Séjour

Plateforme de recherche et de réservation d'hôtels

DOCUMENTATION — SUPPORT DÉVELOPPEUR

Réalisation Professionnelle — Projet AP / TP — Épreuves E5 / E6

PRESENTATION

Contexte : Système d'Information & règles de gestion

« Séjour » est une application web permettant à un internaute de consulter un catalogue d'hôtels, de les rechercher, de les mettre en favori, de réserver une chambre sur une période donnée, et de laisser une critique après un séjour effectué.

Règles de gestion identifiées dans le système :

- Un client doit être connecté pour réserver, aimer un hôtel, ou laisser une critique.
- Un client ne peut laisser qu'une seule critique par hôtel.
- Une critique n'est autorisée que si le client a préalablement réservé une chambre dans cet hôtel.
- Une chambre ne peut pas être réservée par deux clients sur des périodes qui se chevauchent.
- La date d'arrivée doit être antérieure à la date de départ.

Liste des spécifications fonctionnelles : fonctionnalités attendues de l'application

Fonctionnalité	Non connecté	Connecté
Consulter la liste des hôtels	Oui	Oui
Consulter le détail d'un hôtel	Oui	Oui
Rechercher un hôtel (nom / adresse / services)	Oui	Oui
S'inscrire / se connecter / se déconnecter	Oui (inscription, connexion)	Oui (déconnexion)
Réserver une chambre	Non	Oui
Ajouter un hôtel en favori	Non	Oui
Laisser une critique	Non	Oui, si déjà réservé, une seule fois
Sauvegarder ses services préférés	Non	Oui
Consulter son profil	Non	Oui

Liste des spécifications techniques : contraintes techniques

- Langage serveur : PHP procédural, organisation inspirée du modèle MVC
- Base de données : MySQL / MariaDB, accès via PDO (requêtes préparées)
- Frontend : HTML5, CSS3 (variables CSS), Google Fonts (Playfair Display, Inter)
- Authentification : sessions PHP natives ($_SESSION)
- Hébergement mutualisé compatible PHP/MySQL

Choix fonctionnels justifiés

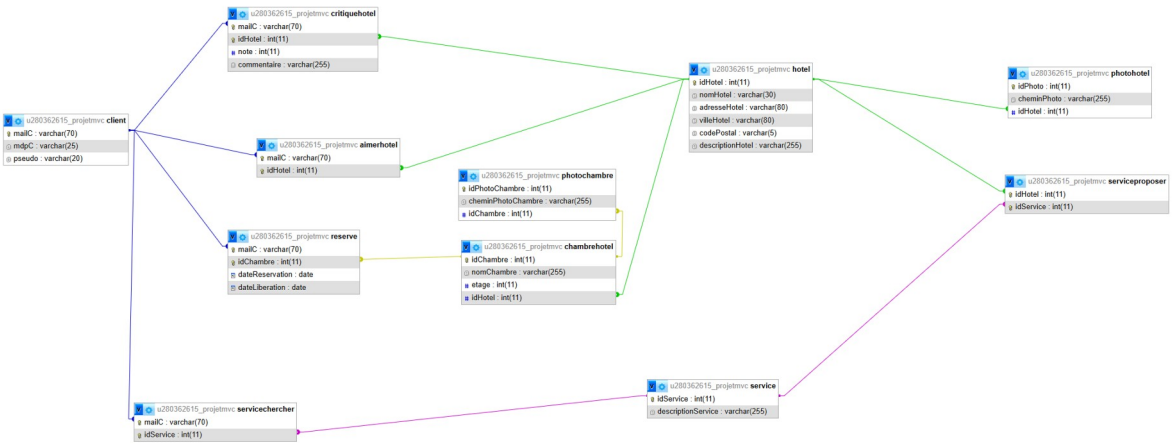
- Réservation en deux étapes (dates puis chambre) pour ne proposer que des chambres réellement disponibles.
- Critique conditionnée à une réservation effective, pour garantir la fiabilité des avis (cf. CGU, lutte contre les faux avis).
- Recherche multicritère (nom / adresse / services) pour couvrir différents profils d'utilisateurs.

Choix techniques justifiés

- PDO plutôt que mysqli : requêtes préparées natives, protection contre les injections SQL.
- Architecture MVC : sépare modele/ (accès données), controleur/ (logique de flux) et vue/ (présentation).
- Point d'entrée unique (index.php) avec table de routage actions → fichiers (controleurPrincipal.php).
- CSS à base de variables pour homogénéiser la charte graphique.

BASE DE DONNEES

MCD et/ou MLD et/ou diagramme de classes



console de requêtes SQL

client

Colonne	Type	Contrainte
mailC	VARCHAR	PK
mdpC	VARCHAR	Mot de passe haché
pseudo	VARCHAR	—

hotel

Colonne	Type	Contrainte
idHotel	INT	PK
nomHotel	VARCHAR	—

descriptionHotel	TEXT	—
adresseHotel	VARCHAR	—
codePostal	VARCHAR	—
villeHotel	VARCHAR	—

chambrehotel

Colonne	Type	Contrainte
idChambre	INT	PK
idHotel	INT	FK → hotel
nomChambre	VARCHAR	Nullable
etage	INT	Nullable

reserve

Colonne	Type	Contrainte
mailC	VARCHAR	FK → client
idChambre	INT	FK → chambrehotel
dateReservation	DATE	Arrivée
dateLiberation	DATE	Départ

critiquehotel

Colonne	Type	Contrainte
mailC	VARCHAR	FK → client
idHotel	INT	FK → hotel
note	INT	1 à 5
commentaire	TEXT	—

Autres tables

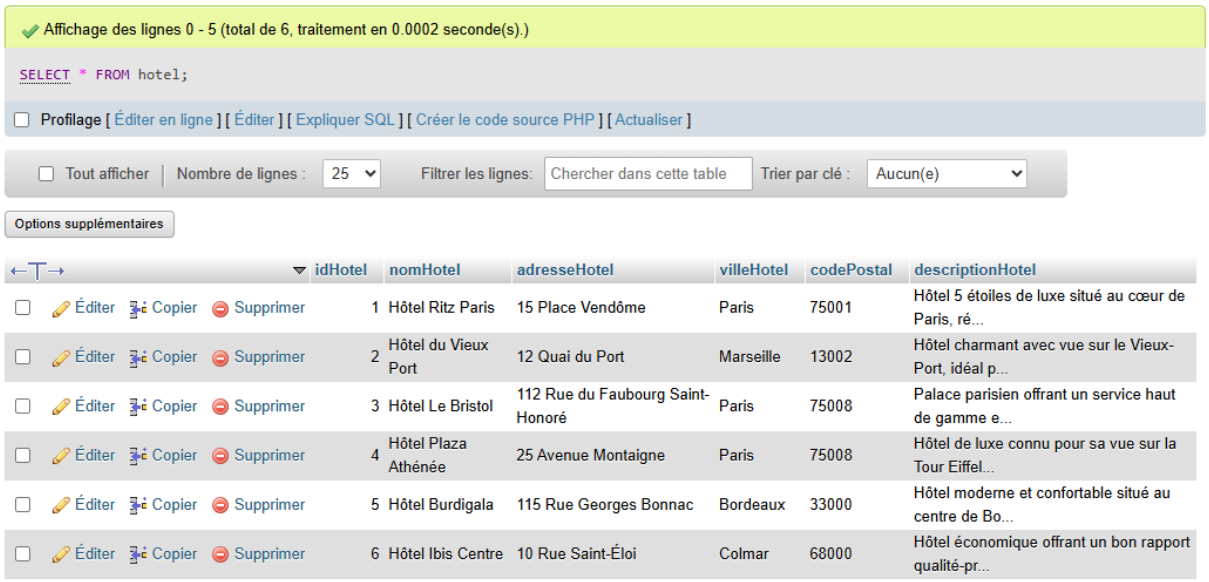
- aimerhotel(mailC, idHotel) — favoris
- service(idService, descriptionService) — référentiel des services
- serviceproposer(idHotel, idService) — services proposés par un hôtel
- servicechercher(mailC, idService) — services préférés d'un client
- photohotel(idHotel, cheminPhoto) / photochambre(idChambre, cheminPhotoChambre)

Cohérence entre les représentations

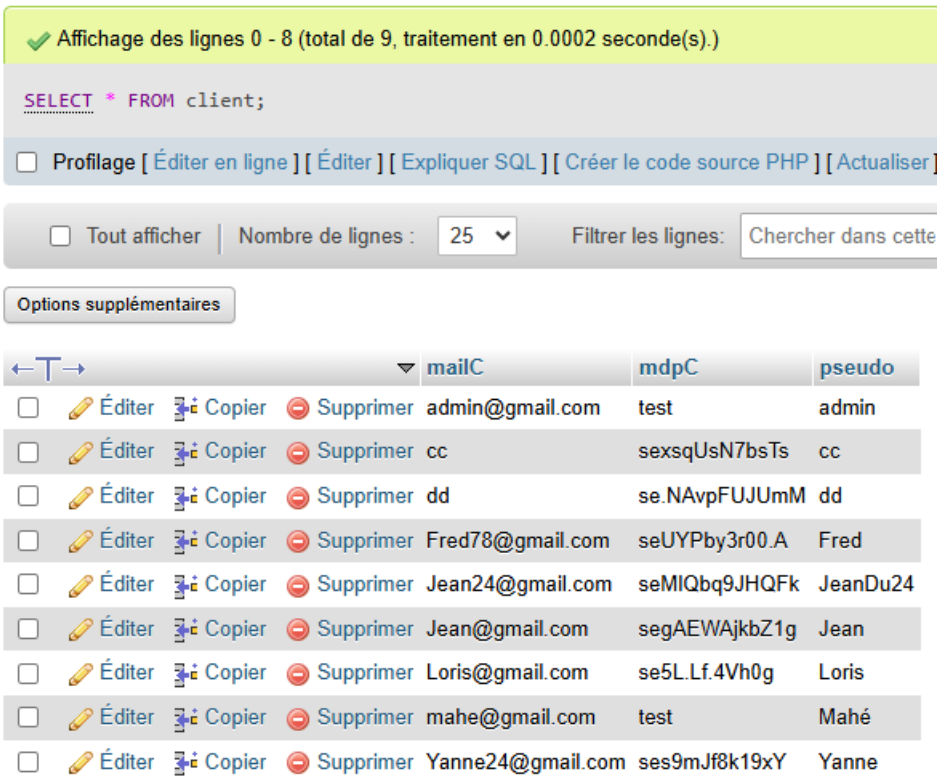
Le MCD/MLD doit correspondre aux jointures effectivement utilisées dans le modèle : par exemple, getReservationsByMailC() joint reserve → chambrehotel → hotel, ce qui confirme que reserve référence chambrehotel et non directement hotel.

Jeu d'essai : description si besoin

Champ	Valeur de test
Email	admin@gmail.com
Mot de passe	test







Affichage des lignes 0 - 8 (total de 9, traitement en 0.0002 seconde(s).)

```
SELECT * FROM client;
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le code source PHP] [Actualiser]

Tout afficher | Nombre de lignes : 25 Filtrer les lignes: Chercher dans cette

Options supplémentaires

		mailC	mdpC	pseudo
☐	Éditer Copier Supprimer	admin@gmail.com	test	admin
☐	Éditer Copier Supprimer	cc	sexsqUsN7bsTs	cc
☐	Éditer Copier Supprimer	dd	se.NAvpFUJUmM	dd
☐	Éditer Copier Supprimer	Fred78@gmail.com	seUYPby3r00.A	Fred
☐	Éditer Copier Supprimer	Jean24@gmail.com	seMlQbq9JHQFk	JeanDu24
☐	Éditer Copier Supprimer	Jean@gmail.com	segAEWAjkbZ1g	Jean
☐	Éditer Copier Supprimer	Loris@gmail.com	se5L.Lf.4Vh0g	Loris
☐	Éditer Copier Supprimer	mahe@gmail.com	test	Mahé
☐	Éditer Copier Supprimer	Yanne24@gmail.com	ses9mJf8k19xY	Yanne

APPLICATION

Organisation/Structure :

MVC ou autre

Le projet suit une architecture inspirée du modèle MVC, répartie en trois répertoires : modele/ (accès aux données), controleur/ (logique de traitement) et vue/ (affichage). Le point d'entrée unique index.php route les requêtes via le paramètre GET action et la fonction controleurPrincipal().

```
// index.php — point d'entrée
$action = $_GET['action'] ?? 'defaut';
$fichier = controleurPrincipal($action);   // ex : 'detailHotel.php'
include "$racine/controleur/$fichier";
```

Liste des répertoires, fichiers : arborescence et/ou schéma

```
/
├── index.php / site-maintenance.php / getRacine.php
├── controleur/
```

```
│   ├── controleurPrincipal.php
│   ├── listeHotel.php / detailHotel.php / rechercheHotel.php
│   ├── connexion.php / deconnexion.php / inscription.php
│   ├── monProfil.php / reservation.php / aimer.php / cgu.php
├── modele/
│   ├── bd.inc.php  (connexion PDO)
│   ├── bd.hotel.inc.php / bd.chambre.inc.php / bd.reservation.inc.php
│   ├── bd.critiquer.inc.php / bd.aimer.inc.php / bd.service.inc.php
│   ├── bd.photo.inc.php / bd.utilisateur.inc.php
│   └── authentification.inc.php
├── vue/
│   ├── entete.html.php / pied.html.php
│   └── vue<Fonctionnalité>.php  (une vue par écran)
├── css/  (base1.css, form1.css, corps1.css, cgu1.css)
└── photos/

└──images/
```

Règles de développement utilisées : nommage, taille des fichiers,...

- Fichiers modèle préfixés bd.<entité>.inc.php, suffixe .inc.php pour signaler un fichier à inclure.
- Fonctions nommées verbe + entité en camelCase : getHotelByIdHotel, addReservation, delAimer...
- Chaque contrôleur teste s'il est appelé directement pour définir $racine, et reste exécutable isolément.
- Chaque fichier modèle embarque un bloc de test autonome (exécuté seulement en accès direct).
- Séparation stricte : aucune requête SQL dans les vues, qui n'affichent que des variables déjà préparées par le contrôleur.

MODELE

POO

Le modèle n'est pas construit avec une approche orientée objet stricte : pas de classes PHP, mais des fichiers regroupant des fonctions par entité métier (hôtel, chambre, client, réservation, critique, service, photo). Chaque fonction encapsule une requête PDO préparée, ce qui reproduit le principe de responsabilité unique d'une couche d'accès aux données sans recourir au vocabulaire objet.

Un passage du projet vers le développement orienté objets est envisagée

Classes Techniques : utilisées et/ou créées

- PDO (PHP Data Objects) : classe native utilisée pour la connexion et l'exécution de requêtes préparées.
- $_SESSION : mécanisme natif de session PHP utilisé pour l'état de connexion (authentification.inc.php).

Classes d'accès aux données : connexion, communication (requêtes SQL)

```
// modele/bd.inc.php
function connexionPDO() {
    $conn = new PDO("mysql:host=$serveur;dbname=$bd", $login, $mdp,
        array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'UTF8'"));
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    return $conn;
}
```

Toutes les requêtes utilisent des requêtes préparées avec liaison de paramètres typés (bindValue + PDO::PARAM_INT / PDO::PARAM_STR), ce qui protège contre les injections SQL. Exemple (bd.hotel.inc.php) :

Fonction	Rôle	Requête SQL (principe)
getHotelByIdHotel($idHotel)	Récupère un hôtel	SELECT * FROM hotel WHERE idHotel=:idHotel
getHotelsByNomHotel($nom)	Recherche par nom	SELECT * FROM hotel WHERE nomHotel LIKE :nom
getHotelsByServices($ids)	Hôtels proposant tous les services cochés	Sous-requête COUNT(DISTINCT ...) = nb services demandés
getHotelsAimesByMailC($mailC)	Favoris d'un client	Jointure hotel / aimerhotel

Organisation et rôle des fichiers

Fichier	Rôle
bd.inc.php	Connexion PDO à la base de données
bd.hotel.inc.php	Fiche, recherche, favoris des hôtels
bd.chambre.inc.php	Chambres et disponibilité
bd.reservation.inc.php	CRUD des réservations
bd.critiquer.inc.php	Critiques/avis
bd.aimer.inc.php	Favoris (« j'aime »)
bd.service.inc.php	Services et préférences de recherche

bd.photo.inc.php	Photos d'hôtel
bd.utilisateur.inc.php	Comptes clients
authentification.inc.php	Connexion/déconnexion, session

CONTROLEURS

Cohérence avec les fonctionnalités

Chaque fonctionnalité correspond à un couple action/contrôleur unique, recensé dans controleurPrincipal.php :

action=	Contrôleur	Fonctionnalité
defaut / liste	listeHotel.php	Accueil, liste des hôtels
detail	detailHotel.php	Fiche hôtel + critiques
recherche	rechercheHotel.php	Recherche multicritère
connexion	connexion.php	Authentification
deconnexion	deconnexion.php	Fin de session
inscription	inscription.php	Création de compte
profil	monProfil.php	Espace personnel
reservation	reservation.php	Réservation en 2 étapes
aimer	aimer.php	Favoris
cgu	cgu.php	CGU

Chaque contrôleur suit le même schéma : inclusion du modèle nécessaire, traitement GET/POST, appel des fonctions du modèle, préparation des variables, inclusion de la vue (entête / vue spécifique / pied).

Navigation dans l'application : modèle et vues

- Toute la navigation transite par index.php?action=... : aucun contrôleur n'est appelé en dehors de ce point d'entrée.
- Un utilisateur non connecté accédant à une action protégée (profil, réservation) reçoit le formulaire de connexion à la place du contenu.
- Après connexion réussie, redirection automatique vers le profil (connexion.php inclut directement monProfil.php).

Chaque contrôleur orchestre modèle → vue en assignant ses variables ($titre, $listeHotel, $unHotel, etc.) puis en incluant successivement entete.html.php, la vue spécifique, et pied.html.php — la vue ne fait qu'afficher ces variables déjà prêtes.

ANALYSE PERSONNELLE

Apports en compétences

Ce projet m'a permis de vraiment comprendre l'intérêt concret de l'architecture MVC (séparer modèle, contrôleur et vue rend le code bien plus facile à faire évoluer) et de maîtriser les requêtes préparées PDO ainsi que la gestion de session PHP pour l'authentification. J'ai aussi progressé sur la structuration d'un projet multi-fichiers et sur l'organisation d'une charte graphique cohérente via des variables CSS.

Difficultés

La principale difficulté a été de gérer correctement les dates de réservation, notamment pour vérifier la disponibilité d'une chambre sur une période donnée sans provoquer de chevauchement entre deux réservations. J'ai aussi eu du mal, au début, à bien faire circuler les données entre contrôleur et vue sans dupliquer de logique métier dans l'affichage.

Evolution

- Renforcement de la sécurité des mots de passe (password_hash) et retrait des blocs de test exposant des données.
- Vérification de la disponibilité de la chambre au moment de la validation finale, pour éviter une double réservation.
- Ajout d'une protection CSRF sur les actions sensibles.

À terme, il serait intéressant d'ajouter un module de paiement en ligne pour finaliser réellement les réservations, ainsi qu'un back-office d'administration pour gérer les hôtels et modérer les avis. Une messagerie simple entre client et hôtelier pourrait aussi enrichir l'expérience utilisateur après une réservation