|  |
| --- |
| UCAC-ICAM |
| EXPLICATION DES DESIGN PATTERN |
| Groupe 13 |

|  |
| --- |
| Année  2024/2025 |

**I/DESIGN PATTERN MVC**

(Modèle-Vue-Contrôleur) est une architecture logicielle largement utilisée pour organiser et structurer le code dans une application. Ce pattern sépare les préoccupations de l'application en trois composants principaux :

- Modèle (Model) : Gère la logique métier et les données.

- Vue (View) : Affiche l'interface utilisateur et reflète l'état du modèle.

- Contrôleur (Controller) : Interagit avec les utilisateurs et orchestre les actions entre le modèle et la vue.

Pourquoi utiliser MVC ?

1. Séparation des responsabilités : Chaque composant est responsable d'un aspect spécifique, facilitant ainsi la maintenance et l'évolution du code.

2. Réutilisabilité : Les modèles et les vues peuvent être réutilisés dans différentes parties de l'application.

3. Testabilité : La séparation logique permet de tester les composants de manière indépendante.

Description de l’implémentation dans notre projet

1. Le Modèle (Model)

- Responsabilité :

- Gère les données de l'application.

- Définit la logique métier et les règles de gestion.

2. La Vue (View)

- Responsabilité :

- Affiche les informations au client (utilisateur).

- Reçoit des données du contrôleur pour les afficher sous forme compréhensible.

3. Le Contrôleur (Controller)

- Responsabilité :

- Sert d'intermédiaire entre le modèle et la vue.

- Traite les requêtes des utilisateurs, exécute la logique métier via le modèle, et retourne les résultats à la vue.

Flux de données dans MVC

1. Requête utilisateur : L'utilisateur effectue une action (par exemple, en cliquant sur un bouton).

2. Contrôleur :

- Reçoit la requête.

- Exécute la logique nécessaire via le modèle.

- Passe les données récupérées à la vue.

3. Vue: Affiche les données sous un format compréhensible.

Avantages :

- Amélioration de l'organisation du code.

- Collaboration facilitée entre développeurs (front-end et back-end).

- Facilité de test et de débogage.

**II/ DESIGN PATTERN SINGLETION**

Le design pattern Singleton garantit qu'une classe n'a qu'une seule instance et fournit un point d'accès global à cette instance.

Pourquoi utiliser Singleton ?

1. Gestion centralisée : Permet de partager une ressource unique entre différents composants du programme.
2. Optimisation des performances : Évite la création de multiples instances inutiles d'un objet.
3. Simplicité : Fournit un accès facile et contrôlé à une ressource unique.

Description de l’implémentation Singleton dans notre projet

-Une seule instance de connexion à la base de données est partagée et utilisée pour tous les appels dans le projet.

-Cela évite la surcharge de gestion de plusieurs connexions simultanées et assure la cohérence des données.

Son utilisation permet de réduire des ressources consommées et des risques liés à la concurrence ou à la duplication des données.

**III/ DESIGN PATTERN OBSERVER**

Le design pattern Observer établit une relation de dépendance entre un objet sujet (observable) et ses observateurs, de sorte que lorsqu’un état du sujet change, tous les observateurs en sont informés automatiquement.

Pourquoi utiliser Observer ?

1. Réactivité : Permet aux composants d’être informés et mis à jour instantanément lors d'un changement d'état.
2. Découplage : Réduit la dépendance entre les objets en remplaçant les appels directs par des notifications.
3. Flexibilité : Facilite l'ajout ou la suppression d'observateurs sans modifier l'objet sujet.

Description de l’implémentation Observer dans notre projet

-Grâce au mécanisme de signaux et slots de Qt, les modifications des objets graphiques (tables, clients, employés) sont suivies automatiquement.

-Par exemple, lorsqu'un client est déplacé, le signal positionChanged peut être émis, et les slots appropriés de la vue ou du contrôleur reçoivent cette notification pour mettre à jour l'interface.

Il garantit une interface utilisateur synchronisée avec les états des objets en temps réel sans couplage direct entre les modèles et les vues.