Architecture logiciel

Je choisis l’architecture mvc pour mon projet

Pourquoi j’ai choisi l’architecture MVC pour ce projet ?

L’architecture MVC, c’est une façon d’organiser une application en trois parties: le Modèle, la Vue et le Contrôleur. Chacun a son rôle, un peu comme une équipe bien rodée. Voici pourquoi je trouve que c’est le choix parfait pour notre système de gestion de réseau d’entreprise :

Une séparation claire des responsabilités

Ce système, il fait plein de choses : surveiller le réseau, gérer les logs, ajuster les configurations, assurer la sécurité… Avec MVC, tout est bien compartimenté :

- Le **Modèle**, c’est lui qui gère les données – les logs, les configs réseau, les alertes, tout ce qui est essentiel.
- La **Vue**, c’est ce qu’on voit à l’écran – les tableaux de bord, les rapports, l’interface qui donne vie aux infos.
- Le **Contrôleur**, c’est celui qui prend les commandes, qui gère ce qu’on fait, comme modifier une config ou générer un rapport.

Ce découpage, c’est un vrai plus : ça rend le système plus facile à maintenir et à faire évoluer. Par exemple, si on veut changer l’apparence des tableaux de bord (la Vue), pas besoin de toucher à la logique derrière (le Modèle). Tout reste clair et indépendant.

Idéal pour des interfaces complexes

On a des tableaux de bord en temps réel, des rapports, des pages pour gérer les configs et les accès… Ça peut vite devenir compliqué ! Avec MVC, les interfaces (les Vues) sont bien séparées du reste. On peut travailler sur chaque écran de manière modulaire, sans se mélanger les pinceaux avec la logique ou les données.

Parfait pour gérer plusieurs utilisateurs et rôles

Dans ce projet, il y a différents profils : les admins réseau, les ingénieurs, les utilisateurs classiques, chacun avec ses propres droits. Le Contrôleur s’occupe de dire : "Toi, tu peux faire ça, mais pas ça." Et la Vue s’adapte en conséquence – par exemple, un utilisateur réseau ne verra pas les options pour modifier les configs. C’est super pratique pour organiser tout ça.

Un système qui peut grandir et se réutiliser

Si on veut ajouter quelque chose plus tard – une nouvelle méthode d’authentification ou un autre type de rapport –, MVC facilite les choses. On peut intégrer un nouveau Modèle, une nouvelle Vue ou un nouveau Contrôleur sans tout chambouler. Et même mieux : certains morceaux du Modèle, comme la gestion des logs ou des alertes, peuvent être réutilisés pour d’autres projets ou modules.

Plus simple à tester

Comme je l’ai mentionné dans les tests (fonctionnels, performance, sécurité), MVC permet de vérifier chaque partie séparément :

- Le Modèle, pour s'assurer que les données sont bien gérées (par exemple, est-ce que les logs sont correctement stockés ?).
- Le Contrôleur, pour valider la logique (est-ce qu'il détecte bien les anomalies ?).
- La Vue, pour confirmer que tout s'affiche comme il faut.
  Ça rend les tests beaucoup plus fluides et efficaces.

Ça colle avec les outils d'aujourd'hui

Beaucoup de frameworks modernes – comme Spring pour Java, Django pour Python ou ASP.NET MVC pour C# – sont basés sur MVC. Ça simplifie le développement, la maintenance et l'intégration avec d'autres systèmes, comme des bases de données ou des outils d'authentification (OAuth2, LDAP, par exemple), qu'on utilise dans ce projet.

1. Modèle : Gestion des données et de la logique métier

- **Composants du Modèle** :
  - **Entités de données** :
    - NetworkDevice : Représente un appareil réseau (routeur, commutateur) avec des attributs comme l'IP, le statut, les métriques (latence, débit).
    - Alert : Représente une alerte ou notification (type d'alerte, seuil, timestamp).
    - Log : Représente une entrée de journal (timestamp, événement, appareil concerné).
    - Configuration : Représente les paramètres de configuration d'un appareil (par exemple, règles de pare-feu, VLAN).
    - User : Représente un utilisateur avec son rôle (Administrateur réseau, Ingénieur réseau, etc.) et ses permissions.
  - **Logique métier** :
    - Gestion des alertes : Détecter une anomalie (par exemple, un débit réseau dépassant un seuil) et générer une alerte.
    - Analyse des performances : Calculer des métriques comme la latence moyenne ou le taux d'erreur.
    - Stockage des logs : Enregistrer les événements réseau dans une base de données.
    - Gestion des configurations : Synchroniser les paramètres entre les appareils.
    - Authentification : Vérifier les identifiants via OAuth2 ou LDAP.
  - **Sources de données** :
    - Base de données relationnelle (pMySQL ou PostgreSQL) pour stocker les logs, les configurations et les utilisateurs.
    - Systèmes externes comme SNMP pour collecter les métriques des appareils réseau.

2. Vue : Interface utilisateur

- **Composants de la Vue** :
  - **Tableau de bord de supervision** :
    - Affichage en temps réel des métriques réseau (graphiques de latence, débit, etc.).

        - Liste des alertes actives avec des options pour les marquer comme résolues.
    - **Section des rapports** :
        - Interface pour générer et visualiser des rapports (par exemple, un rapport sur les performances réseau sur une semaine).
        - Filtres pour sélectionner une période ou un appareil spécifique.
    - **Gestion des configurations** :
        - Formulaire pour modifier les paramètres d'un appareil (par exemple, ajouter une règle de pare-feu).
        - Liste des configurations actuelles avec un bouton pour synchroniser.
    - **Sécurité et authentification** :
        - Page de connexion avec support pour OAuth2/LDAP.
        - Interface pour gérer les rôles et les accès (par exemple, ajouter un nouvel utilisateur avec un rôle spécifique).
    - **Technologies suggérées** :
        - Framework front-end comme React, Angular ou Vue.js pour une interface dynamique.
        - Bibliothèques de visualisation comme Chart.js ou D3.js pour les graphiques de performance.

3. Contrôleur (Controller) : Gestion des interactions

- **Composants du Contrôleur** :
    - **Supervision du réseau** :
        - MonitorController : Récupère les métriques en temps réel du Modèle et les envoie à la Vue pour affichage.
        - AlertController : Gère les alertes (détection, notification, mise à jour du statut).
    - **Journalisation et rapports** :
        - LogController : Récupère les logs du Modèle et les envoie à la Vue pour analyse.
        - ReportController : Gère la génération de rapports (par exemple, collecte des données sur une période donnée et création d'un PDF).
    - **Gestion des configurations** :
        - ConfigController : Gère les modifications de configuration (validation des paramètres, synchronisation avec les appareils).
    - **Sécurité et authentification** :
        - AuthController : Gère l'authentification (connexion via OAuth2/LDAP, génération de tokens).
        - AccessController : Gère les permissions (vérifie si un utilisateur a le droit d'accéder à une fonctionnalité).
    - **Exemple de flux** :
        - Un Administrateur réseau clique sur "Générer un rapport" dans la Vue.
        - Le ReportController reçoit la requête, demande au Modèle de collecter les données sur la période spécifiée, puis renvoie un rapport à la Vue pour affichage.