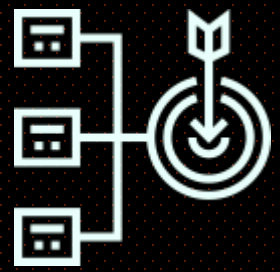
ARCHITECTURE APPLICATIVE

OBJECTIFS DU CHAPITRE

- Vous donner un aperçu d'une architecture logicielle classique
- Se questionner sur que faut-il faire intégralement soi-même, que peut-on créer via un framework ?
- Vous donner quelques éléments de culture générale informatique
- Pour votre projet web : présenter ou rappeler certains points à ceux qui veulent mettre en place une architecture MVC



ARCHITECTURE

ARCHITECTURE 3 TIERS

- Notre contexte : application client-serveur qui manipule des données.

- Tiers : tier = étage

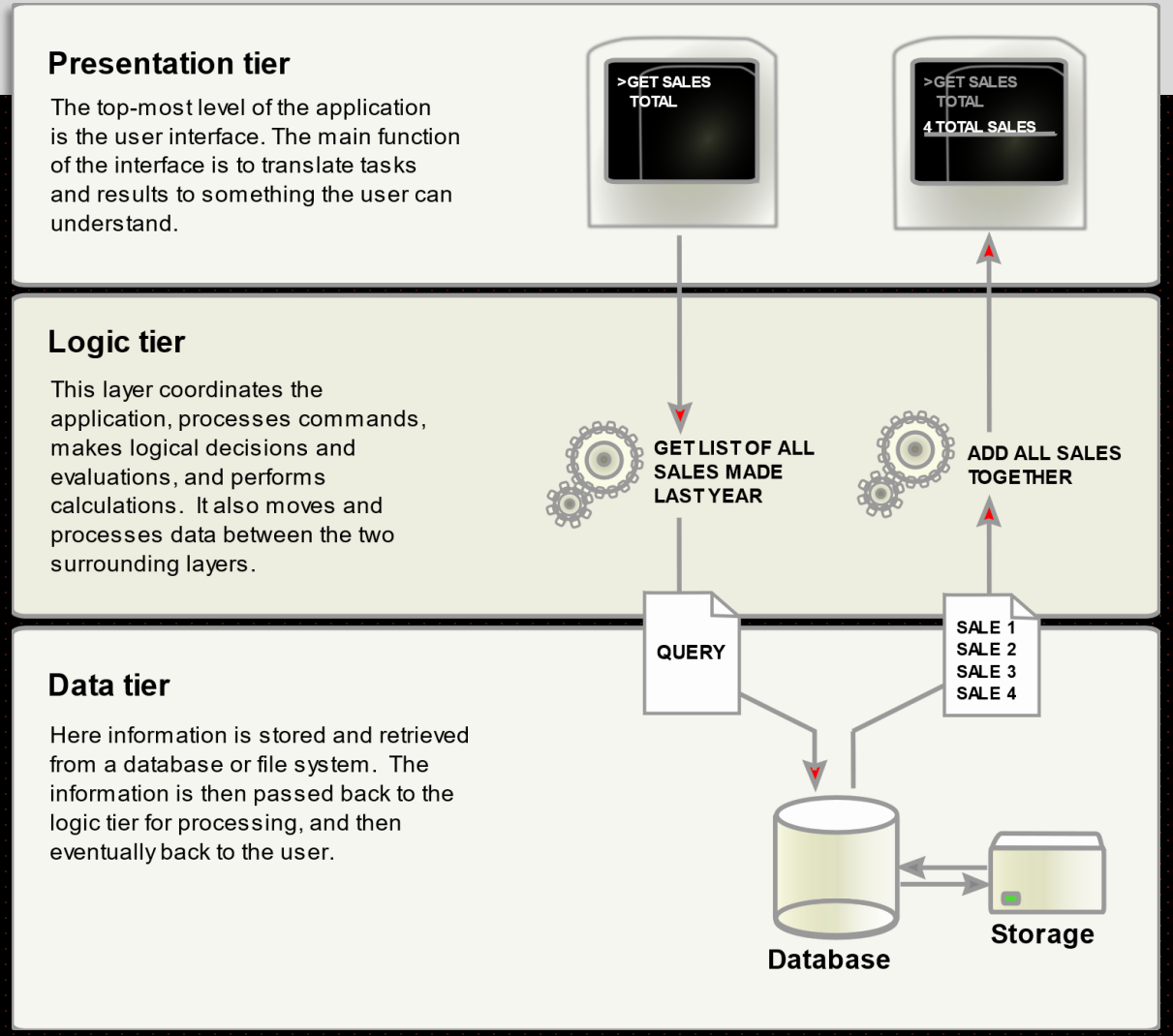
- 3 niveaux :
  - présentation : interface utilisateur (IHM)
  - traitement : intelligence, logique de l'application. Peut accéder aux données via la couche d'accès aux données.
  - données : chargée de la gestion des données persistantes (fichiers, base de données).

ARCHITECTURE 3 TIERS

Schéma wikipedia (version anglaise)

Les 3 couches correspondent à 3 modules distincts, potentiellement physiquement séparés.

Pas de communication directe entre présentation et données.

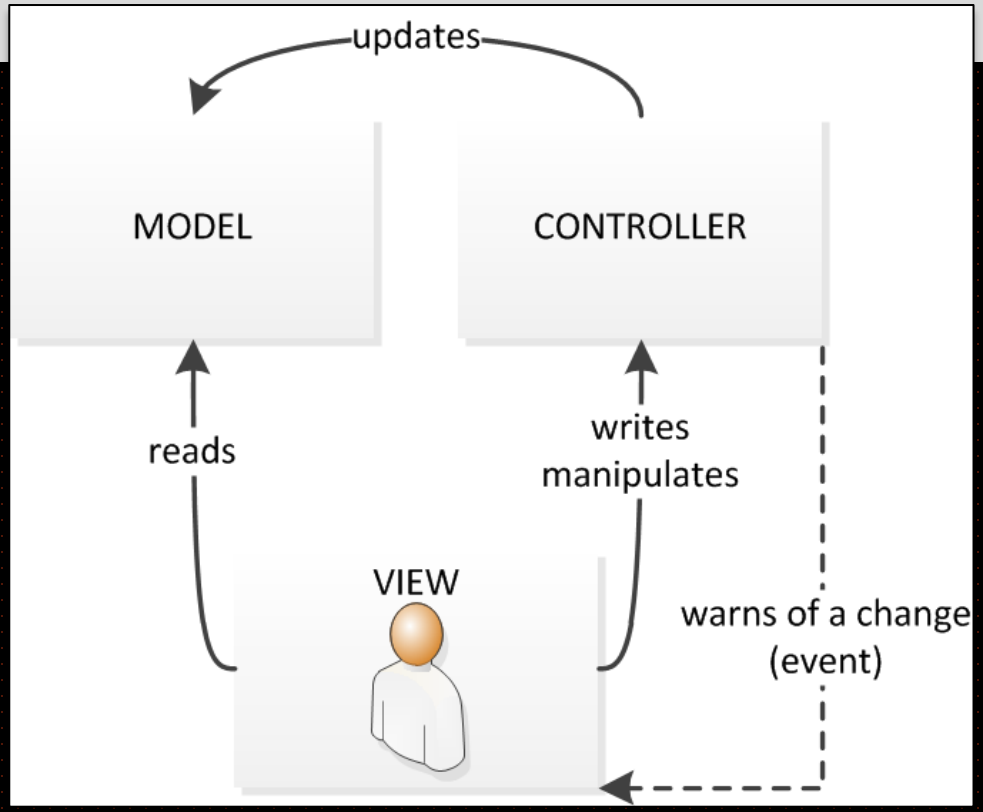


ARCHITECTURE(S) MVC

Une application web écrite en architecture MVC sépare le code en :

- **M**odèles : des classes qui assurent l'accès aux données
- **V**ues : ce qui est présenté à l'utilisateur
- **C**ontrôleurs : l'intelligence, la logique de l'application

ARCHITECTURE(S) MVC



Par Deltacen — Travail personnel, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=23724069

Schéma wikipedia (version française)

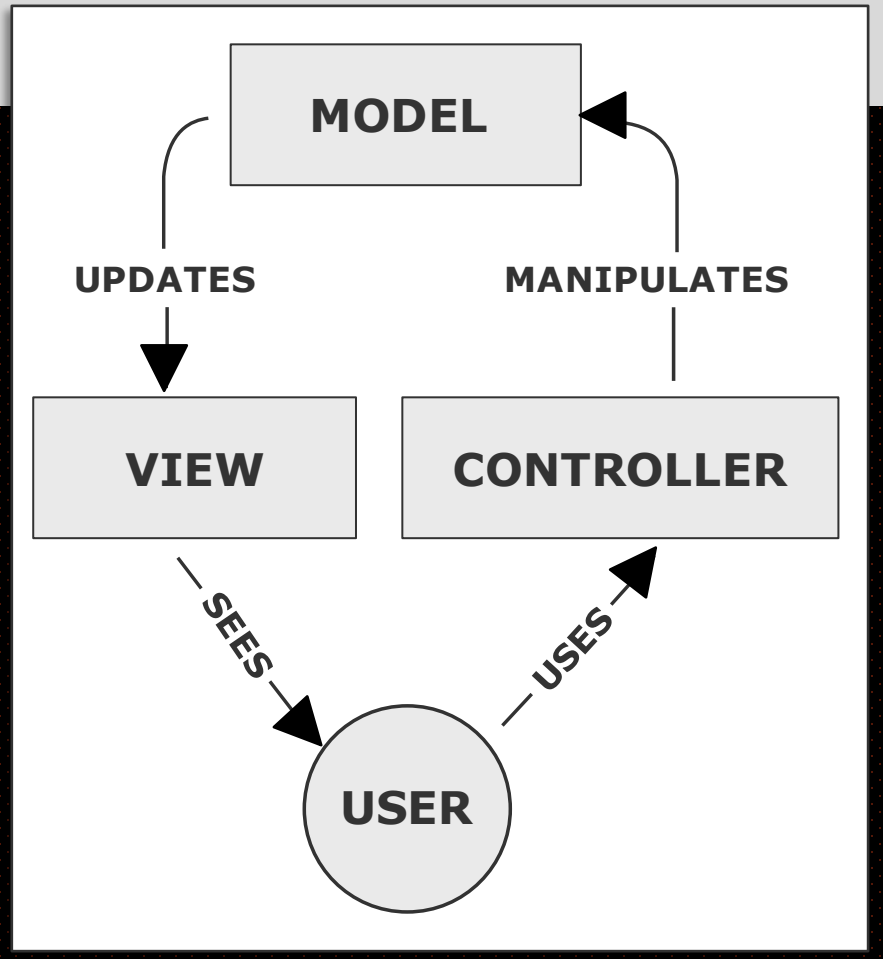


Schéma wikipedia (version anglaise)

ARCHITECTURE(S) MVC

MVC, 3-tiers : rapport ? différences ?

Des concepts et des principes communs. Différentes discussions sur les points communs et différences : au-delà de nos besoins.

Quelques éléments à remarquer :

- communication directe entre présentation et données : pas possible en architecture en couches (3 tiers), « possible » en MVC
- MVC : ne concerne pas la base de données, juste la façon dont elle est accessible depuis l'application

ARCHITECTURE(S) MVC

Une mise en œuvre (implémentation) possible dans notre environnement.

- Modèle : des classes qui assurent l'accès aux données
  - PHP, tout se fait côté serveur
  - classes métiers : représentation volatiles (variables de programmation) des données persistantes (base de données)
  - classes DAO (Data Access Object) : classes qui gèrent l'accès à la base, en masquant sa structure. Par ex : retournent des listes d'instances de classes métiers, pas des curseurs.

ARCHITECTURE(S) MVC

Une mise en œuvre (implémentation) possible dans notre environnement.

- Vues : ce qui est présenté à l'utilisateur, les pages de l'application
  - HTML et CSS comme langages de bases
  - JavaScript si besoin (interactivité, requêtes asynchrones)
  - PHP pour les éléments définis dynamiquement

- Contrôleurs : l'intelligence, la logique de l'application.
  - PHP, uniquement côté serveur
  - Manipulent les données sous la forme de classes métier
  - Accès à la base à travers les classes DAO
  - Déclenchent le rendu des vues
  - Reçoivent les informations envoyées par les vues

ARCHITECTURE(S) MVC

Des points à prendre en compte :

- Sécurité des accès : contrôler « qui accède à quoi »
- Après l'exécution d'un « script » php, aucune donnée n'est conservée (sauf effort particulier de stockage explicite, par exemple en session ou en base de données).
  « Un script » : ici à voir comme un contexte d'exécution, qui peut faire appel à plusieurs fichiers (require).
- Conserver correctement le lien entre les instances de classe et les données en base n'est pas trivial (si on le fait sérieusement…), notamment :
  - ne pas créer deux instances différentes sur la même donnée, sinon pbs potentiels de mises à jour
  - quelles « clés étrangères » suivre, pour disposer des liens pertinents dans les instances sans saturer la mémoire ?

ARCHITECTURE(S) MVC

Une mise en œuvre (implémentation) possible dans notre environnement

- Contrôleur, exemple d'extraction de données :
  - appelé via une URL
  - il va lire des données dans la base, via les classes DAO
  - il transmet les données, sous forme d'instances de classes métiers, à la vue
  - la vue met en forme les données, et est envoyée au visiteur

- Parmi les contrôleurs, un « routeur » :
  - il analyse la requête HTTP reçue puis redirige vers le contrôleur chargé du traitement.
  - donne un point d'entrée unique pour l'exécution de code : facilite la sécurisation

- Une source (très) intéressante : https://bpesquet.developpez.com/tutoriels/php/evoluer-architecture-mvc/

ARCHITECTURE(S) MVC

En pratique :

- On utilise le plus souvent un framework pré-existant (propriétaire ou non)
- Si framework tiers (Laravel, Symfony…) :
  - temps d'apprentissage à prendre en compte
  - étude préalable des fonctionnalités, de la facilité de maintenance, etc…
- Choix de développer un framework propriétaire :
  - peut être lié à des besoins spécifiques
  - investissement en temps important (selon fonctionnalités)

FRAMEWORK
EXEMPLE : SYMFONY

FRAMEWORK

Framework :

- des briques logicielles,
- et un mode d'emploi à suivre pour les employer

Pourquoi utiliser un framework ?

- Ne pas avoir à tout redévelopper
- Bénéficier de composants testés et éprouvés

Pourquoi hésiter ?

- Temps d'apprentissage, investissement
- Pas de maîtrise complète du code (connaissance approfondie des sources, rythme de mise à jour, …)

SYMFONY

Actuellement présenté par les auteurs comme, à la fois :

- un framework PHP pour des développements web
- mais aussi un « simple » ensemble de composants réutilisables

→ deux usages possibles

Ici : présentation rapide de l'aspect framework

Site officiel : https://symfony.com/

SYMFONY

- Eléments de l'application créée :
  - des vues : écrites en HTML, CSS, JS, et un langage dédié appelé TWIG (surcouche de PHP)
  - des contrôleurs : écrits en PHP, avec un socle commun fourni par le framework
  - des classes métier appelées « entités » : en PHP, peuvent être générées en partie automatiquement
  - une communication « transparente » avec la base de données : via des classes PHP fournies par le framework

SYMFONY : DÉMARCHE DE TRAVAIL POSSIBLE

- « Installation » de Symfony et création d'un projet :
  - téléchargement d'un exécutable sur le serveur (accès SSH)
  - utilisation de l'exécutable pour créer un nouveau projet : téléchargement automatique de modules du framework, paramétrable

SYMFONY : DÉMARCHE DE TRAVAIL POSSIBLE

- Conception et création de la base de données : sans Symfony.

- Création des classes métier (« entités ») :
  - Génération automatique des classes avec leurs propriétés en ligne de commande (sur le serveur)
  - Définition manuelle des associations entre classes, avec des annotations (commentaires interprétés)
  - Génération automatique des méthodes d'accès, y compris pour les propriétés de type liste

- Si possible : ne pas avoir à modifier les entités (et la BD) au cours du développement

SYMFONY : DÉMARCHE DE TRAVAIL POSSIBLE

- Développement des vues et des contrôleurs : peut se faire conjointement

- Un contrôleur :
  - peut correspondre à une URL pour le visiteur…
  - … ou bien à une requête asynchrone

- Une vue :
  - est envoyée au visiteur à la demande d'un contrôleur
  - elle met en forme des données transmises par ce contrôleur
  - si elle adresse des requêtes asynchrones : ces requêtes appellent un contrôleur

SYMFONY : DÉMARCHE DE TRAVAIL POSSIBLE

EXEMPLE

SYMFONY : AVIS PERSONNEL, À L'USAGE

- Constats « neutres » :
  - Temps d'apprentissage raisonnable mais non négligeable
  - Mise en place parfois fastidieuse (installation, entités, …), mais à faire une seule fois
- Bénéfices :
  - Code bien organisé, développement et maintenance facilités
  - Outils de débogage efficaces
  - Productivité élevée (par rapport à développement « from scratch »)
  - Modules prédéfinis intéressants (par exemple pour l'authentification)
- Regrets :
  - Documentation : parfois insuffisante
  - Changements de version majeure : gros impact sur les sources

CONCLUSION

FRAMEWORK OU PAS ?

- Raisonnablement : ne pas tout redévelopper
  - Limiter le code à écrire
  - Limiter les sources de bugs

- Exemples de questions à se poser :
  - temps d'apprentissage : investissement rentabilisé ?
  - recréer un framework, y compris avec des modules existants : investissement rentabilisé ?
  - besoin de maîtriser totalement le code produit ?
  - si framework : lequel ?
    - langages mis en oeuvre ?
    - stabilité, qualité du suivi ?
    - spécialisation, domaines préférentiels ?
    - …