Basé sur l'analyse de votre code et de l'architecture présentée dans la conversation précédente, voici une explication détaillée du rôle et du fonctionnement de chaque composant dans votre système de création d'hôtel :

🏗️ Vue d'ensemble de l'architecture

Votre application FastBooking utilise une architecture moderne Next.js 15+ avec une séparation claire des responsabilités entre la **présentation**, la **gestion d'état**, les **services**, et les **APIs**.

📄 Rôle de la page create-hotel/page.tsx

**Interface utilisateur principale**

La page create-hotel/page.tsx agit comme le **composant orchestrateur** qui :

typescript

*// Coordonne l'interface utilisateur multi-étapes*

**export** **default** **function** CreateHotelPage() {

**const** {

hotelCardData, addressData, hotelDetailsData,

isStepComplete, getStepErrors, createHotel, isCreating

} = useHotelStore();

*// Gère la navigation entre les étapes de création*

**const** stepStatus = (step: number) => {

**const** complete = isStepComplete(step);

**const** errors = getStepErrors(step);

**return** { status: complete ? "complete" : "error" };

};

}

**Responsabilités clés**

1. **Présentation** : Affiche les formulaires de création par accordéons
2. **Navigation** : Gère les étapes de création (infos de base, adresse, détails, etc.)
3. **Validation** : Vérifie la complétude de chaque étape
4. **Orchestration** : Coordonne les appels au store pour sauvegarder

🏪 Rôle du useHotelStore.ts

**Hub central de gestion d'état**

Le store Zustand sert de **point de contrôle unique** pour toutes les données liées à la création d'hôtel :

typescript

**interface** HotelStore {

*// État des données*

hotelCardData: Partial<HotelCardData>;

addressData: Partial<AddressData>;

roomTypes: RoomTypeData[];

*// État des opérations*

isCreating: boolean;

operationError: string | **null**;

*// Actions de mise à jour*

setHotelCardData: (data: Partial<HotelCardData>) => **void**;

createHotel: () => Promise<{ success: boolean; message: string }>;

}

**Fonctionnalités principales**

1. **Persistance** : Sauvegarde automatique des données pendant la saisie
2. **Validation** : Contrôle de cohérence des données avant envoi
3. **Synchronisation** : Coordination entre les différents composants
4. **Gestion d'erreurs** : Centralisation des erreurs d'opération

🔧 Rôle de services/hotel-service

**Couche de logique métier**

Le service agit comme une **couche d'abstraction** entre le store et les APIs :

typescript

**export** **class** HotelService {

**async** createCompleteHotel(payload: CreateHotelPayload): Promise<ApiResponse> {

**try** {

*// 1. Créer l'adresse*

**const** addressResult = **await** hotelApiService.createAddress(payload.address);

*// 2. Créer les détails*

**const** detailsResult = **await** hotelApiService.createHotelDetails({

...payload.details,

addressId: addressResult.data.id

});

*// 3. Créer la carte hôtel avec relations*

**const** hotelCardResult = **await** hotelApiService.createHotelCard({

...payload.hotelCard,

hotelDetailsId: detailsResult.data.id

});

**return** { success: true, data: { hotelId: hotelCardResult.data.id } };

} **catch** (error) {

**return** { success: false, message: error.message };

}

}

}

**Responsabilités du service**

1. **Orchestration** : Coordonne les appels aux différentes APIs dans le bon ordre
2. **Gestion des relations** : Lie automatiquement les entités (adresse ↔ détails ↔ hôtel)
3. **Validation métier** : Applique les règles business avant envoi
4. **Gestion d'erreurs** : Centralise et standardise la gestion d'erreurs

🌐 Rôle de api/hotel-api

**Couche d'accès aux données**

Cette couche fournit une **interface standardisée** pour communiquer avec vos API routes Next.js :

typescript

**class** HotelApiService {

**private** **async** apiCall<T>(url: string, options: RequestInit): Promise<ApiResponse<T>> {

**try** {

**const** response = **await** fetch(url, {

headers: { 'Content-Type': 'application/json' },

...options,

});

**const** data = **await** response.json();

**return** { success: true, data };

} **catch** (error) {

**return** { success: false, message: error.message };

}

}

**async** createAddress(addressData: any): Promise<ApiResponse<{ id: string }>> {

**return** **this**.apiCall('/api/address', {

method: 'POST',

body: JSON.stringify(addressData),

});

}

}

**Fonctionnalités de l'API**

1. **Abstraction HTTP** : Simplifie les appels fetch avec gestion d'erreurs
2. **Typage sécurisé** : Assure la cohérence des types TypeScript
3. **Standardisation** : Uniforme le format des réponses API
4. **Réutilisabilité** : Méthodes réutilisables pour toutes les entités

🔄 Flux de fonctionnement complet

**1. Interaction utilisateur**

text

Utilisateur remplit HotelBasicInfo[4] →

setHotelCardData() appelé →

Store mis à jour et persisté

**2. Validation et navigation**

text

isStepComplete() vérifie la validité →

getStepErrors() retourne les erreurs →

Interface s'adapte (étape complète/en erreur)

**3. Sauvegarde déclenchée**

text

Clic "Créer l'hôtel" →

createHotel() du store →

validateForSave() vérifie tout →

Appel hotelService.createCompleteHotel()

**4. Orchestration des APIs**

text

Service crée Address → récupère ID →

Service crée HotelDetails avec addressId →

Service crée HotelCard avec hotelDetailsId →

Service crée RoomTypes avec hotelCardId →

Service crée HostDashboard pour gestion

**5. Retour utilisateur**

text

Succès → Store réinitialisé → Redirection vers manage-hotels →

Erreur → Message affiché → Utilisateur peut corriger

🎯 Avantages de cette architecture

**Séparation des responsabilités**

* **Page** : Interface utilisateur pure
* **Store** : Gestion d'état centralisée
* **Service** : Logique métier complexe
* **API** : Accès aux données standardisé

**Maintenabilité**

* Code modulaire et testable
* Évolutions facilitées par couche
* Réutilisabilité des composants

**Expérience utilisateur**

* Sauvegarde progressive des données
* Validation en temps réel
* Gestion d'erreurs claire

Cette architecture respecte les **bonnes pratiques Next.js 15** avec une **gestion d'état moderne Zustand** et une **intégration Prisma optimale** pour votre application de réservation d'hébergements.