Feuille de route

“Projet de groupe - Hôtel Neptune”

Projet commencé le 16/01/2025 à remettre le 30/04/2025

Participants :

Tom CLEMENT
Clément SALINGUE
Annalia PRIEUR
Antoine BERTHE

Projet encadré par :

Gregory BOUDRINGHIN

Conc. Expl. BDD - Première Année, 2025

EPSI — Ecole de l’ingénierie informatique, Arras

Contexte

Le projet porte sur la digitalisation des réservations de l'Hôtel Neptune, un établissement 3 étoiles situé à Arras, en développant un site web dynamique en PHP associé à une base de données MySQL. Ce site comprendra deux volets principaux : une interface pour les administrateurs permettant de gérer les chambres, les clients et les réservations, et une interface pour les clients, leur offrant la possibilité de rechercher, réserver et gérer leurs profils. Le but est de remplacer les méthodes manuelles de gestion actuelles par une solution numérique plus rapide, fiable et intuitive.

Problématique

Comment créer un site web qui facilite la gestion des chambres, clients et réservations, tout en garantissant une expérience utilisateur fluide pour les clients ?

. Sous Problèmes

1. Gestion des chambres :
    - Comment organiser efficacement les informations des chambres dans une base de données ?

    - Comment permettre aux administrateurs d'ajouter, modifier et supprimer des photos associées aux chambres ?

    - Comment rendre les informations des chambres accessibles et consultables par les clients selon des critères comme les dates ou le prix ?

2. Gestion des clients :
    - Comment permettre aux clients de créer, modifier ou supprimer leurs profils de manière sécurisée ?
    - Comment organiser les données des clients pour simplifier les recherches et filtrages dans l'interface administrateur ?

3. Gestion des réservations
    - Comment permettre aux clients de réserver une chambre de manière intuitive et sécurisée ?

    - Comment gérer l'annulation des réservations à la fois pour les clients et les administrateurs ?

    - Comment afficher les réservations sous forme de tableau, permettant une visualisation claire et organisée ?

4. Fonctionnalités avancées
    - Comment envoyer automatiquement un e-mail de confirmation après une réservation ?

    - Comment générer des factures au format PDF pour les clients, incluant les détails de leur réservation ?

5. Sécurité et performance
    - Comment garantir la sécurité des données sensibles (informations personnelles et bancaires) des utilisateurs ?

    - Comment optimiser les performances des recherches dans la base de données pour offrir des réponses rapides, même avec un grand nombre d'utilisateurs ?

.

Brainstorming

1. Gestion des chambres

ADMINISTRATION	CLIENT
- Ajouter une chambre (formulaire avec champs de saisie). - Modifier et supprimer des chambres existantes. - Gérer des photos associées à chaque chambre (ajout, modification, suppression). - Rechercher et trier les chambres par critères (disponibilité, type, prix, etc.).	- Visualiser les chambres disponibles sous forme de liste ou de tableau. - Consulter les détails d'une chambre (description, photos, caractéristiques). - Filtrer les chambres selon des critères comme les dates, le prix ou le nombre de lits.

2. Gestion des clients

ADMINISTRATION	CLIENT
- Ajouter, modifier et supprimer des profils de clients. - Rechercher et trier les clients par nom, e-mail ou autres critères pertinents.	- Inscription en ligne (création d'un compte). - Modification et suppression de leur propre profil. - Consultation de leurs informations personnelles et de leurs réservations.

3. Gestion des réservations

ADMINISTRATION	CLIENT
- Visualiser toutes les réservations par chambre ou par date. - Annuler ou modifier une réservation. - Afficher les réservations sous forme de tableau avec filtrage (par date ou chambre).	- Réserver une chambre via un formulaire incluant les dates souhaitées. - Annuler une réservation en ligne. - Simuler le paiement d'une réservation (formulaire avec champs pour carte bancaire factice).

4. Fonctionnalités avancées

- Envoi d'un e-mail de confirmation aux clients après chaque réservation.
- Génération d'une facture au format PDF pour les réservations terminées (avec détails).

5. Sécurité et performance

- Implémentation de requêtes préparées pour sécuriser les accès à la base de données.
- Validation des données saisies par les utilisateurs (formulaires sécurisés).
- Gestion des sessions et authentification des utilisateurs (administrateurs et clients).

Description fonctionnelle

Fonctionnalités principales :

- Gestion des chambres (CRUD + gestion de photos associées).
- Gestion des clients (CRUD).
- Réservations (visualisation, filtrage, annulation).
- Améliorations (e-mail de confirmation, factures PDF).

Solutions proposées

Afin de résoudre les problèmes identifiés, une solution complète sera développée.

- **Qui ?** Le projet sera réalisé en groupe de 4 développeurs.
- **Quoi ?** La création d'un site web permettant de gérer efficacement les réservations.
- **Comment ?** En développant des modules distincts pour chaque fonctionnalité et en adoptant une méthodologie modulaire.
- **Où ?** Le développement se fera sur un environnement local, avec un serveur de test PHP et MySQL.
- **Quand ?** Le projet sera découpé en étapes hebdomadaires (détaillées dans le découpage des tâches).
- **Pourquoi ?** Automatiser les tâches de gestion pour gagner en efficacité et en fiabilité.

Justification des choix techniques

Le choix de PHP et MySQL est dicté par leur simplicité et leur adaptabilité pour développer des sites web dynamiques.

Structure du Site

1. Accueil (Page principale)

- **Contenu** :
    - Présentation de l'hôtel (texte, images).
    - Boutons de navigation vers les sections principales : chambres, réservation, connexion.
    - Informations de contact de l'hôtel (adresse, téléphone, e-mail).

2. Front-office (Espace client)

2.1. Visualisation des chambres

- **Page** : Liste des chambres disponibles.
    - Filtres : prix, dates, nombre de lits, options (balcon, vue).
    - Détails d'une chambre : description, galerie photo, prix par nuit, équipements inclus.

2.2. Réservation d'une chambre

- **Page** : Formulaire de réservation.
    - Champs : sélection de la chambre, dates d'arrivée et de départ, informations personnelles.
    - Vérification de la disponibilité avant validation.
    - Confirmation et simulation de paiement (champs pour carte bancaire).

2.3. Gestion du profil

- **Page** : Espace personnel.
    - Modifier les informations du compte (nom, e-mail, mot de passe, etc.).
    - Supprimer son compte.
    - Visualiser l'historique des réservations.

2.4. Annulation de réservation

- **Page** : Liste des réservations du client.
    - Option pour annuler une réservation.

3. Back-office (Espace administrateur)

3.1. Gestion des chambres

- **Page** : Liste des chambres.
    - Ajouter une nouvelle chambre (formulaire).
    - Modifier les informations d'une chambre existante.
    - Supprimer une chambre.

3.2. Gestion des clients

- **Page** : Liste des clients.
    - Ajouter un nouveau client.
    - Modifier ou supprimer les informations d'un client.

3.3. Gestion des réservations

- **Page** : Liste des réservations.
    - Visualiser les réservations par chambre ou par date.
    - Filtrer les réservations selon des critères.
    - Annuler une réservation pour un client.
    - Afficher les réservations sous forme de tableau par semaine.

4. Pages transversales

- **Connexion** :
    - Page commune pour les administrateurs et clients (choix du rôle après connexion).
- **Contact** :
    - Formulaire de contact pour joindre l'hôtel.
- **Erreur 404** :
    - Page pour les liens invalides ou erreurs de navigation.

Tâches et ordres des tâches:

Pages liées aux chambres

chambres.html.twig → Afficher toutes les chambres
addchambre.html.twig → Ajouter une chambre
updatechambre.html.twig → Modifier une chambre

Pages liées aux clients

profil.html.twig → Afficher et modifier les informations du client
register.html.twig → Page d'inscription
login.html.twig → Page de connexion

Pages liées aux réservations

reservations.html.twig → Liste des réservations du client
reservation_details.html.twig → Voir les détails d'une réservation
addreservation.html.twig → Formulaire de réservation

Pages liées aux paiements

paiement.html.twig → Page pour entrer les infos de paiement
facture.html.twig → Afficher / télécharger la facture

Pages liées aux avis

avis.html.twig → Laisser un avis sur une chambre
liste_avis.html.twig → Voir tous les avis des clients

Autres pages

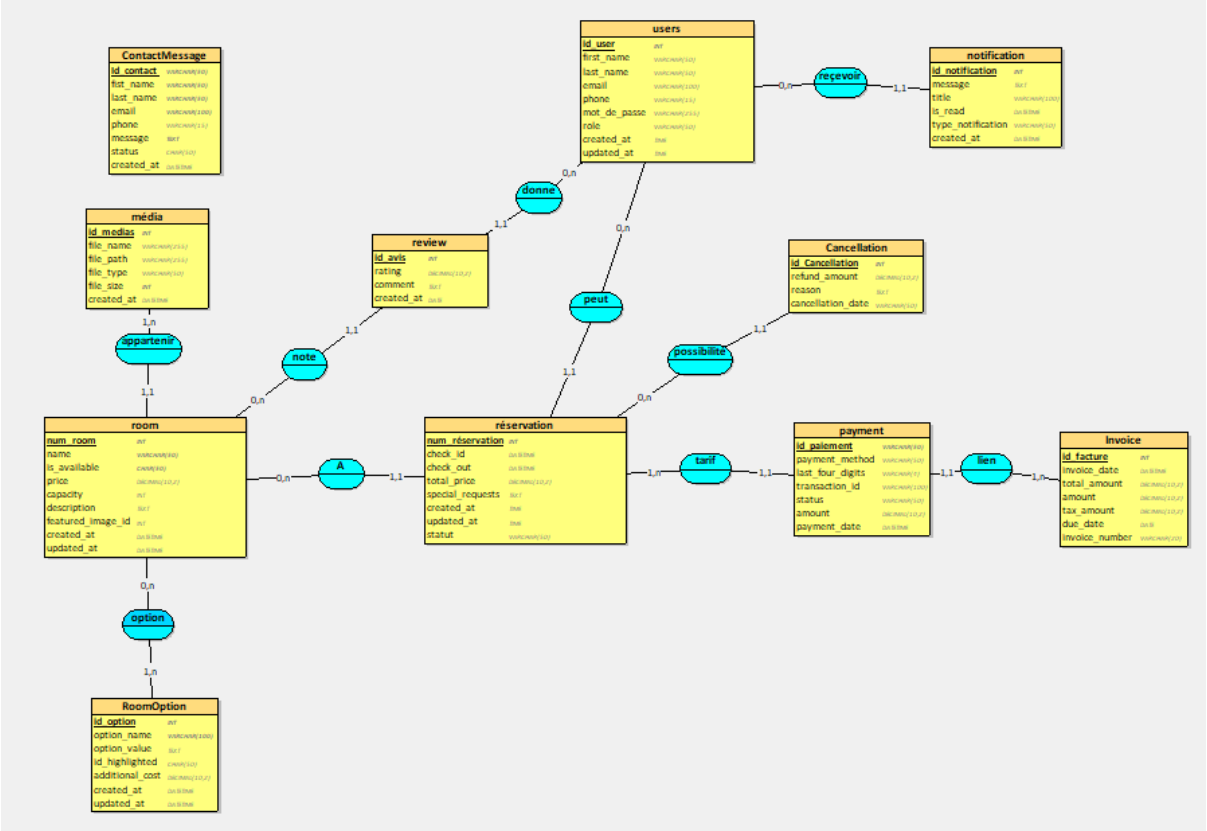
contact.html.twig → Page de contact
administrateur.html.twig → Tableau de bord admin
base.html.twig → Fichier de layout principal

Résumé : Ordre des étapes

1 **Finaliser le CRUD Chambre** (test, affichage, boutons modifier/supprimer)
2 **CRUD Client** (ajout, modification, suppression)
3 **Connexion & Sessions** (Admin vs Client)
4 **CRUD Réservations** (Ajout, annulation, affichage)
5 **Paiement (facultatif)**
6 **Ajout des relations entre entités**

Maquette

MCD:



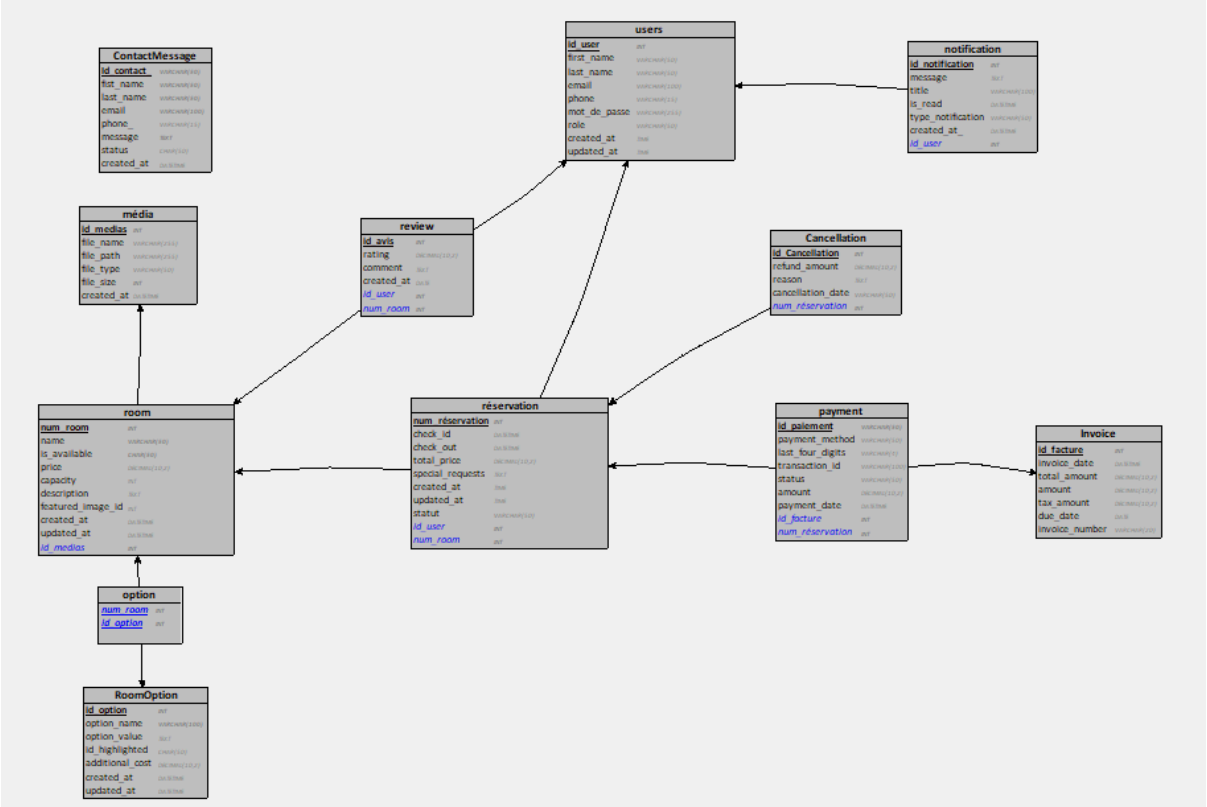
Pour le MCD, l'explication va se dérouler en plusieurs points :

1) l'aspect client : le client a donc la possibilité de laisser un avis, de reçevoir des notifications sur sa réservation mais aussi de pouvoir regarder les différents types de réservation.
2) Lorsque le client à la possibilité de réserver, il faut qu'à travers cette réservation il puisse avoir accès aux différentes chambres disponibles ainsi

que leurs options. De plus, si il peut réserver il doit payer et avoir accès à facture.

3) Chaque chambre est liée aux avis donnés par le client mais aussi les médias soit les images et mis en avant de la chambre.
4) Pour finir il reste une table contactMessage qui permet de stocker les informations du footer du site.

MLD:



On peut donc remarquer que maintenant dans le mld on voit les clés étrangères dans les différents tables qui sont venu s'inclure grâce au différentes relations comme la relation père fils mais aussi la relation maillet avec l'apparition de la table Option comportant la clé primaire de room et RoomOption.

Dictionnaire de données

Cancellation

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
reservation_id	int(11)	Non		
reason	text	Oui	*NULL*	
refund_amount	decimal(10,2)	Oui	*NULL*	
cancellation_date	timestamp	Non	current_timestamp()	
cancelled_by_id	int(11)	Non		

ContactMessage

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
first_name	varchar(50)	Non		
last_name	varchar(50)	Non		
email	varchar(100)	Non		
phone	varchar(15)	Oui	*NULL*	
message	text	Non		
status	enum('new', 'read', 'replied', 'archived')	Oui	new	
created_at	timestamp	Non	current_timestamp()	

Invoice

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
reservation_id	int(11)	Non		
invoice_number	varchar(20)	Non		
amount	decimal(10,2)	Non		
tax_amount	decimal(10,2)	Non		
total_amount	decimal(10,2)	Non		
invoice_date	timestamp	Non	current_timestamp()	
due_date	date	Non		
status	enum('pending', 'paid', 'cancelled', 'refunded')	Oui	pending	
pdf_path	varchar(255)	Oui	*NULL*	

Media

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
file_name	varchar(255)	Non		
file_path	varchar(255)	Non		
file_type	varchar(50)	Non		
file_size	int(11)	Oui	*NULL*	
created_at	timestamp	Non	current_timestamp()	

Notification

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
user_id	int(11)	Non		
title	varchar(100)	Non		
message	text	Non		
is_read	tinyint(1)	Oui	0	
notification_type	enum('reservation', 'payment', 'system', 'other')	Non	system	
created_at	timestamp	Non	current_timestamp()	

Payment

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
invoice_id	int(11)	Non		
amount	decimal(10,2)	Non		
payment_method	enum('credit_card', 'bank_transfer', 'cash', 'paypal')	Non		
transaction_id	varchar(100)	Oui	*NULL*	
status	enum('pending', 'completed', 'failed', 'refunded')	Oui	pending	
payment_date	timestamp	Non	current_timestamp()	
last_four_digits	varchar(4)	Oui	*NULL*	

Reservation

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
user_id	int(11)	Non		
room_id	int(11)	Non		
check_in	datetime	Non		
check_out	datetime	Non		
status	enum('pending', 'confirmed', 'checked_in', 'completed', 'cancelled')	Oui	pending	
total_price	decimal(10,2)	Non		
special_requests	text	Oui	*NULL*	
created_at	timestamp	Non	current_timestamp()	
updated_at	timestamp	Non	current_timestamp()	

Review

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
user_id	int(11)	Non		
room_id	int(11)	Non		
reservation_id	int(11)	Non		
rating	decimal(2,1)	Non		
comment	text	Oui	*NULL*	
created_at	timestamp	Non	current_timestamp()	

Room

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
name	varchar(100)	Non		
is_available	tinyint(1)	Oui	1	
price	decimal(10,2)	Non		
capacity	int(11)	Non		
description	text	Oui	*NULL*	
featured_image_id	int(11)	Oui	*NULL*	
created_at	timestamp	Non	current_timestamp()	
updated_at	timestamp	Non	current_timestamp()	

RoomOption

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
room_id	int(11)	Non		
option_name	varchar(100)	Non		
option_value	text	Oui	*NULL*	
is_highlighted	tinyint(1)	Oui	0	
additional_cost	decimal(10,2)	Oui	0.00	
created_at	timestamp	Non	current_timestamp()	
updated_at	timestamp	Non	current_timestamp()	

User

Colonne	Type	Null	Valeur par défaut	Commentaires
id *(Primaire)*	int(11)	Non		
first_name	varchar(50)	Non		
last_name	varchar(50)	Non		
email	varchar(100)	Non		
phone	varchar(15)	Oui	*NULL*	
password	varchar(255)	Non		
role	enum('client', 'admin', 'staff')	Non	client	
created_at	timestamp	Non	current_timestamp()	
updated_at	timestamp	Non	current_timestamp()	

dépendances fonctionnelles

Cancellation :

id -> #reservation_id, reason, refund_amount, cancellation_date, cancelled_by_id

ContactMessage :

id -> first_name, last_name, email, phone, message, status, created_at

Invoice :

id -> #reservation_id, invoice_number, amount, tax_amount, total_amount, invoice_date, due_date, status, pdf_path

Media :

id -> file_name, file_path, file_type, file_size, created_at

Notification :

id -> #user_id, title, message, is_read, notification_type, created_at

Payement :

id -> #invoice_id, amount, payment_method, transaction_id, status, payment_date, last_four_digits

Reservation :

id -> #user_id, #room_id, check_in, check_out, status, total_price, special_requests, created_at, updated_at

Review :

id -> #user_id, #room_id, #reservation_id, rating, comment, created_at

Room :

id -> #media_id, name, is_available, price, capacity, description, created_at, updated_at

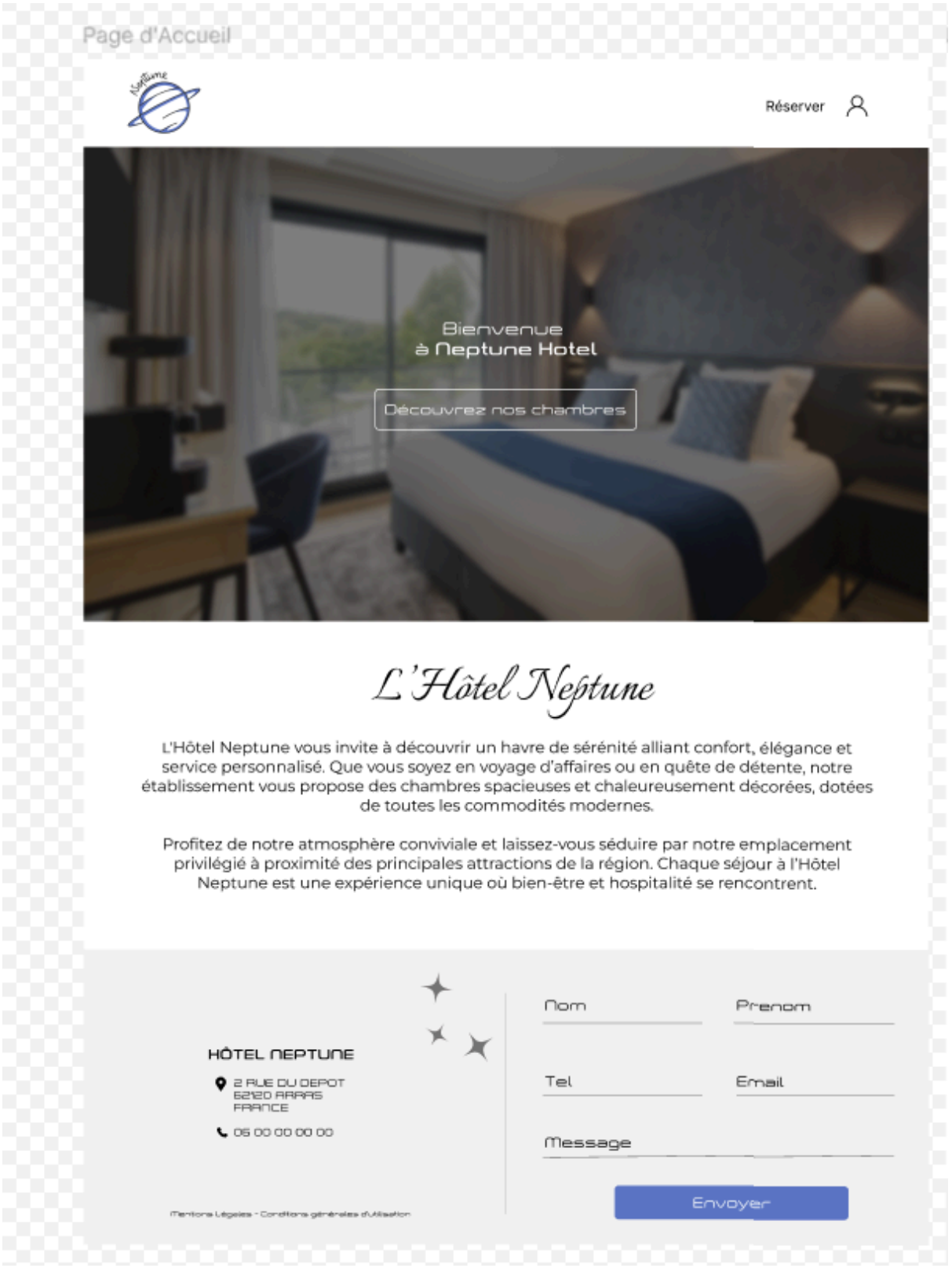
RoomOption :

id -> #room_id, option_name, option_value, is_highlighted, additional_cost, created_at, updated_at

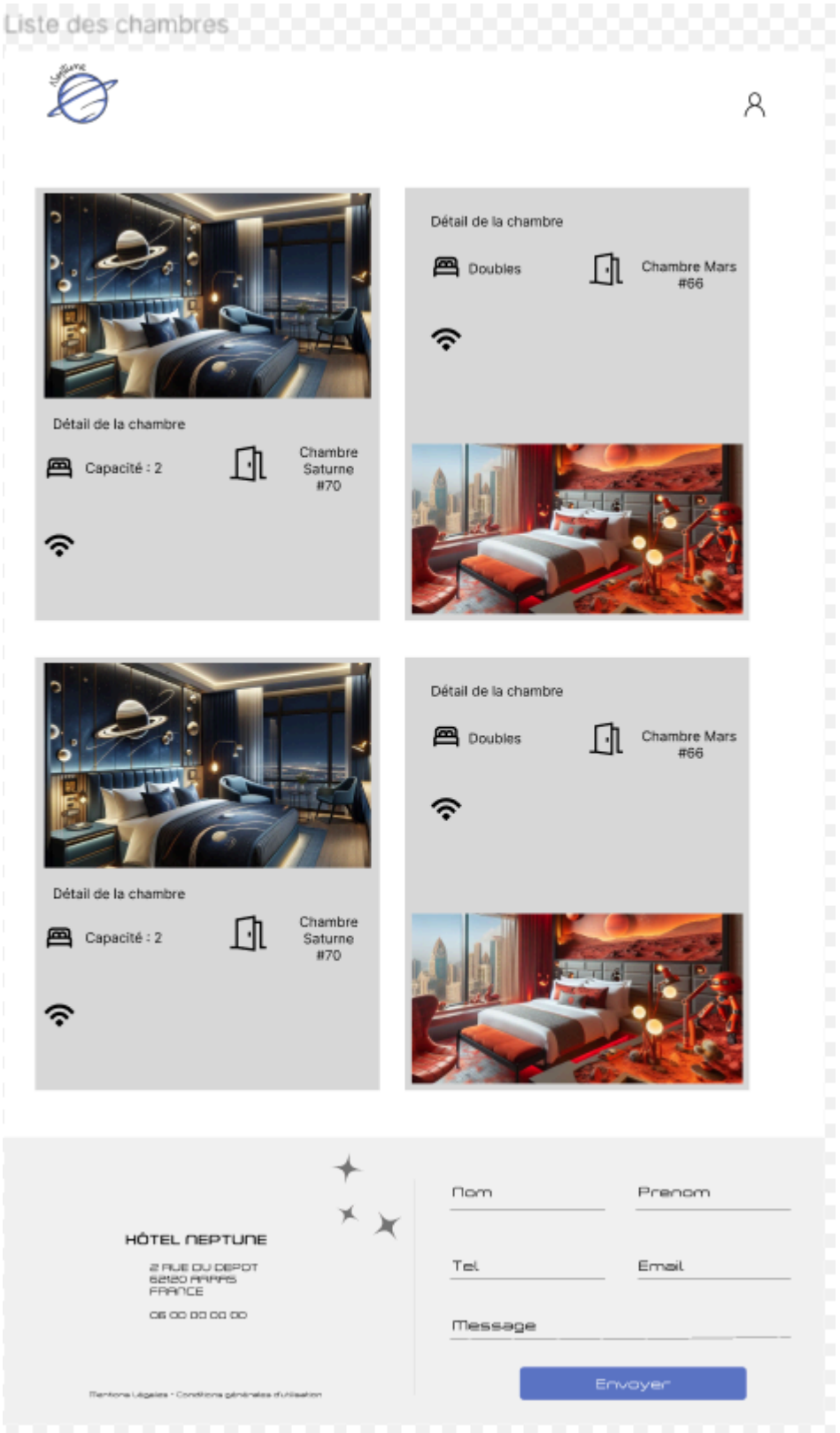
Users :

id -> first_name, last_name, email, phone, password, role, created_at, updated_at

Maquette de la page d’accueil :



Maquette de la liste des chambres :



Maquette formulaire de réservation :

Organisation du groupe

Le projet sera structuré comme suit :

ANNALIA	TOM	CLEMENT	ANTOINE
Réalisation des maquettes. Développement backend (gestion des chambres, clients et réservations). Création du frontend (HTML/CSS) avec formulaires et tableaux interactifs.	Conception de la base de données (MCD, création dans PhpMyAdmin). Développement backend (gestion des chambres, clients et réservations). Création du frontend (HTML/CSS) avec formulaires et tableaux interactifs.	Réalisation des maquettes. Conception de la base de données (MCD, création dans PhpMyAdmin). Développement backend (gestion des chambres, clients et réservations). Création du frontend (HTML/CSS) avec formulaires et	Conception de la base de données (MCD, création dans PhpMyAdmin). Développement backend (gestion des chambres, clients et réservations). Création du frontend (HTML/CSS) avec formulaires et tableaux interactifs.

		tableaux interactifs.	

Technique utilisé :

Le projet repose sur une architecture MVC(Modele-vue-controleur) claire et organisé:

Le Routeur (index.php) :

c'est lui qui regarde ce que l'utilisateur veut faire en fonction de (l'url) et envoie la demande au bon contrôleur

```
private $dependencyContainer;
private $pageMappings;
private $defaultPage;
private $errorPage;
private $unauthorizedPage;
```

Cette classe a plusieurs propriétés privées :

- $dependencyContainer : pour instancier les contrôleurs ou services automatiquement

- $pageMappings : une liste de toutes les routes du site (clé = nom de la page ; valeur = [contrôleur, méthode, rôle requis])

- $defaultPage, $errorPage, $unauthorizedPage : les pages de secours (ex : 404, erreur, page par défaut…)

```php
$this->pageMappings = [
    // Routes par défaut - accessibles à tous
    'home' => [DefaultController::class, 'home', null],
    '404' => [DefaultController::class, 'error404', null],
    '500' => [DefaultController::class, 'error500', null],
    '403' => [DefaultController::class, 'error403', null],

    // Routes utilisateurs avec permissions
    'users' => [UserController::class, 'listUsers', 'admin'],
    'listUsers' => [UserController::class, 'listUsers', 'admin'],
    'showUser' => [UserController::class, 'showUser', 'admin'],
    'addUser' => [UserController::class, 'addUser', 'admin'],
    'updateUser' => [UserController::class, 'updateUser', 'admin'],
    'deleteUser' => [UserController::class, 'deleteUser', 'admin'],
    'changePassword' => [UserController::class, 'changePassword', 'client'],
    'profile' => [UserController::class, 'profile', 'client'],
    'login' => [UserController::class, 'login', null],
    'register' => [UserController::class, 'register', null],
    'logout' => [UserController::class, 'logout', null],
];
$this->defaultPage = 'home';
$this->errorPage = '404';
$this->unauthorizedPage = '403';
```

On retourve toutes les pages du site qui sont déclarées.

- Clé : ce qu'on met dans l'URL ?page=home
- Valeur :

DefaultController::class : le contrôleur à utiliser

'home' : la méthode dans ce contrôleur à exécuter

null ou 'admin' : le rôle requis pour accéder à cette page
Par contre si la page n'existe pas il affiche une erreur 404,et si l'utilisateur n'a pas le droit d'acces alors ca affiche une erreur 403

```php
    private function getUserRole(): ?string
    {

        if (isset($_SESSION['user_role'])) {
            return $_SESSION['user_role'];
        }

        return null;
    }

    private function hasPermission(?string $userRole, string $requiredRole): bool
    {
        if ($userRole === null) {
            return false;
        }

        $roleHierarchy = [
            'client' => 1,
            'staff' => 2,
            'admin' => 3
        ];

        if (!isset($roleHierarchy[$userRole])) {
            return false;
        }

        if (!isset($roleHierarchy[$requiredRole])) {
            return false;
        }

        return $roleHierarchy[$userRole] >= $roleHierarchy[$requiredRole];
    }
}
```

Cette fonction permet de voir dans la session dans la session PHP si l'utilisateur est connecté, et retourne son rôle (ex : "admin", "client"...).

- Si $_SESSION['user_role'] existe → on le retourne (ex : 'admin')
- Sinon → l'utilisateur n'est pas connecté, donc on retourne null

Plus le chiffre est grand, plus le rôle est "haut placé".
Donc :

- Un admin peut accéder à toutes les pages (3 ≥ tout)
- Un staff peut accéder aux pages client (2 ≥ 1), mais pas admin
- Un client ne peut accéder qu'aux pages client

Les controleurs :

Il traite la demande ,

- Reçoit la requête de l’utilisateur
- Appellent les modèles pour récupérer ou modifier les données
- Utilisent les entités pour structurer ces données
- Transmettent ensuite les résultats à une vue Twig

dans le projet nous avons différent Controleur

-CancellationController.php

-ContractMessageController.php

-DefaultController.php

-InvoiceController.php

-NotificationController.php

-PaymentController.php

-ReservationController.php

-ReviewController.php

-RoomController.php

-RoomOption.php

-UserController.php

exemple de Controller InvoicController.php qui permet d’afficher les factures

```php
use MyApp\Model\InvoiceModel;
use MyApp\Model\ReservationModel;
use MyApp\Entity\Invoice;
use MyApp\Entity\Reservation;
use MyApp\Service\DependencyContainer;
use Twig\Environment;
```

On utilise :

- Le modèle InvoiceModel pour gérer les factures en base de données
- Le modèle ReservationModel pour lier la facture à une réservation
- L'entité Invoice pour créer des objets facture
- Twig pour afficher les vues (pages HTML dynamiques)

```php
public function listInvoices()
{
    $invoices = $this->invoiceModel->getAllInvoices();
    echo $this->twig->render('invoiceController/listInvoices.html.twig', [
        'invoices' => $invoices
    ]);
}
```

- Récupère toutes les factures en base de données
- Affiche la page listInvoices.html.twig avec les données

Cette méthode sert à voir toutes les factures existantes.

```php
public function addInvoice()
{
    $reservations = $this->reservationModel->getAllReservations();

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $reservation_id = filter_input(INPUT_POST, 'reservation_id', FILTER_SANITIZE_NUMBER_INT);
        $invoice_number = filter_input(INPUT_POST, 'invoice_number', FILTER_SANITIZE_STRING);
        $amount = filter_input(INPUT_POST, 'amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $tax_amount = filter_input(INPUT_POST, 'tax_amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $total_amount = filter_input(INPUT_POST, 'total_amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $due_date = $_POST['due_date'] ?? '';
        $status = $_POST['status'] ?? 'pending';
        $pdf_path = filter_input(INPUT_POST, 'pdf_path', FILTER_SANITIZE_STRING);

        if ($reservation_id && $invoice_number && $amount && $tax_amount && $total_amount && $due_date) {
            $reservation = $this->reservationModel->getOneReservation((int)$reservation_id);
            if ($reservation) {
                $invoice = new Invoice(null, $reservation, $invoice_number, $amount, $tax_amount, $total_amount, '', $due_date, $status, $pdf_path);
                $this->invoiceModel->createInvoice($invoice);
                $_SESSION['message'] = 'Facture ajoutée';
                header('Location: index.php?page=list-invoices');
                exit();
            }
        }

        $_SESSION['message'] = 'Erreur lors de la création de la facture';
    }

    echo $this->twig->render('invoiceController/addInvoice.html.twig', [
        'reservations' => $reservations
    ]);
}
```

Vérification du formulaire

1)Si le formulaire a été soumis, on récupère les données envoyées (sécurisées avec filter_input et FILTER_SANITIZE).

2)Données récupérées :

- reservation_id → réservation liée à la facture
- invoice_number → numéro de facture
- amount, tax_amount, total_amount → montants
- due_date → date d'échéance
- status → statut (ex: "en attente", "payée")
- pdf_path → lien vers la facture PDF

3)Ensuite , On crée un objet `Invoice`

On l'enregistre en base de données grâce à `createInvoice()`

4)On a un message de confirmation ou d'erreur.
5) affichage de la page du formulaire

```php
public function updateInvoice()
{
    $reservations = $this->reservationModel->getAllReservations();

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $id = filter_input(INPUT_POST, 'id', FILTER_SANITIZE_NUMBER_INT);
        $reservation_id = filter_input(INPUT_POST, 'reservation_id', FILTER_SANITIZE_NUMBER_INT);
        $invoice_number = filter_input(INPUT_POST, 'invoice_number', FILTER_SANITIZE_STRING);
        $amount = filter_input(INPUT_POST, 'amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $tax_amount = filter_input(INPUT_POST, 'tax_amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $total_amount = filter_input(INPUT_POST, 'total_amount', FILTER_SANITIZE_NUMBER_FLOAT, FILTER_FLAG_ALLOW_FRACTION);
        $due_date = $_POST['due_date'] ?? '';
        $status = $_POST['status'] ?? 'pending';
        $pdf_path = filter_input(INPUT_POST, 'pdf_path', FILTER_SANITIZE_STRING);

        $reservation = $this->reservationModel->getOneReservation((int)$reservation_id);
        $invoice = $this->invoiceModel->getOneInvoice((int)$id);

        if ($reservation && $invoice) {
            $updatedInvoice = new Invoice((int)$id, $reservation, $invoice_number, $amount, $tax_amount, $total_amount, $invoice->getInvoiceDate(), $due_date, $status, $pdf_path);
            $this->invoiceModel->updateInvoice($updatedInvoice);
            $_SESSION['message'] = 'Facture mise à jour';
            header('Location: index.php?page=list-invoices');
            exit();
        } else {
            $_SESSION['message'] = 'Erreur sur la mise à jour';
        }
    } else {
        $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
        $invoice = $this->invoiceModel->getOneInvoice((int)$id);

        if (!$invoice) {
            $_SESSION['message'] = 'Facture introuvable';
            header('Location: index.php?page=list-invoices');
            exit();
        }

        echo $this->twig->render('invoiceController/updateInvoice.html.twig', [
            'invoice' => $invoice,
            'reservations' => $reservations
        ]);
    }
```

Cette methode permet la modification d’une facture existante

1) Le formulaire a été soumis (requête POST)

   Les données sont récupérées :

- id → ID de la facture à modifier
- reservation_id → ID de la réservation associée
- invoice_number, amount, tax_amount, total_amount, due_date, status, pdf_path → les champs du formulaire

2) On vérifie si la réservation existe avec getOneReservation()
   La facture à modifier existe avec getOneInvoice()

3) On crée une nouvelle instance d’Invoice avec les nouvelles données :

On conserve

- L’ancienne date de création de la facture avec $invoice->getInvoiceDate() (on ne la modifie pas)
- Tous les autres champs sont mis à jour

4) On sauvegarde la mise à jour et on redirige avec un message de succes.

5) Si l'utilisateur arrive sur la page sans encore avoir validé le formulaire
    - On récupère l'ID de la facture dans l'URL (?id=3)
    - On récupère l'objet facture correspondant en base
    - Si la facture n'existe pas → on redirige vers la liste avec un message

Sinon, on affiche le formulaire de modification

On envoie à la vue :

- La facture à modifier (invoice)
- Toutes les réservations (pour que l'utilisateur puisse en choisir une)

```php
public function showInvoice()
{
    $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
    $invoice = $this->invoiceModel->getOneInvoice((int)$id);

    if (!$invoice) {
        $_SESSION['message'] = 'Facture introuvable';
        header('Location: index.php?page=list-invoices');
        exit();
    }

    echo $this->twig->render('invoiceController/showInvoice.html.twig', [
        'invoice' => $invoice
    ]);
}
```

Afficher une seule facture, avec toutes ses informations (montants, date, réservation associée, statut, etc.)

1) On récupère l'id de la facture dans l'URL
2) On demande au modèle InvoiceModel de retourner la facture avec l'ID reçu. Si elle n'existe pas, $invoice vaudra false ou null.
3) Si la facture n'existe pas :
- On affiche un message d'erreur
- On redirige vers la liste des factures
4) Si la facture existe :
- On affiche la page Twig showInvoice.html.twig

- On lui envoie la variable invoice pour qu'elle affiche toutes les infos

```php
    public function deleteInvoice()
    {
        $id = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT);
        if ($id) {
            $this->invoiceModel->deleteInvoice((int)$id);
        }
        header('Location: index.php?page=list-invoices');
    }
}
```

Permettre à un utilisateur autorisé de supprimer une facture de la base de donnée

1) On récupère l'ID depuis l'URL :

On sécurise l'entrée grâce à FILTER_SANITIZE_NUMBER_INT pour éviter toute tentative d'injection ou de manipulation.

2) Si un ID valide est trouvé, on demande au modèle InvoiceModel de supprimer la facture en base de données.
3) Une fois la suppression effectuée , on redirige l'utilisateur vers la liste des factures.

Les Models :

Ils contiennent la logique d'accès à la base de données : requêtes SQL, insertion, modification, suppression, lecture.

Chaque modèle est lié à une entité du même nom.

Notre projets contients différents models :

-CancellationModel.php

-ContactMessage.php

-InvoiceModel.php

-MediaModel.php

-NotificationModel.php

-PaymentModel.php

-ReservationModel.php

-ReviewModel.php

-RoomModel.php

-RoomOptionModel.php

-UserModel.php

exemple de Model pour InvoiceModel.php

```php
class InvoiceModel
{
    private PDO $db;

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

    public function getAllInvoices(): array
    {
        $sql = "SELECT i.*,
                       r.id as reservation_id, r.user_id, r.room_id, r.check_in, r.check_out, r.status as reservation_status, r.total_price, r.special_requests,
                FROM Invoice i
                INNER JOIN Reservation r ON i.reservation_id = r.id
                ORDER BY i.invoice_date DESC";
        $stmt = $this->db->query($sql);
        $invoices = [];

        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $reservation = new Reservation(
                $row['reservation_id'],
                new User($row['user_id'], '', '', '', null, '', '', '', ''),
                new Room($row['room_id'], '', true, 0, 0, '', null, '', ''),
                $row['check_in'], $row['check_out'], $row['reservation_status'],
                $row['total_price'], $row['special_requests'], $row['reservation_created'], $row['reservation_updated']
            );

            $invoices[] = new Invoice(
                $row['id'],
                $reservation,
                $row['invoice_number'],
                $row['amount'],
                $row['tax_amount'],
                $row['total_amount'],
                $row['invoice_date'],
                $row['due_date'],
                $row['status'],
                $row['pdf_path']
            );
        }
```

1) Connexion à la base via PDO

   Le modèle reçoit une connexion PDO à la base de données via le constructeur.

2) La méthode getAllInvoices() retourne un tableau d'objets invoice
3) Requête SQL : jointure entre Invoice et Reservation

- On sélectionne toutes les colonnes de la facture (i.*)
- On fait une jointure avec la table Reservation pour avoir toutes les infos liées à la réservation de la facture
- On trie les factures par date (ORDER BY invoice_date DESC)

4) On exécute la requête SQL

On prépare un tableau $invoices pour stocker les objets

5) On construit une réservation complète pour chaque facture :

   En créant des objets User et Room liés à cette réservation

   Ensuite, on crée un objet Invoice complet en lui passant tous les champs

6) On retourne un tableau contenant **toutes les factures du site**, prêtes à être utilisées dans un contrôleur ou une vue.

```php
public function getOneInvoice(int $id): ?Invoice
{
    $sql = "SELECT i.*,
                   r.id as reservation_id, r.user_id, r.room_id, r.check_in, r.check_out, r.status as reservation_status, r.total_price, r.special_requests,
            FROM Invoice i
            INNER JOIN Reservation r ON i.reservation_id = r.id
            WHERE i.id = :id";
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':id', $id, PDO::PARAM_INT);
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    if (!$row) return null;

    $reservation = new Reservation(
        $row['reservation_id'],
        new User($row['user_id'], '', '', '', null, '', '', '', ''),
        new Room($row['room_id'], '', true, 0, 0, '', null, '', ''),
        $row['check_in'], $row['check_out'], $row['reservation_status'],
        $row['total_price'], $row['special_requests'], $row['reservation_created'], $row['reservation_updated']
    );

    return new Invoice(
        $row['id'],
        $reservation,
        $row['invoice_number'],
        $row['amount'],
        $row['tax_amount'],
        $row['total_amount'],
        $row['invoice_date'],
        $row['due_date'],
        $row['status'],
        $row['pdf_path']
    );
}
```

Cette méthode permet de récupérer une facture précise (grâce à son id) et de la convertir en un objet Invoice complet avec :

- Ses propres infos (montant, statut, date, etc.)
- Et les infos de la réservation liée

1) On sélectionne toutes les colonnes de la facture (i.*)

   Et on fait une jointure avec la table Reservation pour obtenir :

   - Le client (user_id)
   - La chambre (room_id)
   - Les dates, le statut, le prix, etc.

   On filtre avec WHERE i.id = :id → on veut une seule facture

2) On prépare la requête (prévention SQL injection)

   On lie l'ID avec sécurité

   On exécute la requête

3) Si aucune facture ne correspond → on retourne null

   Sinon → on a toutes les données nécessaires dans $row

4)

On crée un objet Reservation avec :

- Son ID, ses dates, son statut, son prix, etc.
- Un objet User avec juste l'id
- Un objet Room de même, avec juste l'id

5) On retourne un objet Invoice complet avec :

- L'objet Reservation construit juste avant
- Le numéro, les montants, les dates, le statut, le PDF…

```php
public function createInvoice(Invoice $invoice): bool
{
    $sql = "INSERT INTO Invoice (reservation_id, invoice_number, amount, tax_amount, total_amount, due_date, status, pdf_path)
            VALUES (:reservation_id, :invoice_number, :amount, :tax_amount, :total_amount, :due_date, :status, :pdf_path)";
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':reservation_id', $invoice->getReservation()->getId(), PDO::PARAM_INT);
    $stmt->bindValue(':invoice_number', $invoice->getInvoiceNumber(), PDO::PARAM_STR);
    $stmt->bindValue(':amount', $invoice->getAmount());
    $stmt->bindValue(':tax_amount', $invoice->getTaxAmount());
    $stmt->bindValue(':total_amount', $invoice->getTotalAmount());
    $stmt->bindValue(':due_date', $invoice->getDueDate());
    $stmt->bindValue(':status', $invoice->getStatus(), PDO::PARAM_STR);
    $stmt->bindValue(':pdf_path', $invoice->getPdfPath(), PDO::PARAM_STR);
    return $stmt->execute();
}
```

Cette méthode permet permet d'ajouter une nouvelle facture à la base de données à partir d'un objet Invoice.

1) Requete SQL qui va insérer une nouvelle facture dans la table Invoice.

   Elle contient tous les champs importants d'une facture.

2) On utilise prepare() pour sécuriser la requête et éviter les injections SQL.
3) Pour chaque champ, on récupère la valeur depuis l'objet Invoice.
4) Si l'insertion réussit → retourne true

   Sinon → retourne false

```php
public function updateInvoice(Invoice $invoice): bool
{
    $sql = "UPDATE Invoice SET reservation_id = :reservation_id, invoice_number = :invoice_number, amount = :amount,
            tax_amount = :tax_amount, total_amount = :total_amount, due_date = :due_date, status = :status, pdf_path = :pdf_path
            WHERE id = :id";
    $stmt = $this->db->prepare($sql);
    $stmt->bindValue(':reservation_id', $invoice->getReservation()->getId(), PDO::PARAM_INT);
    $stmt->bindValue(':invoice_number', $invoice->getInvoiceNumber(), PDO::PARAM_STR);
    $stmt->bindValue(':amount', $invoice->getAmount());
    $stmt->bindValue(':tax_amount', $invoice->getTaxAmount());
    $stmt->bindValue(':total_amount', $invoice->getTotalAmount());
    $stmt->bindValue(':due_date', $invoice->getDueDate());
    $stmt->bindValue(':status', $invoice->getStatus(), PDO::PARAM_STR);
    $stmt->bindValue(':pdf_path', $invoice->getPdfPath(), PDO::PARAM_STR);
    $stmt->bindValue(':id', $invoice->getId(), PDO::PARAM_INT);
    return $stmt->execute();
}
```

1) Requete SQL update ,Cette requête met à jour tous les champs d'une facture précise identifiée par son id.
2) On prépare la requête SQL via PDO (sécurisé contre les injections SQL).
3) Le getId() est obligatoire pour identifier quelle facture doit être modifiée .
4) true si la mise à jour a réussi
   false sinon

```php
    public function deleteInvoice(int $id): bool
    {
        $sql = "DELETE FROM Invoice WHERE id = :id";
        $stmt = $this->db->prepare($sql);
        $stmt->bindValue(':id', $id, PDO::PARAM_INT);
        return $stmt->execute();
    }
}
```

Supprimer une facture précise de la base de données, identifiée par son id.

1) requete SQl qui va supprimer la facture dont l'ID correspond à la valeur passé en paramètre.Elle utilise le paramètre :id pour sécuriser la requete
2) On utilise PDO pour préparer la requête SQL (ce qui évite les injections SQL).
3) On associe l'ID passé en paramètre à la variable :id dans la requête SQL.

- PDO::PARAM_INT précise que c'est un entier (sécurité + clarté).

4) Si elle réussit → retourne true

Si elle échoue → retourne false

Les entités :

Les entités représentent les objets métiers du projet. Chaque entité correspond à une table dans la base de données.

Dans notre projet nous avons comme entités :

-Cancellation.php

-ContactMessage.php

-Invoice.php

-Medial.php

-Notification.php

-Payment.php

-Reservation.php

-Review.php

-Room.php

-RoomOption.php

```php
class Invoice
{
    private ?int $id;
    private Reservation $reservation;
    private string $invoice_number;
    private float $amount;
    private float $tax_amount;
    private float $total_amount;
    private string $invoice_date;
    private string $due_date;
    private string $status;
    private ?string $pdf_path;

    public function __construct(
        ?int $id,
        Reservation $reservation,
        string $invoice_number,
        float $amount,
        float $tax_amount,
        float $total_amount,
        string $invoice_date,
        string $due_date,
        string $status,
        ?string $pdf_path
    ) {
        $this->id = $id;
        $this->reservation = $reservation;
        $this->invoice_number = $invoice_number;
        $this->amount = $amount;
        $this->tax_amount = $tax_amount;
        $this->total_amount = $total_amount;
        $this->invoice_date = $invoice_date;
        $this->due_date = $due_date;
        $this->status = $status;
        $this->pdf_path = $pdf_path;
    }
```

1) Le constructeur Quand on crée une facture avec new Invoice(...), toutes les données nécessaires sont envoyées à l'objet.
   Cela garantit que l'objet est complet dès sa création.

```php
public function getId(): ?int { return $this->id; }
public function setId(?int $id): void { $this->id = $id; }

public function getReservation(): Reservation { return $this->reservation; }
public function setReservation(Reservation $reservation): void { $this->reservation = $reservation; }

public function getInvoiceNumber(): string { return $this->invoice_number; }
public function setInvoiceNumber(string $invoice_number): void { $this->invoice_number = $invoice_number; }

public function getAmount(): float { return $this->amount; }
public function setAmount(float $amount): void { $this->amount = $amount; }
```

Chaque champ de l'entité a :

- un getter (ex: getAmount()) → pour lire la valeur
- un setter (ex: setAmount()) → pour modifier la valeur
  Cela permet de bien contrôler l'accès aux données.

Le Twig :

Les vues sont des fichiers Twig qui affichent les données à l'utilisateur. Elles sont appelées par les contrôleurs après traitement.

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

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

{% block css %} {{ parent() }} <link rel='stylesheet' href='css/user/auth.css'> {% endblock %}

{% block content %}
<a href="index.php" class="home-link">Accueil</a>
<div class="page-container">
    <div class="login-form">
        <div class="textelogin">
            Inscription
        </div>
```

Cela veut dire que cette page hérite d'un template de base (base.html.twig) qui contient :

- le <head>
  les balises HTML principales
- la structure commune à toutes les pages

Bloc title :

- Ce bloc complète le titre de la page
- {{ parent() }} garde le titre défini dans base.html.twig (ex. "Neptune Hostel |") et ajoute "Inscription" à la suite

bloc content :

C'est ici que se trouve le contenu principal de la page. Tout ce qui est dans ce bloc remplacera le bloc content de base.html.twig.

```html
<form method="post" action="index.php?page=register">
    <label class="form-label">Nom</label>
    <input class="form-input" type="text" placeholder="Saisir votre nom" name="last_name" required/>

    <label class="form-label">Prénom</label>
    <input class="form-input" type="text" placeholder="Saisir votre prénom" name="first_name" required/>

    <label class="form-label">Numéro de téléphone</label>
    <input class="form-input" type="text" placeholder="Saisir votre numéro de téléphone" name="phone"/>

    <label class="form-label">Email</label>
    <input class="form-input" type="email" placeholder="Saisir votre email" name="email" required/>

    <label class="form-label">Mot de passe</label>
    <input class="form-input" type="password" placeholder="Saisir votre mot de passe" name="password" required/>

    <label class="form-label">Confirmation de mot de passe</label>
    <input class="form-input" type="password" placeholder="Confirmez votre mot de passe" name="confirm_password" requi

    <button type="submit" class="submit-button">S'inscrire</button>
</form>
```

Le formulaire utilise la méthode POST

Il est traité par la route index.php?page=register il est utilisé par le controleur UserController::register().

Methode Login dans UserController.php

```php
public function login()
{
    if (session_status() === PHP_SESSION_NONE) {
        session_start();
    }

    if (isset($_SESSION['user_id'])) {
        header('Location: index.php?page=profile');
        exit();
    }

    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
        $password = $_POST['password'] ?? '';

        if (!empty($email) && !empty($password)) {
            $user = $this->userModel->getUserByEmail($email);

            if ($user !== null && password_verify($password, $user->getPassword())) {
                $_SESSION['user_id'] = $user->getId();
                $_SESSION['user_name'] = $user->getFirstName() . ' ' . $user->getLastName();
                $_SESSION['user_role'] = $user->getRole(); // Utilisation de getRole()
                $_SESSION['message'] = 'Bienvenue, ' . $user->getFirstName() . ' !';
                $_SESSION['success'] = true;
```

1) Démarrer la session.

Vérifie si une session PHP est active. Si non, il en démarre une.
Obligatoire pour utiliser $_SESSION.

2) Si l'utilisateur est déjà connecté, on le redirige

Si l'utilisateur est déjà connecté (il a un `user_id` en session), inutile de lui montrer le formulaire : on le redirige vers sa page de profil.

3) Si le formulaire a été soumis (méthode POST)

On vérifie si l'utilisateur vient de valider le formulaire de connexion

4) On récupère l'email et on le sécurise

On récupère le mot de passe brut (car on va le comparer avec password_verify())

5) On vérifie que les champs ne sont pas vides
6) Chercher l'utilisateur dans la base

On cherche un utilisateur avec cet email.
S'il existe, $user contient un objet User.

7) vérifier le mot de passe avec password_verify

On vérifie que :

- L'utilisateur existe
- Le mot de passe saisi correspond au mot de passe haché en base

8) Connexion réussie : enregistrer en session

On stocke :

- L'ID de l'utilisateur
- Son nom complet
- Son rôle (admin, client…)
- Un message de bienvenue

```
                ob_clean();
                header('Location: index.php?page=profile');
                exit();
            } else {
                $_SESSION['message'] = 'Email ou mot de passe invalide.';
                $_SESSION['success'] = false;
            }
        } else {
            $_SESSION['message'] = "Veuillez renseigner à la fois l'email et le mot de passe.";
            $_SESSION['success'] = false;
        }
    }

    $message = $_SESSION['message'] ?? null;
    $success = $_SESSION['success'] ?? null;
    unset($_SESSION['message'], $_SESSION['success']);

    echo $this->twig->render('userController/login.html.twig', [
        'session' => [
            'message' => $message,
            'success' => $success
        ]
```

Affichage du formulaire avec message éventuel

- On récupère les messages de session (s'il y en a)
- On les transmet à la vue Twig `login.html.twig`
- Puis on efface les messages de la session pour éviter qu'ils s'affichent à nouveau après actualisation

```
public function logout()
{
    if (session_status() === PHP_SESSION_NONE) {
        session_start();
    }

    $_SESSION = [];


    if (ini_get('session.use_cookies')) {
        $params = session_get_cookie_params();
        setcookie(
            session_name(),
            '',
            time() - 42000,
            $params['path'],
            $params['domain'],
            $params['secure'],
            $params['httponly']
        );
    }

    session_destroy();

    header('Location: index.php');
    exit();
```

1)Vider toutes les données de la session

Cela supprime toutes les variables stockées dans $_SESSION :

2) Supprimer le cookie de session (si utilisé)

Cela supprime le cookie PHP qui contient l'ID de session dans le navigateur.

Nécessaire pour que la déconnexion soit vraiment complète, même côté client.

3) Détruire la session sur le serveur

Termine la session PHP sur le serveur.
C'est la vraie déconnexion serveur.

4) Redirection vers la page d'accueil

L'utilisateur est redirigé automatiquement vers la page d'accueil (ou toute autre page publique).

Methode Registration dans UsercController.php

```php
public function register()
{
    if ($_SERVER['REQUEST_METHOD'] === 'POST') {
        $firstName = filter_input(INPUT_POST, 'first_name', FILTER_SANITIZE_STRING);
        $lastName = filter_input(INPUT_POST, 'last_name', FILTER_SANITIZE_STRING);
        $email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);
        $phone = filter_input(INPUT_POST, 'phone', FILTER_SANITIZE_STRING);
        $password = $_POST['password'] ?? '';
        $confirmPassword = $_POST['confirm_password'] ?? '';

        if (!empty($firstName) && !empty($lastName) && !empty($email) && !empty($password)) {
            if ($password !== $confirmPassword) {
                $_SESSION['message'] = 'Les mots de passe ne correspondent pas.';
                $_SESSION['success'] = false;
            } else {
                $existingUser = $this->userModel->getUserByEmail($email);

                if ($existingUser !== null) {
                    $_SESSION['message'] = 'Cet email existe déjà. Veuillez vous connecter.';
                    $_SESSION['success'] = false;
                } else {
                    $hashedPassword = password_hash($password, PASSWORD_DEFAULT);
```

Elle permet à un nouvel utilisateur de s'inscrire sur ton site en créant un compte.

1)On vérifie si l'utilisateur vient de valider le formulaire d'inscription. avec la methode POST

2) On récupère les données du formulaire et on les nettoie :

- filter_input pour éviter les attaques (XSS, injection…)
- Le mot de passe est lu brut pour être haché ensuite

3)Tous les champs obligatoires doivent être remplis, sinon → message d'erreur.

4) Si les mots de passe ne sont pas identiques → erreur utilisateur.

5) Si un compte existe déjà avec cet email → l'utilisateur est invité à se connecter au lieu de s'inscrire.

6) Hachage du mot de passe

On sécurise le mot de passe en le hachant.

```
                $user = new User(
                    null,
                    $firstName,
                    $lastName,
                    $email,
                    $phone,
                    $hashedPassword,
                    'client'
                );

                $success = $this->userModel->createUser($user);

                if ($success) {
                    $_SESSION['message'] = 'Votre compte a été créé avec succès. Vous pouvez maintenant vous connecter';
                    $_SESSION['success'] = true;
                    header('Location: index.php?page=login');
                    exit;
                } else {
                    $_SESSION['message'] = 'Erreur lors de l\'inscription.';
                    $_SESSION['success'] = false;
                }
            }
        }
    } else {
        $_SESSION['message'] = 'Veuillez remplir tous les champs obligatoires.';
        $_SESSION['success'] = false;
    }
}

echo $this->twig->render('userController/register.html.twig', ['session' => $_SESSION ?? []]);
```

1)Création d'un objet User

On crée un objet utilisateur avec le rôle "client".

2) L'utilisateur est ajouté à la base de donnée

3) En cas de succès → redirection vers la page de connexion.

Sinon → message d'erreur affiché dans le formulaire.

Amélioration :

1) Possibilité de mettre plus d'image pour chaque chambre
2) mode de paiement réel et fonctionnel
3) Réglage de la notification
4) Ajout des avoirs

Problèmes rencontrés :

Au sein de notre projet nous avons rencontré quelques soucis comme la refonte de la base de données plusieurs fois, car nous n'étions pas satisfait de nos anciennes bases de données et avons donc choisi une version finale de la base de données qui nous paraît complète.

Conclusion :

Ce projet nous a permis de travailler en groupe en développant un site web pour un hôtel, en utilisant du HTML, du CSS et du PHP. Nous avons également dû interagir avec une base de données. Grâce à ce projet, nous avons pu acquérir de nouvelles compétences, notamment en maîtrisant différents langages de programmation et en apprenant à gérer l'interaction avec la bases de données. Cette expérience a renforcé notre capacité à collaborer tout en développant des solutions concrètes dans un contexte réel.

Sources

- Documentation PHP officielle.
- Tutoriels pour la gestion des BDD MySQL avec PHP.
- Forums et communautés.
- Modèles Bootstrap gratuits.