Résumé des Design Patterns (Frontend & Backend)

1. Singleton (Créationnel)

Fonctionnement : Garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance.

Frontend (TypeScript) :

```
// services/api.ts
class ApiService {
  private static instance: ApiService;
  private constructor() {}
  public static getInstance(): ApiService {
    if (!ApiService.instance) {
      ApiService.instance = new ApiService();
    }
    return ApiService.instance;
  }
}
// Utilisation
const api = ApiService.getInstance();
```

Backend (Python) :

```
# database.py
class DatabaseConnection:
    _instance = None
    def __new__(cls):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            cls._instance.connection = create_connection()
        return cls._instance
# Utilisation
db = DatabaseConnection()
```

2. Observer (Comportemental)

Fonctionnement : Définit une dépendance un-à-plusieurs pour notifier automatiquement les observateurs des changements.

Frontend (TypeScript) :

```
// context/AuthContext.tsx
const AuthContext = createContext({ user: null, login: () => {}, logout: () => {} });
const UserProfile = () => {
  const { user } = useContext(AuthContext);
  return <div>{user?.name}</div>;
};
```

Backend (Python) :

```
# signals.py
@receiver(post_save, sender=Reservation)
def update_stock(sender, instance, **kwargs):
    plat = instance.plat
```

```
    plat.quantite -= 1
    plat.save()
```

3. MVT - Model-View-Template (Architecture)

Fonctionnement : Variante de MVC. Le framework gère le contrôleur, la vue est divisée en logique (vue) et présentation (template).

Backend (Django) :

```
# models.py
class Plat(models.Model):
    nom = models.CharField(max_length=100)
    prix = models.DecimalField(max_digits=5, decimal_places=2)

# views.py
class PlatListView(APIView):
    def get(self, request):
        plats = Plat.objects.all()
        serializer = PlatSerializer(plats, many=True)
        return Response(serializer.data)

# serializers.py
class PlatSerializer(serializers.ModelSerializer):
    class Meta:
        model = Plat
        fields = ['id', 'nom', 'prix']
```

4. Component (Structurel)

Fonctionnement : Permet de construire des interfaces complexes avec des composants réutilisables.

Frontend (React) :

```
// components/PlatCard.tsx
const PlatCard: React.FC<PlatCardProps> = ({ plat, onSelect }) => (
  <div className="plat-card" onClick={() => onSelect(plat.id)}>
    <h3>{plat.nom}</h3>
    <p>{plat.description}</p>
    <span>{plat.prix}?</span>
  </div>
);
```

5. Repository (Structurel)

Fonctionnement : Fournit une interface entre le domaine et les données.

Frontend (TypeScript) :

```
// repositories/PlatRepository.ts
class PlatRepository {
  static async getPlatsDisponibles(): Promise<Plat[]> {
```

```
    const response = await fetch('/api/plats/disponibles/');
    return response.json();
  }
  static async reserverPlat(platId: number, quantite: number): Promise<Reservation> {
    const response = await fetch('/api/reservations/', {
      method: 'POST',
      body: JSON.stringify({ plat: platId, quantite }),
      headers: { 'Content-Type': 'application/json' }
    });
    return response.json();
  }
}
```

Backend (Python) :

```
# repositories/plat_repository.py
class PlatRepository:
    @staticmethod
    def get_plats_par_categorie(categorie_id):
        return Plat.objects.filter(categorie_id=categorie_id, disponible=True).select_related('categorie')
    @staticmethod
    def get_plats_populaires(limit=5):
        return Plat.objects.annotate(nb_reservations=Count('reservations')).order_by('-nb_reservations')[:limit]
```

6. Factory (Créationnel)

Fonctionnement : Fournit une interface pour créer des objets dans une superclasse, les sous-classes décident de leur type.

Frontend (TypeScript) :

```
// factories/NotificationFactory.ts
class NotificationFactory {
  static create(type: 'success' | 'error', message: string): Notification {
    switch (type) {
      case 'success': return new SuccessNotification(message);
      case 'error': return new ErrorNotification(message);
      default: throw new Error('Type de notification non supporté');
    }
  }
}
```

Backend (Python) :

```
# factories/plat_factory.py
class PlatFactory:
    @staticmethod
    def create_plat(nom, prix, categorie_nom, **kwargs):
        categorie, _ = Categorie.objects.get_or_create(nom=categorie_nom)
        return Plat.objects.create(nom=nom, prix=prix, categorie=categorie, **kwargs)

    @classmethod
```

Résumé des Design Patterns (Frontend & Backend)

```
def create_plat_avec_ingredients(cls, nom, prix, categorie, ingredients):
    plat = cls.create_plat(nom, prix, categorie)
    plat.ingredients.set(ingredients)
    return plat
```