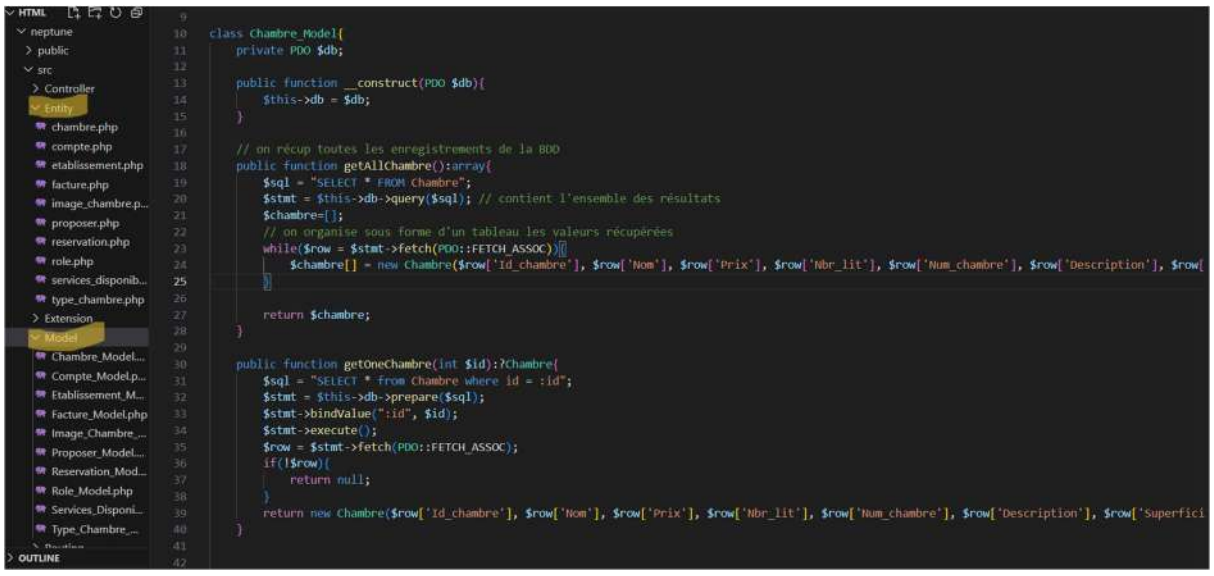
```php
<?php
namespace MyApp\Service;

use PDO;
```

On constate que la méthode « createPDOinstance » utilise plusieurs paramètres pour créer une nouvelle installe de PDO stockée dans « .ENV ».

En effet dans notre projet on vient créer un fichier .env.local à la racine de notre projet, dans lequel on va mettre les informations nécessaires à la connexion de la BDD avec les identifiants de son créateur : (on mettra ce ficher dans .gitignore par sécurité, ce fichier ne sera pas envoyé sur github).

```
DB_HOST=localhost
DB_NAME=dbShop
DB_USER=login8075
DB_PASS=mot_de_passe
```

Dans les répertoires de notre projet on crée dans 'src' le répertoire « model » ainsi que le répertoire « entity » avec les fichier .php correspondant pour chaque tables de notre BDD.



On peut voir l'utilité d'un model ci-dessus avec le model de la table « Chambre » qui va permettre d'écrire et envoyer du code SQL à notre base de données pour récupérer les enregistrements de cette table soit entièrement grâce à la méthode «getAllChambre », soit un enregistrement en particulier récupérable par sa clé primaire «id » avec la méthode «getOneChambre ». On peut faire tout cela grâce à la connexion effectuée à la base de données avec PDO vu précédemment.

Nous prendrons l'exemple de la table Chambre tout le long de cette explication.

L'entity de chaque table quant-à-elle permettra de jouer avec les données spécifiées en paramètre ; de les récupérer ; de les modifier etc.

Exemple dl'Entity Chambre :

```php
<?php

declare(strict_types =1);

namespace MyApp\Entity;

class Chambre{
    private ?int $id = null; // vide tant qu'il n'a pas récup de valeur de la BDD
    private string $nom;
    private float $prix;
    private int $nbr_lit;
    private int $num_chambre;
    private string $description;
    private float $superficie;
    private string $etat;
    private ?type_chambre $id_type_chambre = null;


    public function __construct(?int $id, string $nom, float $prix, int $nbr_lit, int $num_chambre, string $description, float $superficie, string $etat, ?type_chambre $id_type_chambre){
        $this->id = $id;
        $this->nom = $nom;
        $this->id_chambre = $id_chambre;
        $this->prix = $prix;
        $this->nbr_lit = $nbr_lit;
        $this->num_chambre = $num_chambre;
        $this->description = $description;
        $this->superficie = $superficie;
        $this->etat = $etat;
        $this->id_type_chambre = $id_type_chambre;
    }

    // get pour récuperer des données dans la BDD
    public function getId():?int{
        return $this->id;
    }
    public function setId(?int $id):void{
        $this->id = $id;
    }

    public function getNom():string{
        return $this->nom;
```

```php
    }
    public function setNom(string $nom):void{
        $this->nom = $nom;
    }

    public function getPrix():float{
        return $this->prix;
    }
    public function setPrix(float $prix):void{
        $this->prix = $prix;
    }

    public function getNbr_lit():int{
        return $this->nbr_lit;
    }
    public function setNbr_lit(int $nbr_lit):void{
        $this->nbr_lit = $nbr_lit;
    }

    public function getNum_chambre():int{
        return $this->num_chambre;
    }
    public function setNum_chambre(int $num_chambre):void{
        $this->num_chambre = $num_chambre;
    }

    public function getDescription():string{
        return $this->description;
    }

    public function setDescription(string $description):void{
        $this->description = $description;
    }

    public function getSuperficie():float{
        return $this->superficie;
    }
    public function setSuperficie(float $superficie):void{
        $this->superficie = $superficie;
    }

    public function getEtat():string{
        return $this->etat;
    }
    public function setEtat(string $etat):void{
        $this->etat = $etat;
    }

    public function getId_Type_Chambre():?type_chambre{
        return $this->id_type_chambre;
    }
    public function setId_Type_Chambre(?type_chambre $id_type_chambre):void{
```

```
        $this->id_type_chambre = $id_type_chambre;
    }

}
```

Dans Service -> Dependency Container :

On ajoute une méthode createInstance en précisant dans le switch notre model de BDD permettant de créer la connexion à une table précise de notre BDD.

```
   private function createInstance($key)
    {
        switch ($key) {

            case 'PDO': return $this->createPDOInstance();
            case 'ChambreModel':
                $pdo = $this->get('PDO');
                return new ChambreModel($pdo);

            default:
                throw new \Exception("No service found for key: " . $key);
        }
    }
```

On récupère les infos de notre table de la BDD dans DefaultController.php, on crée la variable locale de celle-ci et son constructeur :

```
<?php
declare (strict_types = 1);
namespace MyApp\Controller;
use MyApp\Service\DependencyContainer;
use Twig\Environment;
use MyApp\Model\ChambreModel;


class DefaultController
{
    private $twig;
    private $ChambreModel;

    public function __construct(Environment $twig, DependencyContainer
$dependencyContainer)
    {
        $this->twig = $twig;
        $this->ChambreModel = $dependencyContainer->get('ChambreModel');
    }
```

Cependant notre base de données à des tables liés entre-elles par des jointures, et des clés étrangères, on doit donc prendre en compte cette jointure dans notre code vscode :

Tout d'abord dans srv -> model -> chambre.model dans la requête sql envoyé à l'aide de PDO on rajoute une jointure entre la table chambre et type chambre pour récupérer les données de la clé étrangère id_type_chambre :

SELECT Chambre.*, Type_chambre.* FROM Chambre JOIN Type_chambre.Id_type_chambre = Chambre.Id_type_chambre ;

```php
use MyApp\Entity\chambre;
use MyApp\Entity\type_chambre;
use PDO;

class Chambre_Model{
    private PDO $db;

    public function __construct(PDO $db){
        $this->db = $db;
    }

    // on récup toutes les enregistrements de la BDD
    public function getAllChambre():array{
        $sql = "SELECT Chambre.*, Type_chambre.* FROM Chambre Join Type_chambre ON Type_chambre.Id_type_chambre = Chambr
        $stmt = $this->db->query($sql); // contient l'ensemble des résultats
        $chambre=[];
        // on organise sous forme d'un tableau les valeurs récupérées
        while($row = $stmt->fetch(PDO::FETCH_ASSOC)){

            $typeChambre = null;
            if ($row['Id_type_chambre'] !== null) {
                $typeChambre = new Type_chambre($row['Id_type_chambre'], $row['Nom_type']);
            }
            $chambre[] = new Chambre($row['Id_chambre'], $row['Nom'], floatval($row['Prix']), $row['Nbr_lit'], $row['Num
        }

        return $chambre;
    }
```

Dans les méthodes getAllChambre et getOneChambre on crée une instance de la classe Type_chambre pour récupérer le contenu de cette table et l'insérer dans la table chambre.

```php
$row['Num_chambre'], $row['Description'], floatval($row['Superficie']), $row['Etat'], $typeChambre);
```

Dans les paramètres de la classe Chambre, on vient mettre l'instance de la classe Type_chambre.

On aura au préalable créé un argument type_chambre de type Type_Chambre dans src -> Entity -> chambre.php

```php
class Chambre{
    private ?int $id = null; // vide tant qu'il n'a pas récup de valeur de la BDD
    private string $nom;
    private float $prix;
    private int $nbr_lit;
    private int $num_chambre;
    private string $description;
    private float $superficie;
    private string $etat;
    private ?type_chambre $id_type_chambre = null;
```

Comment récupérer les valeurs d'une table spécifique sur une vue ?

Dans service -> DependencyContainer :

On crée une méthode create instance qui va lancer la connexion à la base de donnée avec l'appelle de createPDOInstance() ;

```php
private function createInstance($key)
{
    switch ($key) {

        case 'PDO': return $this->createPDOInstance();

        case 'Image_Chambre_Model' :
            $pdo = $this->get('PDO');
            return new Image_Chambre_Model($pdo);

        case 'Type_Chambre_Model' :
            $pdo = $this->get('PDO');
            return new Type_Chambre_Model($pdo);

        case 'Chambre_Model' :
            $pdo = $this->get('PDO');
            return new Chambre_Model($pdo);

        default:
            throw new \Exception("No service found for key: " . $key);
    }
}
```

Puis dans le switch on ajoute un case pour pointer vers chaque model de chaque table de notre BDD

On oublie pas à chaque fois d'importer les bon fichiers pour accéder aux appels de class/méthodes

```php
<?php
namespace MyApp\Service;

use PDO;
use MyApp\Model\Image_Chambre_Model;
use MyApp\Model\Type_Chambre_Model;
use MyApp\Model\Chambre_Model;
```

Dans src-> Controller -> DefaultController :

On crée une variable local pour stocker chaque Model de table, puis dans les méthodes qui redirige les données vers les page.twig, on les valeurs nécessaires dans nos table_Model et on les ajoutent au tableau PHP pour pouvoir les récupérer sur notre page twig :

```php
class DefaultController
{
    private $twig;
    private $Image_Chambre_Model;
    private $Type_Chambre_Model;
    private $Chambre_Model;

    public function __construct(Environment $twig, DependencyContainer $dependencyContainer)
    {
        $this->twig = $twig;
        $this->Image_Chambre_Model = $dependencyContainer->get('Image_Chambre_Model');
        $this->Type_Chambre_Model = $dependencyContainer->get('Type_Chambre_Model');
        $this->Chambre_Model = $dependencyContainer->get('Chambre_Model');
    }

    public function accueil()
    {
        $image_chambre = $this->Image_Chambre_Model->getAllImageChambre();
        $type_chambre = $this->Type_Chambre_Model->getAllTypeChambre();
        $chambre = $this->Chambre_Model->getAllChambre();
        echo $this->twig->render('defaultController/accueil.html.twig', ['image_chambre'=>$image_chambre, 'type_chambre'=>$type_chambre, 'chambre'=>
    }
```

Dans la page .twig on peut désormais récupérer les valeurs souhaitées :

```twig
// element de la table image_chambre de notre BDD
    {% for image in image_chambre %}
        {{image.id}} {{image.lien_image}}
    {% endfor %}

    {% for types in type_chambre %}
        {{types.id}} {{types.type}}
    {% endfor %}

    {% for ch in chambre %}
        {{ch.id}} {{ch.nom}} {{ch.prix}} {{ch.description}}
    {% endfor %}
```

3/ Gestion de connexion et d'inscription

Pour l'inscription et la connexion on a fait de la récupération de données.

```
<body>
    <div class="login-box">
        <h2>S'inscrire</h2>
        <br>
        <h3>veuillez rentrer vos informations s'il vous plait</h3>
        <form action="" method="post">
            <div class="form-group">
                <label for="nom">Nom :</label>
                <input type="text" id="nom" name="nom" required placeholder="nom">
            </div>
            <div class="form-group">
                <label for="prenom">prenom :</label>
                <input type="text" id="prenom" name="prenom" required placeholder="prenom">
            </div>
            <div class="form-group">
                <label for="num_telephone">Numéro de telephone :</label>
                <input type="text" id="num_telephone" name="num_telephone" required placeholder="Numéro de telephone">
            </div>
            <div class="form-group">
                <label for="mail">adresse mail :</label>
                <input type="mail" id="mail" name="mail" required placeholder="mail">
            </div>
            <div class="form-group">
                <label for="mot_de_passe">mot de passe :</label>
                <input type="password" id="mot_de_passe" name="mot_de_passe" required placeholder="mot de passe">
            </div>
            <p>Vous êtes déjà inscript ?
                <a href="index.php?page=connexion">
                    connectez-vous
                </a>
            </p>
            <div class="form-group">
                <button type="submit" class="btn btn-primary" >S'enregistrer</button>
            </div>
        </div>
    </body>
```

```
<body>
    <div class="login-box">
        <h2>Connexion</h2>
        <form action="" method="post">
            <div class="form-group">
                <label for="mail">Adresse mail :</label>
                <input type="email" id="mail" name="mail" required placeholder="mail">
            </div>
            <div class="form-group">
                <label for="mot_de_passe">Mot de passe :</label>
                <input type="password" id="mot_de_passe" name="mot_de_passe" required placeholder="Mot de Passe">
            </div>
            <p>Vous n'avez pas de compte ?
                <a href="index.php?page=inscription">
                    Inscrivez-vous
                </a>
            </p>
            <div class="form-group">
                <button type="submit" class="btn btn-primary">Se connecter</button>
            </div>
        </div>
    </body>
</body>
```

Une fois que l'on a récupéré les données on vient juste transmettre les données à la BDD pour l'inscription. On vient les récupérent dans le default contrôleur avec la méthode inscription. Puis on vient transmettre les données dans le compte_model ou à partir d'ici on va mettre les informations dans la BDD.

```
public function inscription()
{
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $nom = filter_input(INPUT_POST, 'nom', FILTER_SANITIZE_STRING);
        $prenom = filter_input(INPUT_POST, 'prenom', FILTER_SANITIZE_STRING);
        $num_telephone = filter_input(INPUT_POST, 'num_telephone', FILTER_SANITIZE_STRING);
        $mail = filter_input(INPUT_POST, 'mail', FILTER_SANITIZE_STRING);
        $mot_de_passe = filter_input(INPUT_POST, 'mot_de_passe', FILTER_SANITIZE_STRING);
        $id_role = filter_input(INPUT_POST, 'id_role', FILTER_SANITIZE_NUMBER_INT);

        if (!empty($_POST['nom']) && !empty($_POST['prenom']) && !empty($_POST['num_telephone']) && !empty($_POST['mail']) && !empty($_POST['mot_de_passe'])) {
            $id_role = $this->role_Model->getOneRole(2);
            // on crée le nouvel enregistrement de compte
            $compte = new Compte(null, $nom, $prenom, $num_telephone, $mail, $mot_de_passe, $id_role);
            $success = $this->compte_Model->createCompte($compte);
            if ($success) {
                header('Location: index.php?page=accueil_client');
            }
        }
    }
    echo $this->twig->render('defaultController/inscription.html.twig', []);
}
```

```php
// méthode pour ajout erun enregistrement à la table compte
public function createCompte(Compte $compte): bool
{
    $sql = "INSERT INTO Compte (Id_compte, Nom, Prenom, Num_telephone, Mail, Mot_de_passe, Id_role) VALUES (:Id_compte, :Nom, :Prenom, :Num_telephone, :Mail, :Mot
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':Id_compte', $compte->getId(), PDO::PARAM_STR);
    $stmt->bindValue(':Nom', $compte->getNom(), PDO::PARAM_STR);
    $stmt->bindValue(':Prenom', $compte->getPrenom(), PDO::PARAM_STR);
    $stmt->bindValue(':Num_telephone', $compte->getNum_telephone(), PDO::PARAM_STR);
    $stmt->bindValue(':Mail', $compte->getMail(), PDO::PARAM_STR);

    // Hacher le mot de passe avant de l'insérer dans la base de données
    $mot_de_passe_hache = password_hash($compte->getMot_de_passe(), PASSWORD_DEFAULT);
    $stmt->bindValue(':Mot_de_passe', $mot_de_passe_hache, PDO::PARAM_STR);

    $stmt->bindValue(':Id_role', $compte->getId_role()->getId_role(), PDO::PARAM_STR);
    return $stmt->execute();
}
```

Pour la connexion, on récupère l'adresse mail et on vérifie si c'est la bonne. On vient ensuite regarder le mdp avec la fonction PHP password vérify. Cette fonction permet d'aller chercher le mdp dans la bdd et de vérifier si le mdp haché est le même.

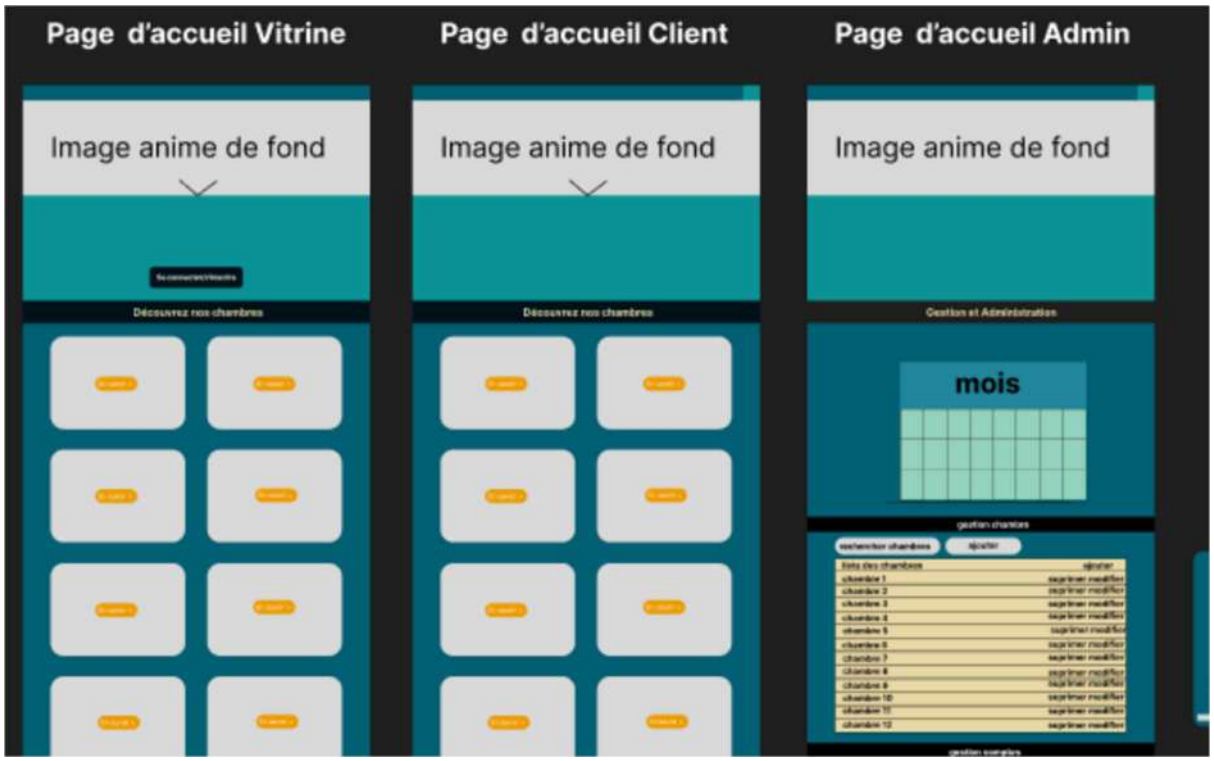
```php
public function connexion()
{
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $mail = filter_input(INPUT_POST, 'mail', FILTER_VALIDATE_EMAIL);
        $mot_de_passe = $_POST['mot_de_passe'];
        if ($mail === false) {
            $_SESSION['message'] = 'Utilisateur ou mot de passe erroné';
            header('Location: index.php?page=connexion');
            exit;
        } else {
            $compte = $this->compte_Model->getCompteByMail($mail);
            if (!$compte) {
                $_SESSION['message'] = 'Utilisateur ou mot de passe erroné';
                header('Location: index.php?page=connexion');
            } else {
                if ($compte->verifyMot_de_passe($mot_de_passe)) {
                    $_SESSION['connexion'] = $compte->getMail();
                    $_SESSION['id_role'] = $compte->getId_role();
                    header('Location: index.php?page=accueil_client');
                    exit;
                } else {
                    $_SESSION['message'] = 'Utilisateur ou mot de passe erroné';
                    header('Location: index.php?page=connexion');
                    exit;
                }
            }
        }
    }
    echo $this->twig->render('defaultController/connexion.html.twig', []);
}
```

```php
public function verifyMot_de_passe(string $mot_de_passe): bool
{
    return password_verify($mot_de_passe, $this->mot_de_passe);
}
```

```php
public function getCompteByMail(string $mail): ?Compte
{
    $sql = "SELECT Compte.*, Role.* FROM Compte JOIN Role ON Role.Id_role = Compte.Id_role WHERE Mail = :mail";
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':mail', $mail); // Utilisation de ':mail' au lieu de ':Mail'
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$row) {
        return null;
    }
    $role = null;
    if ($row['Id_role'] !== null) {
        $role = new Role($row['Id_role'], $row['Nom_role']);
    }
    return new Compte($row['Id_compte'], $row['Nom'], $row['Prenom'], $row['Num_telephone'], $row['Mail'], $row['Mot_de_passe'], $role);
}
```

4/ Création des pages d’accueils de notre site

On a commencé par définir l’accueil à quoi cela devrait ressembler sur Figma.



Puis on a utilisé Webpik.com pour créer l'image de fond.

Une fois cela fait, on s'est occupé de la mise en forme et de ce à quoi cela devait ressembler.

La page d'accueil vitrine (non-connecté) et celle des clients (connecté) étant la même avec comme seul différence le fait d'avoir un bouton se connecter :

```
{% extends "base.html.twig" %}

{% block title %} {{ parent() }} Accueil {% endblock %}

{% block content %}

<ul class="navbar">
   <li><a>Accueil</a></li>
   <li><a href="index.php?page=connexion">Se connecter/s'inscrire</a></li>
</ul>
<div class="cyan">
<br><br><br><br>
   <img src="images/image_neptune.png" width="100%">
   <br><br><br><br>
   <div>
      <a href="index.php?page=connexion"><button  class="bouton">Se connecter/s'inscrire</button></a>
   </div>
   <br>
   <div class="noir_centre">
      <h2 class="or">Découvrez nos chambres</h2>

      {% for i in range(0, image_chambre|length - 1) %}
         <a href="index.php?page=connexion">
            <img src="{{ image_chambre[i].lien_image }}" width="50%">
         </a>
         <p class="text">{{ chambre[i].description }}</p>
      {% endfor %}

      <br>

   </div>
</div>
<div>
</div>
<div class="noir_centre">
   <h2 class="or">Notre restaurant</h2>
</div>
<div class="img_rest">
   <img src="images/restaurant.jpg"width="100%">
   <img src="images/bar.jpg"width="100%">
   <p class="text">Près de la gare et du centre historique, le 58 revendique fièrement son ancrage local. Des gravures et papiers buvards aux murs à l'ancien plan d'Arras au plafond, ce lounge bar et restaurant
</div>
```

```
<div class="noir_centre">
   <h2 class="or">Notre Salle de réception</h2>
</div>
<div class="img_recep">
   <img src="images/reception1.png"width="100%">
   <img src="images/reception2.png"width="100%">
   <p class="text" width="100%">Près de la gare et du centre historique, le 58 revendique fièrement son ancrage local. Des gravures et papiers buvards aux murs à l'ancien plan d'Arras au plafond, ce lounge bar
</div>

<div class="noir_centre">
   <h2 class="or">Notre Piscine</h2>
</div>
<div class="img_recep">
   <img src="images/piscine.jpg"width="100%">
</div>

<div class="noir_centre">
<br><br><br><br>
   <h2 class="or">Nous trouver</h2>
</div>
<div class="img_recep">
   <img src="images/localisation.png"width="100%">
</div>
{% endblock %}
```

Pour afficher les chambres on a rajouté cette ligne de commande pour récupérer les infos de la bdd et les afficher.

Dans defaultcontroller on récupère les données, on les met dans le tableau :

```php
public function accueil()
{
    $image_chambre = $this->image_Chambre_Model->getAllImageChambre();
    $reservation = $this->reservation_Model->getAllReservation();
    $chambre = $this->chambre_Model->getAllChambre();
    echo $this->twig->render('defaultController/accueil.html.twig', ['image_chambre' => $image_chambre, 'reservation' => $reservation, 'chamb
}
```

Sur nos pages accueil.hmtl.twig on récupère les variables et array passé dans le tableau :

```twig
{% for i in range(0, image_chambre|length - 1) %}
    <a href="index.php?page=connexion">
        <img src="{{ image_chambre[i].lien_image }}" width="50%">
    </a>
    <p class="text">{{ chambre[i].description }}</p>
{% endfor %}
```

5/ Gestion et création de chambres et de comptes CRUD

Gestion des comptes dans un panel admin

CRUD = **Create, Retrieve, Update, Delete**

Nous avons déjà mis en place le retrieve = récupérer vu précédemment ;

Attaquons maintenant aux méthodes permettant de créer de nouveaux contenus dans la base de données, des modifier des enregistrements déjà existants, et d'en supprimer certains.

1/ Ajout de nouveaux comptes :

Tout d'abord on crée une méthode dans le classe du Model de notre table, ici compte_Model : createCompte() avec une requête sql pour ajouter un enregistrement complet :

```php
// méthode pour ajout erun enregistrement à la table compte
public function createCompte(Compte $compte):bool {
    $sql = "INSERT INTO Compte (Id_compte, Nom, Prenom, Num_telephone, Mail, Mot_de_passe, Id_role) VALUES (:Id_compte, :Nom
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':Id_compte', $compte->getId(), PDO::PARAM_STR);
    $stmt->bindValue(':Nom', $compte->getNom(), PDO::PARAM_STR);
    $stmt->bindValue(':Prenom', $compte->getPrenom(), PDO::PARAM_STR);
    $stmt->bindValue(':Num_telephone', $compte->getNum_telephone(), PDO::PARAM_STR);
    $stmt->bindValue(':Mail', $compte->getMail(), PDO::PARAM_STR);
    $stmt->bindValue(':Mot_de_passe', $compte->getMot_de_passe(), PDO::PARAM_STR);
    $stmt->bindValue(':Id_role', $compte->getId_role()->getId_role(), PDO::PARAM_STR);
    return $stmt->execute();
}
```

Toutes les données vont appeler l'entité de notre table.

Tout d'abord dans notre page accueil_admin.html.twig qui correspond à notre panel admin : là où les administrateurs pourront gérer les comptes du site on crée un formulaire en post permettant de créer de nouveau compte utilisateurs dans la base de données :

```twig
<!--Pannel de gestion et d'administration des comptes des utilisateurs du site -->
<div class="noir_centre">
    <h2 class="or">Gestion des comptes</h2>
</div>
<form action="" method="post">
    <input type="text" id="nom" name="nom" class="form-control"placeholder="Nom" required>
    <input type="text" id="prenom" name="prenom" class="form-control"placeholder="Prenom" required>
    <input type="text" id="num_telephone" name="num_telephone" class="form-control"placeholder="Téléphone" required>
    <input type="text" id="mail" name="mail" class="form-control"placeholder="Adresse mail" required>
    <input type="text" id="mot_de_passe" name="mot_de_passe" class="form-control"placeholder="Mot de passe" required>
    <input type="text" id="id_role" name="id_role" class="form-control"placeholder="Role" required>
    <input type="submit" value="Ajouter" class="btn btn-primary m-2">
</form>
```

Ensuite dans la méthode appelant cette page dans Controller>DefaultController on écrit une PHP en condition permettant de récupérer les informations saisies par le formulaire grâce à la méthode POST et de rediriger ces données vers la méthode createCompte() pour les

stocker dans la base de données :

```php
public function accueil_admin()
{
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $nom = filter_input(INPUT_POST, 'nom', FILTER_SANITIZE_STRING);
        $prenom = filter_input(INPUT_POST, 'prenom', FILTER_SANITIZE_STRING);
        $num_telephone = filter_input(INPUT_POST, 'num_telephone', FILTER_SANITIZE_STRING);
        $mail = filter_input(INPUT_POST, 'mail', FILTER_SANITIZE_STRING);
        $mot_de_passe = filter_input(INPUT_POST, 'mot_de_passe', FILTER_SANITIZE_STRING);
        $id_role = filter_input(INPUT_POST, 'id_role', FILTER_SANITIZE_STRING);

        if (!empty($_POST['nom']) && !empty($_POST['prenom']) && !empty($_POST['num_telephone']) && !empty($_POST['mail']) && !empty($_POST['mot
            // traduction role en identifiant de la table Role
            if ($id_role === 'admin' || $id_role === 'administrateur'){
                $id_role = $this->Role_Model->getOneRole(1);
            } else {
                $id_role = $this->Role_Model->getOneRole(2);
            }
            // on crée le nouvel enregistrement de compte
            $compte = new Compte(null, $nom, $prenom, $num_telephone, $mail, $mot_de_passe, $id_role);
            $success = $this->Compte_Model->createCompte($compte);
            if ($success) {
                header('Location: index.php?page=accueil_admin');
            }
        }
    }
```

Les informations du formulaire s'enregistrent donc désormais dans la table de données « Compte ». On peut ajouter un compte.

2/ Affichages des comptes du site :

Pour voir une liste avec toutes les comptes du site, il faut dans Controller> DefaultController, puis dans la méthode de la page qui affichera la liste : ici c'est accueil_admin.html.twig : on récupérer un tableau avec tous les enregistrements de notre table :

```php
    }
    // on récupère le tableau de tous les comptes du site.
    $users = $this->Compte_Model->getAllCompte();

    echo $this->twig->render('defaultController/accueil_a.html.twig', ['comptes'=>$users]);
}
```

On stocke le tableau de données retourné par getAllCompte() dans $users, et on passe cette variable dans un tableau vers notre page .html.twig.

Dans cette fameuse page de vue on parcourt chaque élément du tableau de données et on affiche les données en les mettant en forme pour qu'ils soient classé par enregistrement :

```
<table>
    <tr> <th>Numéro Compte</th> <th>Nom</th> <th>Prenom</th> <th>Téléphone</th> <th>Mail</th> <th>Mot de passe</th> <th>Role</th></tr>
    {% for compte in comptes %}
    <tr><th>{{compte.id}}</th><th>{{compte.nom}}</th><th>{{compte.prenom}}</th><th>{{compte.num_telephone}}</th><th>{{compte.mail}}</th><th>{{compte
    {% endfor%}
</table>



{% endblock %}
```

3/ Supprimer des données d'une table :

Dans cet exemple nous allons implémenter cette fonctionnalité à la table Compte pour pouvoir supprimer des comptes depuis le panel admin :

Dans srv>Model> Compte_Model : on ajoute la méthode qui demande un id et supprime l'enregistrement associé à cette ID dans la table compte.

```php
// suppprimer un enregistrement d'un compte
public function deleteCompte(int $id): bool {
    $sql = "DELETE Compte, Reservation FROM Compte
            LEFT JOIN Reservation ON Compte.Id_compte = Reservation.Id_compte
            WHERE Compte.Id_compte = :id";
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':id', 1, PDO::PARAM_INT);
    return $stmt->execute();

}
```

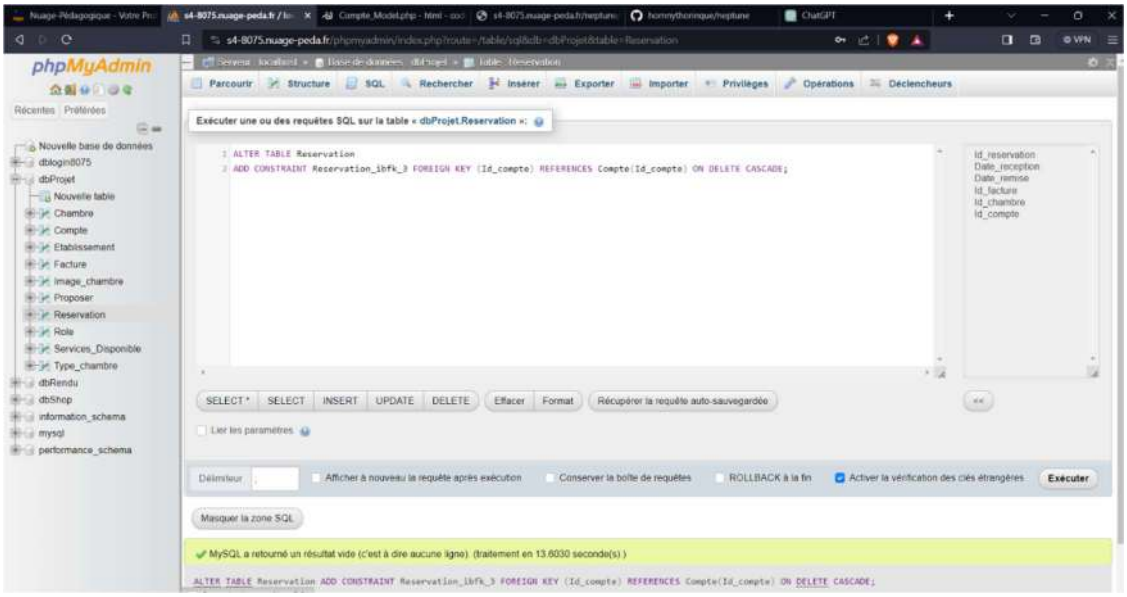
Cependant la table compte est importé dans la table Réservation, nous devons donc d'abord supprimer toutes les réservations du compte avant de supprimer celui-ci. Pour faire cela automatiquement nous allons modifier notre base de données avec un en récréant une clé

étrangère ‘Id_compte’ pour la table Réservation, en précisant DELETE CASCADE afin que l’enregistrement de cette table se supprime automatiquement s’il on supprime le compte associé dans le table compte.

ALTER TABLE Reservation

DROP FOREIGN KEY Reservation_ibfk_3,

ALTER TABLE Reservation ADD CONSTRAINT Reservation_ibfk_3 FOREIGN KEY (Id_compte) REFERENCES Compte(Id_compte) ON DELETE CASCADE;



Ensuite dans Router :

On ajoute un page virtuelle pour récupérer à l’aide la méthode GET l’id de l’enregistrement à supprimer :

```php
$this->pageMappings = [
    'accueil' => [DefaultController::class, 'accueil'],
    'accueil_client' => [DefaultController::class, 'accueil_client'],
    'accueil_admin' => [DefaultController::class, 'accueil_admin'],
    'deleteCompte' => [DefaultController::class, 'deleteCompte'], // page pour supprimer des enregistrements de compte
    'update_compte' => [DefaultController::class, 'update_compte'], // page pour modifier des enregistrements de compte
    'connexion' => [DefaultController::class, 'connexion'],
    'inscription' => [DefaultController::class, 'inscription'],
    'profil' => [DefaultController::class, 'profil'],
    'detailschambre' => [DefaultController::class, 'detailschambre'],
    'detailschambre_client' => [DefaultController::class, 'detailschambre_client'],
    'reservation_1' => [DefaultController::class, 'reservation_1'],
    'reservation_2' => [DefaultController::class, 'reservation_2'],
    '404' => [DefaultController::class, 'error404'],
    '500' => [DefaultController::class, 'error500'],
];
$this->defaultPage = 'accueil';
```

Puis dans DefaultController on ajoute la méthode associée pour récupérer en GET:

```php
// méthode pour supprimer des comptes de la Table de données Compte
public function deleteCompte()
{
    $id = filter_input(INPUT_GET, 'Id_compte', FILTER_SANITIZE_NUMBER_INT);
    $this->compte_Model->deleteCompte(intVal($id));
    header('Location: index.php?page=accueil_admin');

}
```

Désormais il ne reste plus qu'à mettre un bouton «supprimer » sur notre page accueil_admin.html.twig :

```
><th><a href="index.php?page=deleteCompte&Id_compte={{compte.id}}">supprimer</a></th>
```

Nom	Prenom	Téléphone	Adresse mail	Mot de pa

Numéro Compte	Nom	Prenom	Téléphone	Mail	Mot de passe	Role	
3	Gwendal	Ferain	0672254582	gwendal.ferain@ecoles-epsi.net	admin	1	supprimer
11	test	test	0000000000	test@gmail.com	test	1	supprimer
12	Parain	Pierre	0655847595	pierre.parain@gmail.com	motdepasse	1	supprimer
13	Nakache	Didier	1234567891	test@gmail.com	tropfortcemec	2	supprimer
16	huleux	paul	1235478523	paul.huleux@ecoles-epsi.net	coucou	2	supprimer

4 / Pour modifier des enregistrements :

On crée une nouvelle page update_compte.html.twig, qu'on précise dans le routeur :

```php
$this->pageMappings = [
    'accueil' => [DefaultController::class, 'accueil'],
    'accueil_client' => [DefaultController::class, 'accueil_client'],
    'accueil_admin' => [DefaultController::class, 'accueil_admin'],
    'deleteCompte' => [DefaultController::class, 'deleteCompte'], // page pour supprimer des enregistrements de compte
    'update_compte' => [DefaultController::class, 'update_compte'], // page pour modifier des enregistrements de compte
    'connexion' => [DefaultController::class, 'connexion'],
    'inscription' => [DefaultController::class, 'inscription'],
    'profil' => [DefaultController::class, 'profil'],
    'detailschambre' => [DefaultController::class, 'detailschambre'],
    'detailschambre_client' => [DefaultController::class, 'detailschambre_client'],
    'reservation_1' => [DefaultController::class, 'reservation_1'],
    'reservation_2' => [DefaultController::class, 'reservation_2'],
    '404' => [DefaultController::class, 'error404'],
    '500' => [DefaultController::class, 'error500'],
];
$this->defaultPage = 'accueil';
```

Ensuite dans DefaultController on crée un méthode qui permettra de récupérer en GET les données d'un enregistrement sur la page update_compte :

```php
// méthode pour modifier l'enregistrement d'un compte
public function update_compte() {
    $id = filter_input(INPUT_GET, 'Id_compte', FILTER_SANITIZE_NUMBER_INT);
    $compte = $this->compte_Model->getOneCompte(intVal($id));
    echo $this->twig->render('defaultController/update_compte.html.twig', ['compte'=>$compte]);
    }
```

Puis sur cette page on prépare un formulaire en POST pour envoyer les données modifiées :

```twig
</div>

<form method="POST">
    <input type="hidden" value="{{compte.id}}" name="Id_compte" class="form-control">
    <input type="text" value="{{compte.nom}}" name="Nom" placeholder="Nom" class="form-control">
    <input type="text" value="{{compte.prenom}}" name="Prenom" placeholder="Prenom" class="form-control">
    <input type="text" value="{{compte.num_telephone}}" name="Telephone" placeholder="Telephone" class="form-control">
    <input type="text" value="{{compte.mail}}" name="Mail" placeholder="Mail" class="form-control">
    <input type="text" value="{{compte.mot_de_passe}}" name="Mot_de_passe" placeholder="Mot de passe" class="form-control">
    <input type="text" value="{{compte.id_role.getId_role()}}" name="Id_role" placeholder="Role" class="form-control">
    <input type="submit" value="Modifier">
</form>
```

Puis dans Compte_Model on ajoute une méthode updateCompte pour modifier un enregistrement dans une table de données :

```php
public function updateCompte(Compte $compte): bool {
    $sql = "UPDATE Compte
    SET Id_compte = :Id_compte,
        Nom = :Nom,
        Prenom = :Prenom,
        Num_telephone = :Num_telephone,
        Mail = :Mail,
        Mot_de_passe = :Mot_de_passe,
        Id_role = :Id_role
    WHERE Id_compte = :Id_compte";

    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':Id_compte', $compte->getId(), PDO::PARAM_STR);
    $stmt->bindValue(':Nom', $compte->getNom(), PDO::PARAM_STR);
    $stmt->bindValue(':Prenom', $compte->getPrenom(), PDO::PARAM_STR);
    $stmt->bindValue(':Num_telephone', $compte->getNum_telephone(), PDO::PARAM_STR);
    $stmt->bindValue(':Mail', $compte->getMail(), PDO::PARAM_STR);
    $stmt->bindValue(':Mot_de_passe', $compte->getMot_de_passe(), PDO::PARAM_STR);
    $stmt->bindValue(':Id_role', $compte->getId_role()->getId_role(), PDO::PARAM_STR);
    return $stmt->execute();
    }
```

Pour finir dans DefaultController on ajoute du contenu dans la méthode update_compte pour récupérér les informations en POST et modifier l'enregistrement dans la table de données Compte grâce à la méthode updateCompte() :

```php
// méthode pour modifier l'enregistrement d'un compte
public function update_compte() {
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $id = filter_input(INPUT_POST, 'Id_compte', FILTER_SANITIZE_NUMBER_INT);
        $nom = filter_input(INPUT_POST, 'nom', FILTER_SANITIZE_STRING);
        $prenom = filter_input(INPUT_POST, 'prenom', FILTER_SANITIZE_STRING);
        $num_telephone = filter_input(INPUT_POST, 'num_telephone', FILTER_SANITIZE_STRING);
        $mail = filter_input(INPUT_POST, 'mail', FILTER_SANITIZE_STRING);
        $mot_de_passe = filter_input(INPUT_POST, 'mot_de_passe', FILTER_SANITIZE_STRING);
        $id_role = filter_input(INPUT_POST, 'id_role', FILTER_SANITIZE_STRING);
        if (!empty($_POST['nom']) && !empty($_POST['prenom']) && !empty($_POST['num_telephone']) && !empty($_POST['mail']) && !empty($_POST['
            $compte = new Compte(intVal($id), $nom, $prenom, $num_telephone, $mail, $mot_de_passe, $id_role);
            $success = $this->compte_Model->updateCompte($compte);
            if ($success) {
                header('Location: index.php?page=accueil_admin');
            }
        }
    }else{
        $id = filter_input(INPUT_GET, 'Id_compte', FILTER_SANITIZE_NUMBER_INT);
    }
    $compte = $this->compte_Model->getOneType(intVal($id));
    echo $this->twig->render('defaultController/update_compte.html.twig', ['compte'=>$compte]);
}
```

5/ Pour pour faire une recherche d'enregistrement :

Dans compte_Model : on ajoute la méthode getSerachCompte qui va prendre en paramètre le texte recherché par l'utilisateur :

```php
// récupéré des enregistrements en fonction d'une recherche
public function getSearchCompte(?string $search):array{
    $sql = "SELECT Compte.*, Role.* FROM Compte JOIN Role ON Role.Id_role = Compte.Id_role where Nom = :nom OR Prenom = :prenom OR Num_telephon
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(":nom", $search);
    $stmt->bindValue(":prenom", $search);
    $stmt->bindValue(":num_telephone", $search);
    $stmt->bindValue(":mail", $search);
    $stmt->bindValue(":mot_de_passe", $search);
    $stmt->bindValue(":id_role", $search);
    $stmt->execute();
    $compte=[];
    // on organise sous forme d'un tableau les valeurs récupérées
    while($row = $stmt->fetch(PDO::FETCH_ASSOC)){

        $role = null;
        if ($row['Id_role'] !== null) {
            $role = new Role($row['Id_role'], $row['Nom_role']);
        }
        $compte[] = new Compte($row['Id_compte'], $row['Nom'], $row['Prenom'], $row['Num_telephone'], $row['Mail'], $row['Mot_de_passe'], $role
    }
    if($search === null){
        return $this->getAllCompte();
    }else{
        return $compte;
    }
}
```

Si l'utilisateur ne recherche rien on récupère et affiche la liste complète des comptes. Sinon on retourne tous les enregistrements qui contiennent le texte saisi par l'utilisateur.

Pour exécuter cette méthode on remplace la méthode getAllcompte dans la méthode la page accueil_admin dans defaultController par cette méthode en essayant de récupérer une donnée de l'utilisateur en GET :

```php
        }
    }
    // on récupère les comptes
    $liste = filter_input(INPUT_GET, 'search', FILTER_SANITIZE_STRING);
    $comptes = $this->compte_Model->getSearchCompte($liste);
    echo $this->twig->render('defaultController/accueil_a.html.twig', ['comptes'=>$comptes]);
}
```

Enfin sur notre page accueil_admin.html.twig : on met en place un formulaire GET avec la zone de saisie pour rechercher un mot clé :

```html
<form action="index.php" method="get">
    <input type="hidden" name="page" value="accueil_admin">
    <input type="text" id="search" name="search" class="form-control"placeholder="Rechercher">
</form>
```

On peut désormais gérer les comptes facilement :

Il ne reste plus qu'à faire les mêmes étapes pour les chambres.

6/ Création et envoi automatique de Mails et de PDF

Comment créer un fichier pdf personnalisé et envoyer un mail automatiquement en PHP

I/ Création d'un fichier PDF

Dans composer.json :

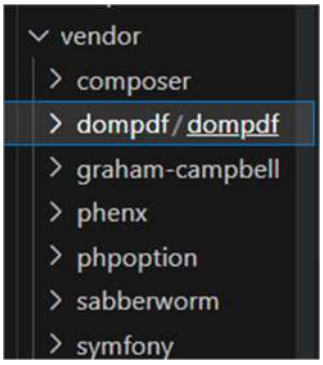
On ajoute la bibliothèque dompdf qui nous permettra de créer des pdf facilement à l'aide de code html, mais également phpmailer qui nous permettra d'envoyer des mails automatiquement :

neptune > {} composer.json > ...

```
{
    "require": {
        "twig/twig": "3.*",
        "vlucas/phpdotenv": "^5.6",
        "dompdf/dompdf": "^2.0",
        "phpmailer/phpmailer" : "^6.9.1"
    },
    "autoload": {
        "psr-4": {
            "MyApp\\": "src/"
        }
    }
}
```

On met à jour composer pour qu'il modifie le fichier composer.lock

```
root@HTTPS-8075:/var/www/html# su login8075
login8075@HTTPS-8075:~$ cd neptune/
login8075@HTTPS-8075:~/neptune$ composer update
```



On peut voir apparaitre désormais la bibliothèque dompdf dans notre projet.

Sur la page réservation_2.html.twig qui permet de finaliser la dernière étape de réservation d'une chambre : le payement, on crée un formulaire pour rentrer les informations bancaires et les transmettre grâce à la méthode POST sécurisée :

```
<!-- formulaire de payement qui enverra automatiquement un pdf par mail à l'utilisateur -->
<form action='' method='post'>
    <input type="hidden" id="send_pdf" name="send_pdf" value="send_pdf">
    <div>
        <label>Nom sur la carte</label>
        <input type="text" id="card_name" name="card_name" class="form-control"placeholder="nom sur la carte" required>
    </div>
    <div>
        <label>Numéro de la carte</label>
        <input type="text" id="num_card" name="num_card" class="form-control"placeholder="numéro de la carte" required>
    </div>
    <div>
        <div>
            <label>Date d'expiration</label>
            <input type="text" id="expiry_date" name="expiry_date" class="form-control"placeholder="MM/AA" required>
        </div>
        <div>
            <label>Code de sécurité</label>
            <input type="text" id="security_code" name="security_code" class="form-control"placeholder="xxx" required>
        </div>
    </div>
    <div>
            <label>Code postale</label>
            <input type="text" id="postal_code" name="postal_code" class="form-control">
    </div>
    <input type="submit" value="Payer et confirmer la réservation">
</form>

{% endblock %}
```

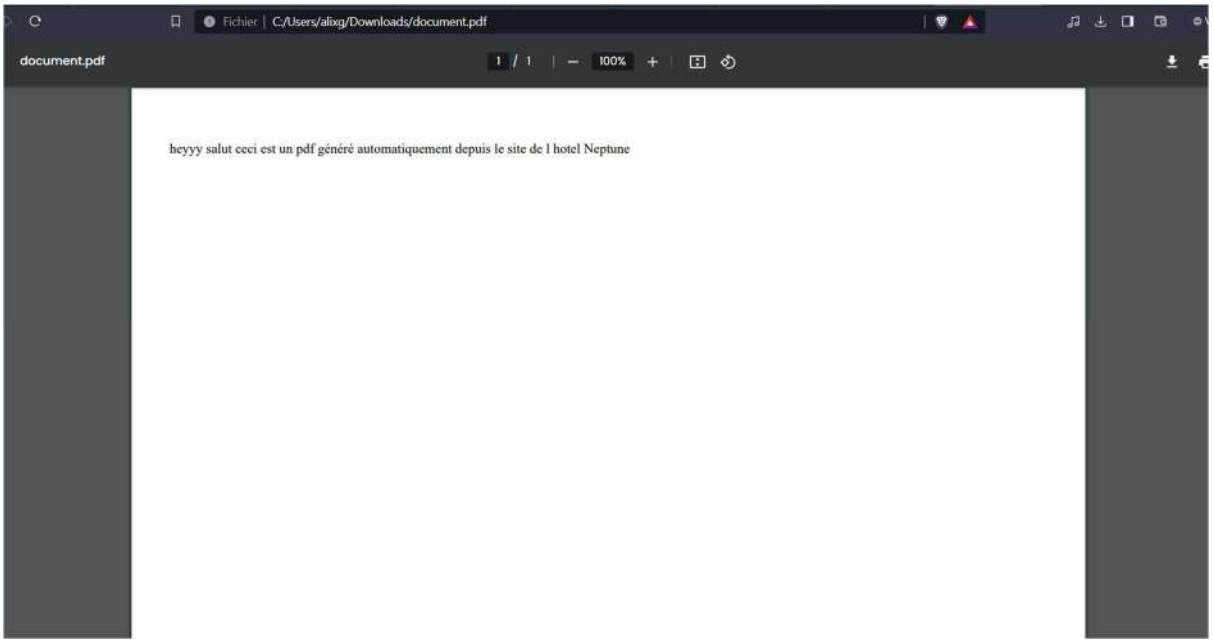
On ajoute dans ce formulaire un champ caché qui passera grâce à la méthode post le mot **« send_pdf »** .

Puis dans Controller>DefaultController, dans la méthode la page réservation_2.html.twig, on intercepte les informations envoyées en POST, et si le message « send_pdf » est récupéré, on crée un formulaire grâce à Dompdf que l'on initialise dans un premier temps avec « require_once » :

```
use MyApp\Entity\Chambre;
use MyApp\Entity\Image_chambre;
use Dompdf\Dompdf;
use PHPMailer\PHPMailer\PHPMailer;
use PHPMailer\PHPMailer\Exception;
```

```
public function reservation_2()
{
    if(isset($_POST['send_pdf'])){
        require_once __DIR__ . '/../../vendor/autoload.php';
        $dompdf = new Dompdf();
        $html = '<p>heyyy salut ceci est un pdf généré automatiquement depuis le site de l hotel Neptune</p>';
        $dompdf->loadHtml($html);
        $dompdf->setPaper('A4', 'landscape');
        $dompdf->render();
        $dompdf->stream();
    }
    echo $this->twig->render('defaultController/reservation_2.html.twig', []);
}
```

Notre pdf est désormais crée et est téléchargeable une fois le formulaire envoyé grâce à la commande `$dompdf->stream();`



2/ Envoyer un mail en avec phpmailer :

Grâce à PHPMailer installé précédemment dans vendor, on peut facilement créer et envoyer des mails automatiquement en se connectant dans un premier temps à un serveur SMTP, dans notre cas il s'agira du server smtp.gmail.com car notre e-mail a été crée par google : arrasneptune@gmail.com :

```php
        $pdf = $dompdf->output(); // on récupère le fichier pdf
        file_put_contents(__DIR__ . '/Details_reservation.pdf', $pdf);

        // on crée un mail
        // on se connecte à notre serveur SMTP
        $mail = new PHPMailer(true);
        $mail->isSMTP();
        $mail->Host = 'smtp.gmail.com';
        $mail->SMTPAuth = true;
        $mail->Username = 'arrasneptune@gmail.com';
        $mail->Password = 'ozey rluw doow gjja';
        $mail->SMTPSecure = PHPMailer::ENCRYPTION_STARTTLS;
        $mail->Port = 587;
        // On défini un expediteur et un recepteur
        $mail->setFrom('arrasneptune@gmail.com', 'Hotel Neptune');
        $mail->addAddress('alix.galant4@gmail.com');
        // on ajoute le pdf
        $mail->addAttachment(__DIR__ . '/Details_reservation.pdf');
        // Contenu
        $mail->isHTML(true);
        $mail->Subject = 'Details de votre reservation de chambre Hotel Neptune';
        $mail->Body = '<p>Votre réservation dans un Hôtel Neptune à Arras a été réglée avec succès.<br> Retrouvez tous les détails de votre rése
        $mail->Send(); // on envoi le mail
        unlink(__DIR__ . '/Details_reservation.pdf');
    }
    echo $this->twig->render('defaultController/reservation_2.html.twig', []);
```

On récupère également le fichier pdf précédemment crée dans la variable $pdf grâce à la commande : $pdf = $dompdf->output() ;

Une fois connecté au serveur SMTP avec les identifiants de nôtre adresse mail du site, on peut définir le sujet, le corps du mail, ainsi que la pièce jointe ici ce sera notre fichier pdf.

Création du calendrier de réservation

1)gestion des dates avec les commandes timestamps

dans le fichier reservation_1.html.twig créer un calendrier se fixant automatiquement sur les dates actuelles, pour ce faire le plus simple en php était d'utiliser les commandes timestamp permettant de se fixer au temps réel grâce au nombre de secondes écoulés depuis le 1er janvier 1970
Ensuite, il utilise des variables Twig pour déterminer le premier jour du mois actuel,le nombre de jours dans le mois, et le jour de la semaine du premier jour, et définie le mois affiché en français.
En utilisant une boucle, le programme génère les en-têtes de colonne du tableau avec les noms des jours de la semaine.

```
{% set premierJourDuMois = now|date_modify('first day of this month') %}
{% set jourActuel = now|date('j') %}
{% set timestampPremierJour = premierJourDuMois|date('U') %}

{# Générer le timestamp pour le mois précédent et le mois suivant #}
{% set timestampMoisPrecedent = premierJourDuMois|date_modify('-1 month')|date('U') %}
{% set timestampMoisSuivant = premierJourDuMois|date_modify('+1 month')|date('U') %}

{# gestion des jours #}
{% set joursDuMois = [] %}
{% set joursDansLeMois = premierJourDuMois|date('t') %}
{% set jourDeLaSemaine = premierJourDuMois|date('N') %}

{# Ajuster les noms des jours #}
{% set joursDeLaSemaine = ['Lun', 'Mar', 'Mer', 'Jeu', 'Ven', 'Sam', 'Dim'] %}
{% set premierJour = jourDeLaSemaine %}

{% set moisEnFrancais = {
    1: 'Janvier',
    2: 'Février',
    3: 'Mars',
    4: 'Avril',
    5: 'Mai',
    6: 'Juin',
    7: 'Juillet',
    8: 'Août',
    9: 'Septembre',
    10: 'Octobre',
    11: 'Novembre',
    12: 'Décembre'
} %}
```

En utilisant une boucle, le programme génère les en-têtes de colonne du tableau avec les noms des Une autre boucle est utilisée pour générer les jours du mois dans le tableau. Chaque jour est affiché avec un lien cliquable pour rediriger vers la page de réservation (index.php?page=reservation_2) si le jour est dans le futur, sinon il est affiché en tant que jour inactif. de la semaine.

```
<tbody>
{% set semaine = [] %}
{% for i in range(1, joursDansLeMois + premierJour) %}
    {% set caseJour %}
        {% if i >= premierJour and i - premierJour + 1 <= joursDansLeMois %}
            {% set jour = i - premierJour + 1 %}
            {% set timestampJour = timestampPremierJour + (jour - 1) * 86400 %}
            {% if timestampJour > now|date('U') %}
                {% if jour == jourActuel %}
                    <a href="index.php?page=reservation_2" class="today">{{ jour }}</a>
                {% else %}
                    <a href="index.php?page=reservation_2" class="active-day">{{ jour }}</a>
                {% endif %}
            {% else %}
                <span class="inactive-day">{{ jour }}</span>
            {% endif %}
        {% else %}
            <span class="inactive-day"></span>
        {% endif %}
    {% endset %}
    {% set semaine = semaine|merge([caseJour]) %}
```

Enfin, des boutons "Mois Précédent" et "Mois Suivant" sont ajoutés en bas du tableau pour permettre à l'utilisateur de naviguer entre les mois.