DOCUMENTATION TECHNIQUE

L'OPALE BLANCHE

APPLICATION WEB DE RÉSERVATION

Victoria Tahay

2025

Réalisé dans le cadre de la formation Développeur Web Full Stack.

ilaria Digital School

TABLE DES MATIÈRES

INTRODUCTION

L'Opale Blanche est une application web de réservation en ligne dédiée à un chalet de bien-être. Elle permet aux utilisateurs de réserver facilement une table au restaurant, un accès au spa ou encore un soin ou un massage. Grâce à un calendrier interactif et une gestion intuitive des profils et des réservations, l'application offre une expérience moderne, accessible et fluide.

Conçue pour répondre aux besoins des clients et des gestionnaires de services, L'Opale Blanche simplifie la gestion des réservations tout en garantissant une navigation agréable et adaptée à tous les appareils.

I. FRONT-END

1. Les technologies utilisées

A. Les langages

HTML5 : Standard universel pour la structuration du contenu web. Permet une bonne accessibilité et un bon référencement naturel (SEO). Intégré directement dans les pages et composants jsx.

CSS3 : Permet une personnalisation visuelle avancée avec des animations, des transitions, etc. Gère la responsivité avec des outils comme les media queries (cf. Framework CSS) pour adapter le design aux différents écrans (smartphones, tablettes ou encore ordinateurs), et notamment avec des fonctionnalités modernes comme Flexbox et Grid.

JavaScript : Ajoute de l'interactivité nécessaire à une expérience utilisateur fluide (barre de navigation dynamique). Prend en charge les frameworks modernes (comme React) pour une structure plus modulaire et maintenable.

JSX (JavaScript XML) : Dans le cadre du développement de mon site avec React et Vite.js (cf. Framework), j'utilise JSX (JavaScript XML), une extension syntaxique de JavaScript qui permet d'écrire du code HTML directement dans les fichiers JavaScript. JSX facilite la création d'interfaces utilisateur en rendant le code plus lisible et proche du HTML standard, tout en bénéficiant de la puissance de JavaScript. Il permet d'incorporer des expressions JavaScript en les encapsulant dans des accolades {} et favorise la modularisation des composants en React. Grâce à JSX, la structure de l'application est plus claire et mieux organisée, ce qui améliore la maintenance du code et la réutilisabilité des composants.

B. Les Media Queries

Essentielles pour garantir une expérience utilisateur optimale adaptée à un large panel de tailles d'écrans : (smartphones, tablettes, ordinateurs), en modifiant la mise en page, les tailles de texte et l'affichage des éléments en fonction des besoins des utilisateurs. L'utilisation du modèle Flexbox a notamment permis d'adapter plus facilement les différents blocs d'éléments sur les petits appareils (en particulier pour les images des services sur la page d'accueil et pour les blocs combinés images/textes de la page à propos.)

Page d'accueil – Home.jsx

Page à propos - About.jsx

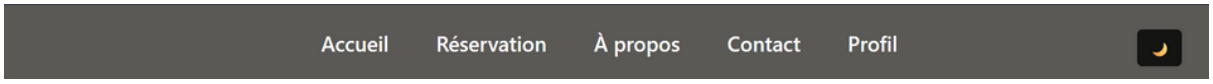
Cela permet : d'améliorer la lisibilité en ajustant la taille des polices et des espacements, de réorganiser les blocs de contenu pour qu'ils restent ergonomiques même sur de petits écrans (exemples : pas besoin de zoom ou de défilement horizontal excessif, éviter les erreurs de clic), de naviguer de manière fluide (exemple : changement de disposition du menu en version mobile). (cf. tests browserstack)

Les Media Queries permettent aussi d'optimiser la performance en chargeant uniquement les styles qui sont nécessaires, réduisant ainsi la consommation de ressources et évitant les redirections inutiles entre mobile et ordinateurs. En outre, elles améliorent le référencement en rendant le site « mobile-friendly », un critère essentiel pour le SEO.

2. Les principales fonctionnalités implémentées

A. La barre de navigation dynamique.

La barre de navigation du composant *Header.jsx* est dynamique et s'adapte automatiquement à l'état de connexion de l'utilisateur. L'affichage des éléments du menu est géré à partir du fichier data/menuItems.js, qui centralise les données de la barre de navigation. Cette logique permet par exemple d'afficher "Profil" si l'utilisateur est connecté, ou "Connexion" s'il ne l'est pas. En plus de cette navigation conditionnelle, un utilisateur déconnecté ne peut pas accéder à la page de profil : l'accès est restreint et sécurisé. Cela garantit une expérience utilisateur fluide tout en respectant les règles d'authentification.



Barre de navigation avec les items des pages et le bouton du mode sombre - Header.jsx

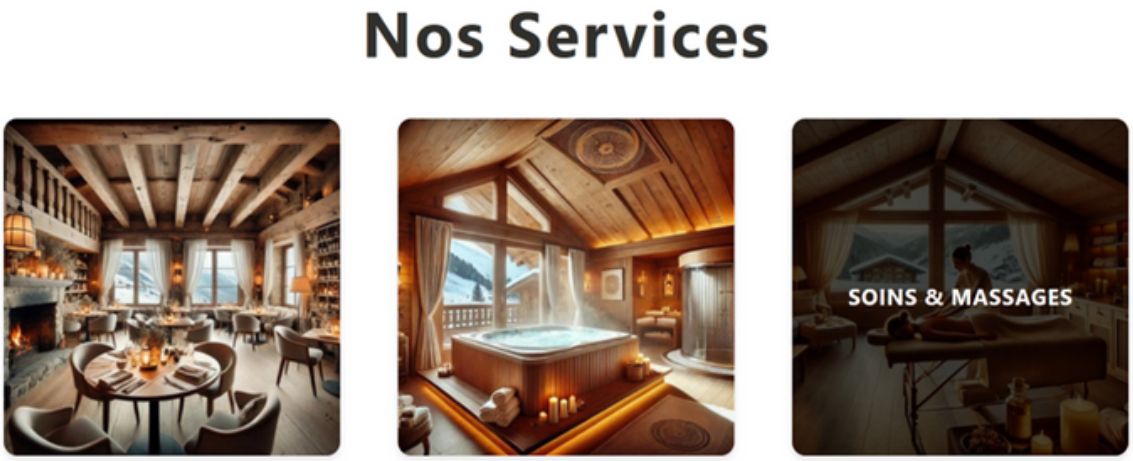
B. L’affichage des services

Sur la page d’accueil (Home.jsx), un bloc d’images met en avant les différents services proposés (récupéré dynamiquement via le composant *ServiceList.jsx* et le fichier *getServices.php* relié à la base de données, table “services”*)*, permettant à l’utilisateur de visualiser rapidement les prestations disponibles. Il peut sélectionner le service qui l’intéresse en cliquant directement sur l’image correspondante, facilitant ainsi la navigation vers les pages dédiées.

Une interaction visuelle fluide et immersive a été mise en place pour renforcer l’expérience utilisateur. Lors du survol d’une image, plusieurs effets se déclenchent :

✓ **Affichage du titre de la catégorie sur un filtre sombre semi-transparent, améliorant la lisibilité.**
✓ **Léger agrandissement de l’image, créant un effet d’animation subtil.**
✓ **Transition fluide et progressive, assurant une expérience douce et harmonieuse.**

Cette animation a été conçue pour être lente et apaisante, afin de préserver l’atmosphère paisible et relaxante que propose le chalet bien-être. L’objectif est d’offrir une navigation à la fois esthétique et intuitive, en accord avec l’image et les valeurs du site.



Images des services sur la page d’accueil menant aux pages dédiées. Le survol est actif sur l’image du spa. – home.jsx

Chaque service dispose d'une page dédiée (dossier pages/services) comprenant une description détaillée, des images et les tarifs associés. Cette organisation permet de mettre en valeur chaque prestation tout en facilitant la navigation.

Un bouton d'action « Réserver » est intégré sur chaque page de service, directement sous la description de celui-ci. Il redirige directement vers l'ancre correspondante à la bonne catégorie sur la page de réservation, garantissant ainsi une transition fluide entre la présentation du service et l'acte de réservation.

Cette séparation entre présentation et réservation du service répond à un double objectif :

✓ **Améliorer l'expérience utilisateur en offrant des pages de services détaillées, imagées et bien structurées pour donner envie de réserver un service.**

✓ **Optimiser le processus de réservation en proposant une interface simplifiée, réduisant le texte au strict nécessaire pour permettre une réservation rapide et intuitive. Par exemple, pour les soins et les massages, une présentation exhaustive incluant chaque prestation, les horaires et la description complète de la catégorie occuperait trop d'espace sur la page de réservation.**

Grâce à cette approche, la page réservation reste épurée et efficace, ce qui permet un accès rapide aux services souhaités.

C. La réservation du service

La partie réservation du site (*Reservation.jsx*) a été conçue pour être à la fois pratique et immersive. Lorsqu'un utilisateur clique sur un service, le formulaire de réservation s'ouvre en dessous et récupère les services via *getServices.php*. Cette approche permet de garder une vue globale sur les autres catégories de services tout en facilitant la réservation. L'affichage des services est dynamique avec des boutons pour chaque catégorie.

La section de réservation contient un formulaire interactif et le prix total de la réservation à payer sur place où l'utilisateur peut sélectionner :

✓ **Une date disponible via un calendrier dynamique.**
✓ **Un horaire limité aux horaires d'ouverture des services mais aussi disponibles.**
✓ **Le nombre de personnes concernées par la réservation.**

Affichage de la section de réservation avec le calendrier interactif. – Reservation.jsx

D. Un Calendrier Interactif et Intelligent

L'un des points forts de cette interface est l'intégration d'un calendrier interactif, le *React Calendar*, qui simplifie la sélection des disponibilités. Les dates et horaires disponibles sont automatiquement synchronisés avec la base de données, garantissant que l'utilisateur ne peut réserver que des créneaux réellement ouverts. Cette fonctionnalité évite les conflits de réservation et améliore l'expérience utilisateur en proposant uniquement des créneaux disponibles, via le fichier *getSlots.php*.

Le fichier *createReservation.php* permet ensuite de créer la réservation et de l'intégrer dans la base de données l_opale_blanche, dans la table "reservations".

E. La gestion des réservations

Dans l'application, la gestion des réservations est centralisée à travers deux interfaces distinctes sur le profil (accessibles seulement par les rôles correspondants) : *ProviderReservations.jsx* (gestion des réservations des gestionnaires du Restaurant et du Spa) et *AdminReservations.jsx* (gestion des réservations de l'administrateur), chacune ayant sa propre page dédiée côté serveur en PHP.

Pour le **rôle administrateur**, la page lui permet d'accéder à l'ensemble des réservations du site (avec *getAllReservations.php*). Il peut filtrer les réservations par date afin d'afficher uniquement celles d'un jour spécifique. Il dispose également d'une option pour supprimer des réservations (avec *adminDeleteReservation.php*), garantissant ainsi une gestion efficace et fluide des données.

Du côté du **rôle de gestionnaire**, l'affichage des réservations est restreint à celles de sa propre catégorie : Restaurant ou Spa. Les services sont triés par catégorie, et le gestionnaire peut sélectionner un jour ainsi qu'une catégorie pour obtenir une vue précise des réservations correspondantes (avec *getReservationsProvider.php*).

Une fois les réservations affichées, il a la possibilité de mettre en évidence l'état de chaque réservation en cliquant dessus :

- ✓ **Vert si le client a honoré sa réservation (1 clic),**
- ✓ **Rouge s'il ne s'est pas présenté (2 clics),**
- ✓ **Neutre par défaut (3 clics).**

Cette organisation permet à chaque intervenant d'avoir une gestion claire et adaptée à son rôle, garantissant un suivi optimal des réservations.

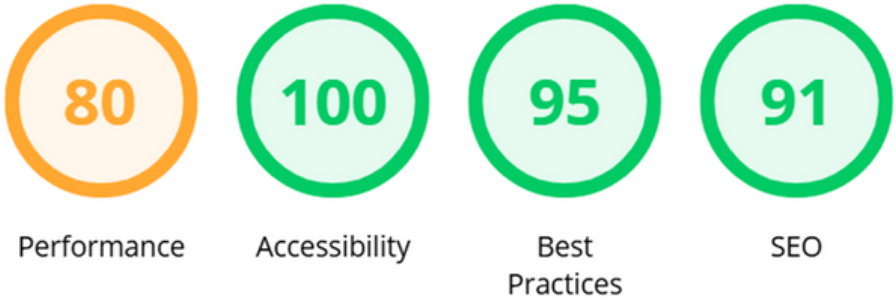
Exemple de la page de gestion des réservations du gestionnaire du restaurant, avec la coloration rouge sur la réservation - ProviderReservations.jsx

3. L'Accessibilité et l'*Audit Web* (Tests réalisés)

A. Les tests généraux avec *Google Lighthouse*

Afin d'évaluer la performance, l'accessibilité, la conformité aux meilleures pratiques ainsi que l'optimisation SEO, des tests ont été réalisés avec Google Lighthouse. D'après ces tests, l'application web respecte au mieux les normes ARIA et WCAG visant à la rendre accessible aux personnes en situation de handicap. Ces analyses ont permis d'identifier des axes d'amélioration de la performance, concernant la vitesse de chargement de certains éléments du site (voir les résultats dans le dossier test du projet).

Voici une moyenne des différents résultats obtenus :



✓ **Concernant le contraste des couleurs, *Contrast Finder* a été utilisé en amont de l'analyse précédente.**

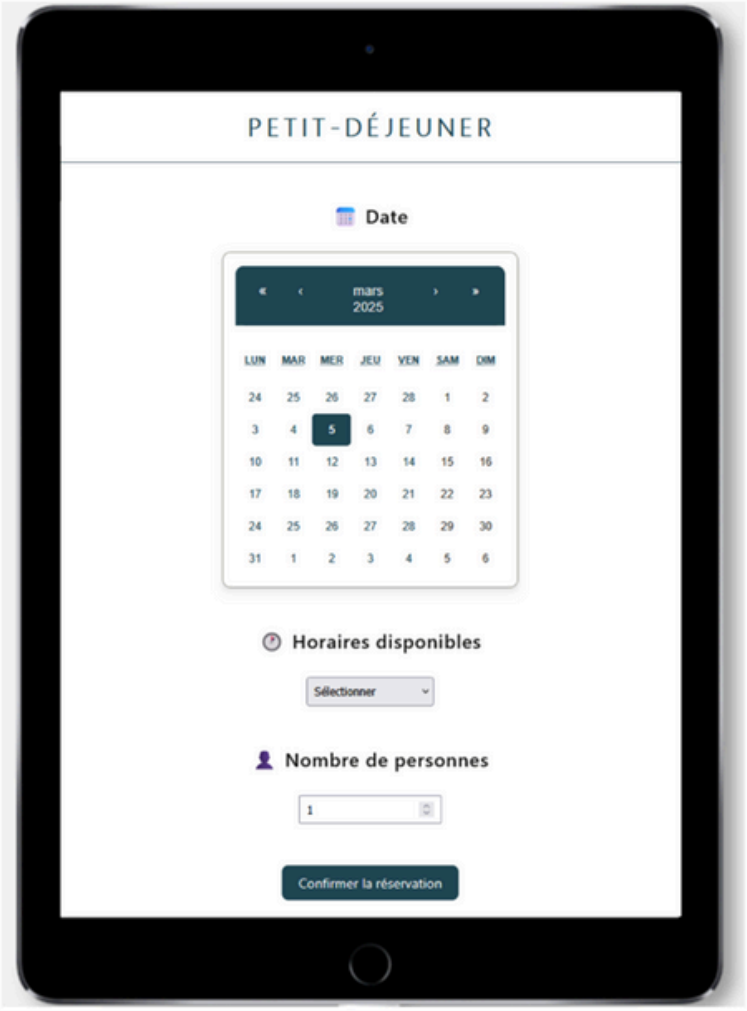
✓ **Une recherche sur les caractéristiques typographiques a également été réalisée afin d'adapter les textes aux personnes dyslexiques ou présentant d'autres déficiences.**

✓ **Le format webp a été choisi pour les images car il est plus moderne et optimisé que png ou jpg par exemple. En effet, il réduit la taille des fichiers tout en préservant la qualité visuelle. la fonction "alt" est correctement détaillé pour décrire au mieux les images.**

✓ **Le site est entièrement accessible au clavier.**

B. Les tests de responsivité avec *Browserstack*

Des tests de responsivité permettant de vérifier l'affichage et l'adaptabilité du site ont également été réalisés avec Browserstack, sur différents appareils et navigateurs (Safari, Google, Firefox, etc.) tels que : les Ipads 9 et Air 6, le Samsung Galaxy Fold 5, le Samsung Galaxy S23, les Macs 133 et 134 notamment (cf. dossier test). L'évaluation confirme que le site s'adapte correctement aux différentes tailles d'écran, garantissant une navigation fluide à l'utilisateur, un bon agencement des éléments et une lisibilité optimale sur tous les supports numériques.



Page de réservation avec section du formulaire sur Ipad 9 - Reservation.jsx

C. Le mode sombre

Pour améliorer le confort des utilisateurs, un mode *Dark/Light* est intégré, permettant de changer les couleurs de l'interface, via un bouton en haut à droite du site qui appelle la fonction du fichier *ThemeToggle.jsx* (dans omponents) pour une meilleure lisibilité. Des tests de contrastes sont effectués afin d'optimiser l'accessibilité. Ce mode sombre contribue également à réduire la fatigue visuelle et à économiser de l'énergie, notamment sur les écrans OLED.

Exemple du monde sombre sur la page Home.jsx

II. BACK-END

1. Les technologies utilisées

A. Le langage

PHP a été choisi pour implémenter la logique serveur de l'application. Le backend fonctionne comme une API REST, qui reçoit les requêtes du frontend React, interagit avec la base de données MySQL sur *phpmyadmin*, puis renvoie les données au format JSON. Cela permet de maintenir une architecture claire et découplée entre le client et le serveur.

2. Les fonctionnalités implémentées

A. Sécurité et API REST

✓ **Création d'une API REST : tous les échanges de données (services, utilisateurs, réservations, créneaux) passent par des endpoints personnalisés (GET, POST, PUT, DELETE), comme dans *getReservations.php*, *getSlots.php* et *updateReservation.php*, qui renvoient des réponses structurées en JSON pour être affichées par le frontend React.**

✓ **Requêtes CRUD sécurisées : les opérations Create, Read, Update et Delete sont toutes gérées via requêtes préparées (avec mysqli), ce qui protège contre les injections SQL. On utilise des paramètres bindés dans tous les fichiers de requêtes (par exemple dans *createReservation.php*, *getUser.php*, etc.) qui permettent de sécuriser les requêtes SQL en séparant les données du code. Au lieu d'insérer les valeurs directement dans la requête (ce qui est risqué), on utilise des marqueurs (?) et on lie les valeurs ensuite avec bind_param().**

✓ **Sécurisation des entrées : avant traitement, les données issues des formulaires ou des requêtes AJAX sont nettoyées ou validées. Même si htmlspecialchars() n'est pas toujours appelé directement dans les fichiers PHP, le frontend React traite les entrées avec attention, ce qui limite les risques de XSS côté client. Les réponses sont échangées au format JSON, ce qui évite l'interprétation directe du HTML**

✓ **Gestion d'authentification : des fichiers comme *checkAuth.php* et *getUserRole.php* permettent de vérifier dynamiquement l'état de connexion de l'utilisateur, ce qui sécurise l'accès aux zones protégées (comme l'espace "admin" ou "provider").**

✓ **Intégration d'un système de réinitialisation de mot de passe en cas d'oublie par exemple, basé sur l'envoi d'un token sécurisé par email.**

B. La base de données avec *phpmyadmin*

MySQL est utilisé pour stocker toutes les informations du site, telles que :

- **Les utilisateurs** (nom, email, rôle).
- **Les services** proposés à la réservation.
- **Les réservations** effectuées, avec les dates et heures disponibles.

Organisation de la base de données l_opale_blanche (voir dossier back-end dans la documentation) :

- **Table users** : Stocke les informations des utilisateurs (ex: id, nom, email, mot de passe hashé).
- **Table services** : Contient les services proposés (ex: id, nom, prix).
- **Table reservations :** Gère les réservations (ex: id, user_id, service_id, date).
- **Table slots :** Gère les créneaux de chaque service.
- **Table contact :** Stocke les messages envoyés par les utilisateur du site.

C. La gestion des utilisateurs

L'inscription et la connexion des utilisateurs sont gérées via un formulaire React connecté à l'API PHP.

- Côté frontend, les champs sont gérés avec useState, et les échanges avec le serveur se font via fetch().
- Côté backend, les informations saisies sont validées, puis le mot de passe est haché avec password_hash() avant d'être stocké en base de données.

Un contrôle est effectué pour éviter les doublons (email unique) et garantir la sécurité des données. De plus, la connexion initialise une session, et un système de déconnexion permet de la fermer proprement.
Les erreurs et messages de confirmation sont gérés dynamiquement côté React pour améliorer l'expérience utilisateur.

Le système de rôles permet de distinguer plusieurs types d'utilisateurs :

- **admin** (administrateur),
- **provider_restaurant** et **provider_spa** (gestionnaires des réservations),
- **user** (utilisateur classique).

Dès la connexion, le rôle est stocké en session et accessible via l'API *getUserRole.php*. Il est utilisé côté frontend pour adapter dynamiquement l'interface (par exemple : affichage conditionnel d'un bouton admin ou redirection automatique vers la page Provider). Côté backend, le rôle peut être vérifié pour protéger l'accès aux ressources sensibles, comme les fichiers de suppression, d'administration ou de gestion globale des réservations.

Cette gestion par rôle permet une navigation personnalisée et sécurisée, adaptée aux droits de chaque utilisateur inscrit.

D. La gestion des réservations

Etape 1 : La récupération des services et créneaux disponibles :
Lorsqu'un utilisateur souhaite faire une réservation, le frontend (en React) interroge l'API via des appels HTTP (comme *getServices.php* et *getSlots.php*). Les services sont classés par catégories (Restaurant, Spa, Soins & Massages), et chaque service est associé à une capacité maximale définie dans la base (max_capacity). L'API *getSlots.php* permet de calculer dynamiquement les créneaux encore disponibles, en fonction des services, de la date et de l'heure choisis. Cela garantit que l'utilisateur ne réserve pas un créneau qui est complet.

Etape 2 : La création de la réservation :
Une fois le formulaire complété (et validé côté React avec useState et fetch()), les données sont envoyées à *createReservation.php* via une requête POST au format JSON. Le back-end effectue les contrôles nécessaires (disponibilité du créneau, cohérence des champs) et enregistre la réservation dans la base si tout est valide. Les données sont sécurisées contre les injections SQL via des requêtes préparées.
Exemple :

```
<?php
// Préparation de la requête SQL avec un paramètre à lier
$stmt = $conn->prepare("SELECT * FROM reservations WHERE user_id = ?");

// Liaison du paramètre $user_id (int)
$stmt->bind_param("i", $user_id);

// Exécution de la requête préparée
$stmt->execute();
?>
```

Etape 3 : La récupération des réservations selon le type d'utilisateur :
Chaque utilisateur peut consulter ses propres réservations depuis son espace personnel. Cela se fait via un appel à *getReservations.php* qui filtre les données selon l'user_id stocké en session.

Les providers (restaurant ou spa) disposent d'une interface dédiée (*ProviderReservations.jsx*) où ils peuvent afficher toutes les réservations de sa catégorie de services, par jour et par type de service, grâce à l'API *getReservationsProvider.php*. De son côté, l'administrateur peut accéder à toutes les réservations via *getAllReservations.php*, triées par date et catégorie, avec la possibilité de les supprimer.

E. Le système de réinitialisation du mot de passe

Le fichier *ForgotPassword.jsx* envoie l'email saisi via une requête POST à *forgotPassword.php*, qui vérifie l'existence de l'utilisateur en base. Si l'email est valide, un token sécurisé est généré avec random_bytes, stocké dans une table password_resets avec une date d'expiration, puis renvoyé sous forme de lien. Ce lien redirige vers la page *ResetPassword.jsx*, qui extrait le token de l'URL avec useSearchParams. L'utilisateur saisit alors un nouveau mot de passe, envoyé à *resetPassword.php*. Ce dernier vérifie le token, met à jour le mot de passe haché via password_hash, puis supprime le token de la base pour invalider le lien.

3. Les tests réalisés

A. Les tests unitaires

Des tests unitaires manuels ont été réalisés pour vérifier le bon fonctionnement des fonctionnalités du back-end, notamment la création de réservation et la récupération des données utilisateur.

Environnement de test utilisé :

- WampServer (Apache, MySQL, PHP)
- phpMyAdmin pour la gestion de la base de données
- Navigateur pour tester les requêtes

Les étapes du test de la fonctionnalité de réservation :

1. Récupération des services (getServices.php) : Exécution du script, comparaison des services affichés avec ceux en base de données.
➔ **Résultat attendu :** La liste des services est correcte.

2. Récupération des créneaux disponibles (getSlots.php) : Exécution du script, vérification des créneaux affichés correspondant à ceux en base de données.
➔ **Résultat :** La liste des créneaux disponibles est correcte.

3. Création d'une réservation (createReservation.php) : Exécution du script avec des données valides, vérification de la réponse du script, confirmation de l'insertion de la réservation dans phpMyAdmin.
➔ **Résultat :** La réservation apparaît en base de données.

4. Récupération des informations d'un utilisateur (getUser.php) : Exécution le script avec un identifiant utilisateur, vérification de l'exactitude des informations retournées.
➔ **Résultat :** Les données utilisateur sont exactes.

5. Récupération des réservations de l'utilisateur (getReservations.php) : Exécution du script avec un identifiant utilisateur, vérification des réservations retournées.
➔ **Résultat :** L'utilisateur retrouve ses réservations.

B. Les tests d'intégration

En lien avec les tests unitaires précédents, des tests d'intégration ont été réalisés pour vérifier la communication entre le **front-end (React)** et le **back-end (PHP)** via l'API. Ces échanges sont testés à chaque interaction réelle, telles que la soumission de formulaires.

Environnement de test utilisé :

- Front-end : React (avec axios pour les requêtes API)
- Back-end : PHP sous WampServer
- Base de données : MySQL, gérée via phpMyAdmin
- Observations dans le navigateur : via DevTools > Network et vérifications en base de données.

Exemples de tests d'intégrations :

- **Connexion et inscription (Login.jsx, Signup.jsx, login.php) :** Vérification de l'envoi des identifiants et de la réception du token utilisateur, saisie et soumission des identifiants via le formulaire React (Login.jsx), envoi des données avec axios.post('http://localhost/opale-blanche-api/login.php'), réception de la réponse JSON, stockage du rôle utilisateur et redirection vers l'espace approprié et vérification en base de données que l'utilisateur est bien authentifié.

- **Affichage des réservations pour les différents rôles (Profile.jsx, MyReservations.jsx,AdminReservations.jsx, ProviderReservations.jsx, getReservations.php, getAllReservations.php) :** Vérification de l'affichage conditionnel des réservations selon l'utilisateur, test de connexion en tant qu'utilisateur, admin ou provider, appel à l'API via axios.get("http://localhost/opale-blanche-api/getReservations.php"), affichage conditionnel dans React (si res.data.success renvoie des réservations), vérification en base de données que seules les réservations associées à l'utilisateur sont affichées.

C. Tests de sécurité

Des tests de sécurités manuels ont été réalisés afin de permettre de vérifier que l'application bloque les tentatives d'injection malveillante (XSS, SQL, etc).

Le projet utilise en effet plusieurs protections manuelles :

- **Contre les injections SQL** : Tous les fichiers critiques (getUser.php, createReservation.php, getReservations.php, etc.) utilisent des requêtes préparées (prepare() + bind_param()), ce qui empêche l'injection directe de code SQL via les formulaires.
- **Contre les failles XSS** : Côté frontend React, les champs sont filtrés et contrôlés (ex : pas de champ <textarea> libre pour du code). De plus, les données sont échangées uniquement via JSON, donc non interprétées comme du HTML. Et côté PHP, il n'y a pas d'affichage HTML direct (pas de echo de texte utilisateur dans une page HTML), ce qui réduit fortement le risque XSS.

III. FRAMEWORK

1. Les technologies utilisées

A. Le Framework

React : Il permet de créer une interface dynamique et modulaire grâce à ses composants réutilisables. Son Virtual DOM optimise les mises à jour, améliorant ainsi les performances. De plus, la gestion de l'état avec les Hooks facilite l'organisation du code et l'interactivité des pages.

B. Les outils associés

Vite.js : Il offre un temps de démarrage ultra-rapide grâce à son serveur basé sur ES modules (système de gestion des fichiers JavaScript permettant d'importer et d'exporter des fonctionnalités entre différents fichiers). Son Hot Module Replacement (HMR) permet un rechargement instantané des modifications, accélérant le développement. En production, il génère des bundles optimisés ce qui permet des performances élevées.

Ces choix assurent rapidité, efficacité et facilité de maintenance pour mon site web.

2. Principales fonctionnalités implémentées

A. Affichage dynamique des services

L'affichage des services disponibles est dynamique, permettant ainsi une mise à jour automatique en fonction des données stockées dans la base de données.

✓ **L’utilisation d’une API REST :** Les services sont récupérés depuis une API PHP via des requêtes fetch() en JavaScript, sans nécessiter de rechargement de la page.

✓ **La gestion de l'état avec React Hooks :** useState est utilisé pour stocker et mettre à jour la liste des services et useEffect permet de récupérer les services dès le chargement de la page et lors de la sélection d’une nouvelle catégorie.

✓ **L’affichage réactif :** Les services sont affichés sous forme de cartes dynamiques en utilisant map() pour parcourir les données reçues et les injecter dans des composants React.

Grâce à cette approche, les services sont toujours à jour et s'affichent immédiatement après toute modification dans la base de données.

B. Le routage avec *React Router*

Pour la navigation entre les différentes pages du site, l’application utilise React Router. Contrairement au rechargement traditionnel des pages HTML, React Router permet une navigation sans rechargement, améliorant ainsi l’expérience utilisateur.

✓ **La gestion des routes avec React Router :** Chaque page (accueil, services, calendrier, réservation, compte utilisateur) est définie via <Route>, dans le *main.jsx*.

✓ **L’utilisation de <Link> au lieu de <a> :** Pour éviter le rechargement de la page, les liens de navigation utilisent <Link> plutôt que des balises <a href="...">.

✓ **Les routes dynamiques :** Certaines routes incluent des paramètres, comme l'affichage des détails d’un service (/services/:id).

Avec cette approche, la navigation est rapide et améliore la fluidité du site.

C. L’interativité avec React

✓ **Calendrier interactif en PHP :** Les dates réservées sont stockées en base de données (MySQL) et récupérées via une API pour les afficher dans le calendrier. C’est grâce à une requête AJAX que l’’actualisation du calendrier se fait sans rechargement de la page.

✓ **Gestion des réservations en temps réel :** Lorsqu'un utilisateur sélectionne une date et valide une réservation, celle-ci est envoyée via AJAX à un script PHP, qui met à jour la base de données. Une confirmation instantanée s’affiche sans rechargement de la page.

3. Les tests réalisés

A. Les tests unitaires

Des tests unitaires avec **Vitest** et **React Testing Library** ont été mis en place pour valider le bon fonctionnement de deux composants **Header.jsx** et **ServiceList.jsx** :

- **Header.test.jsx** : Vérifier l'affichage du menu de navigation et si le bouton du menu burger est bien présent dans le DOM. Si le bouton apparaît bien, le test est réussi.
- **ServiceList.test.jsx** : Vérifier l'affichage dynamique des services en fonction de la catégorie. Dans l’exemple, le service “Massage Relaxant” de la catégorie “Spa” est testé. S’il apparaît bien, le test est réussi.

Ces tests assurent que les composants React s'affichent et fonctionnent comme prévu en simulant l’action de l’utilisateur.

B. Les tests fonctionnels

Un test fonctionnel a été mis en place pour vérifier le bon fonctionnement du formulaire de connexion du composant **Login.jsx**. Le test vérifie qu'un utilisateur peut se connecter avec des identifiants valides et qu'une réponse correcte est renvoyée.

Login.test.jsx effectue les étapes suivantes :

- Simulation de la saisie des identifiants : Le test simule l'utilisateur remplissant les champs de saisie pour l'email et le mot de passe.
- Simulation du clic sur le bouton de connexion : L'utilisateur clique sur le bouton pour soumettre le formulaire.
- Vérification de l'appel à l'API : Le test vérifie que l'API de connexion est appelée avec les bons paramètres (email et mot de passe).
- Vérification de la redirection : Une fois la connexion réussie, le test s'assure que la redirection (simulée par navigate()) se produit correctement.

Si l'API répond positivement avec un message de succès pour la connexion et que la redirection vers une autre page s'effectue, alors le test est réussi.

Ce test permet de s'assurer que le composant Login fonctionne correctement en simulant une interaction réelle de l'utilisateur.

C. Le test de performance

Lors du test de performance de l'application avec Vite, des avertissements de performance sont apparus. Ces avertissements indiquent que certains fichiers sont importés à la fois de manière dynamique et statique, ce qui empêche Vite de bien séparer le code en morceaux plus petits(*chunks*). Cela entraîne des fichiers trop lourds, ce qui ralentit le temps de chargement de l'application de réservation.

Voici les problèmes constatés par le test :

- Problèmes de séparation du code : Certaines pages comme **Home.jsx** ou **Reservation.jsx** sont chargées à la fois de manière dynamique et statique, ce qui empêche une séparation correcte.
- Taille des fichiers : Certains fichiers, comme **Login.test.js**, sont trop gros et peuvent ralentir le chargement de l'application. Mais ce fichier est un fichier test et n’est pas intégré sur la version mise en ligne du site.

Pour palier à cela, voici des corrections à apporter :

- **Optimiser la séparation du code :** Configurer Vite pour diviser le code en plus petits morceaux afin de rendre l'application plus rapide.
- **Compresser les images :** avec des outils comme vite-plugin-imagemin pour réduire la taille des fichiers image.
- **Charger le code que si nécessaire :** avec des fonctions comme React.lazy pour ne charger que ce qui est utile, et donc alléger le temps de chargement.

Ces changements permettront de réduire la taille des fichiers et de rendre l'application plus rapide en diminuant le temps d'attente de l’utilisateur.

CONCLUSION

L'application développée offre déjà une expérience fluide et intuitive aux utilisateurs, facilitant l'accès aux différents services proposés. Toutefois, pour aller encore plus loin et assurer une évolution continue du site, plusieurs améliorations peuvent être envisagées afin d'optimiser les performances, enrichir l'interaction avec les clients et renforcer la sécurité des données.

Tout d'abord, en matière de performance, l'optimisation avancée des scripts accélèreraient le chargement des pages du site. (voir tests de performance du framework)

Dans un second temps, l'intégration d'un *chatbot* intelligent ainsi que d'un système de recommandations dynamiques des services les plus populaires pourraient enrichir l'expérience des clients et futurs clients.

Enfin, concernant la cybersécurité, d'autres mesures pourraient être établies pour rendre le site plus sûr comme le chiffrement des données ou encore le renforcement des protections contre les attaques DDoS (cyberattaque visant à saturer le serveur).

Ces évolutions rendraient le site davantage performant et innovant mais aussi plus accessible aux utilisateurs.

CHARTE GRAPHIQUE

La typographie

'SEGOE UI', TAHOMA, GENEVA, VERDANA, SANS-SERIF;

Titres :

Contactez-nous

Textes :

N'hésitez pas à nous contacter via le formulaire

Les couleurs

#242424

#F5F5F5

#214754

MODE CLAIR

#087784

MODE SOMBRE