**Documentation du Système de Réservation de Voyages**

**1. Introduction**

**1.1 Objectif du projet**

Le projet de **Système de Réservation de Voyages** a pour but de moderniser la gestion des réservations pour une agence de voyages. L’application permettra :

* La gestion des **utilisateurs** (avec différents rôles : administrateur, client, visiteur).
* La gestion des **activités** (vols, hôtels, circuits touristiques).
* La gestion des **réservations**.

**1.2 Architecture Générale**

Cependant, l’essentiel est de mettre en avant les **concepts Objet** (classes, héritage, etc.).

Exemple d'arborescence de fichiers :

.

├── Admin/

├── assets/

├── Classes/

├── Client/

├── diagrams/

├── db.php

├── index.php

├── login.php

├── logout.php

├── register.php

└── README.md

**2. Introduction à la Programmation Orientée Objet (POO) en PHP**

**2.1 Concepts Clés de la POO**

1. **Classe** : Représente un modèle ou un plan de création d’objets.
2. **Objet** : Instance d’une classe, représentant un élément concret.
3. **Propriétés (ou Attributs)** : Variables définies dans une classe pour décrire l’état de l’objet.
4. **Méthodes** : Fonctions définies dans une classe pour décrire le comportement de l’objet.
5. **2.2 Avantages de la POO**

* **Modularité** : Le code est organisé en classes pour mieux séparer les responsabilités.
* **Réutilisabilité** : Les classes et méthodes peuvent être réutilisées dans différents contextes.
* **Maintenance facilitée** : Les modifications sont plus faciles à réaliser grâce à l’encapsulation.

**3. Encapsulation et Modificateurs d’Accès**

**3.1 Encapsulation**

L’encapsulation consiste à **protéger** les données d’un objet en rendant ses attributs inaccessibles directement depuis l’extérieur de la classe. Cela permet de **contrôler** la façon dont les données sont lues ou modifiées.

**3.2 Modificateurs d’Accès**

* **public** : Accessible à l’intérieur et à l’extérieur de la classe.
* **protected** : Accessible uniquement dans la classe où il est défini et dans les classes héritées.
* **private** : Accessible uniquement dans la classe où il est défini.

**4.Héritage**

**4.1 Héritage**

L’héritage permet de définir une **classe parent** et des **classes enfants** qui héritent de ses propriétés et méthodes. Cela favorise la réutilisation du code et l’extension des fonctionnalités.

**4.2 \_\_construct et \_\_destruct**

* \_\_construct : Appelé automatiquement à la création de l’objet.
* \_\_destruct : Appelé automatiquement à la destruction de l’objet (libération de ressources, etc.).

**4.3 \_\_get et \_\_set**

Permettent de **contrôler** l’accès en lecture et en écriture aux propriétés **inaccessibles** ou non définies.

**5. Mise en place de la Base de Données**

Un **fichier SQL** (par ex. db.sql) pourrait contenir :

CREATE DATABASE IF NOT EXISTS agence\_de\_voyage\_OOP;

USE agence\_de\_voyage\_OOP;

CREATE TABLE IF NOT EXISTS roles (

idRole INT PRIMARY KEY AUTO\_INCREMENT NOT NULL,

nomRole ENUM('superadmin', 'admin', 'client')

);

CREATE TABLE IF NOT EXISTS users (

id\_client INT PRIMARY KEY AUTO\_INCREMENT NOT NULL,

name VARCHAR(100),

email VARCHAR(150) NOT NULL UNIQUE,

password VARCHAR(15),

idRole INT,

FOREIGN KEY (idRole) REFERENCES roles(idRole)

);

CREATE TABLE IF NOT EXISTS reservations (

id\_reservation INT PRIMARY KEY NOT NULL AUTO\_INCREMENT,

id\_client INT,

id\_activite INT,

date\_reservation TIMESTAMP,

nbr\_places int not null,

status ENUM('En\_attente', 'Confirmée', 'Annulée'),

FOREIGN KEY (id\_client) REFERENCES user(id\_client) ON DELETE CASCADE,

FOREIGN KEY (id\_activite) REFERENCES activites(id\_activite) ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS activites (

id\_activite INT PRIMARY KEY NOT NULL AUTO\_INCREMENT,

name VARCHAR(150),

description TEXT,

destination VARCHAR(100),

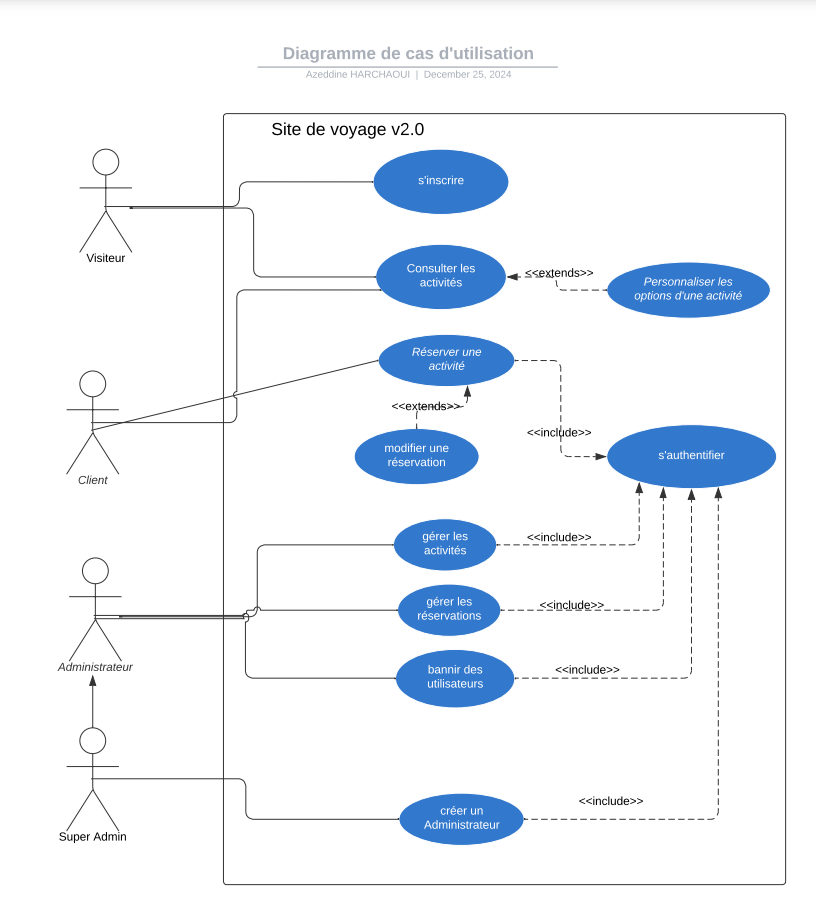
price DECIMAL(10, 2) NOT NULL,

start\_date DATE,

end\_date DATE,

);

1. **Diagrammes UML (à adapter selon votre design)**

**Diagreamme use cas :  
 **

**7. Conclusion**

Le **Système de Réservation de Voyages** proposé ici est un **exemple** pour comprendre et mettre en pratique la POO en PHP.  
Points-clés à retenir :

* L’utilisation de **classes** et de **concepts OOP** (encapsulation, héritage, polymorphisme).
* La bonne gestion des **rôles** et des **droits** (Administrateur, Client, Visiteur).
* La **séparation** claire des **couches** de l’application (MVC).
* L’implémentation des **principes SOLID** pour un code maintenable et évolutif.