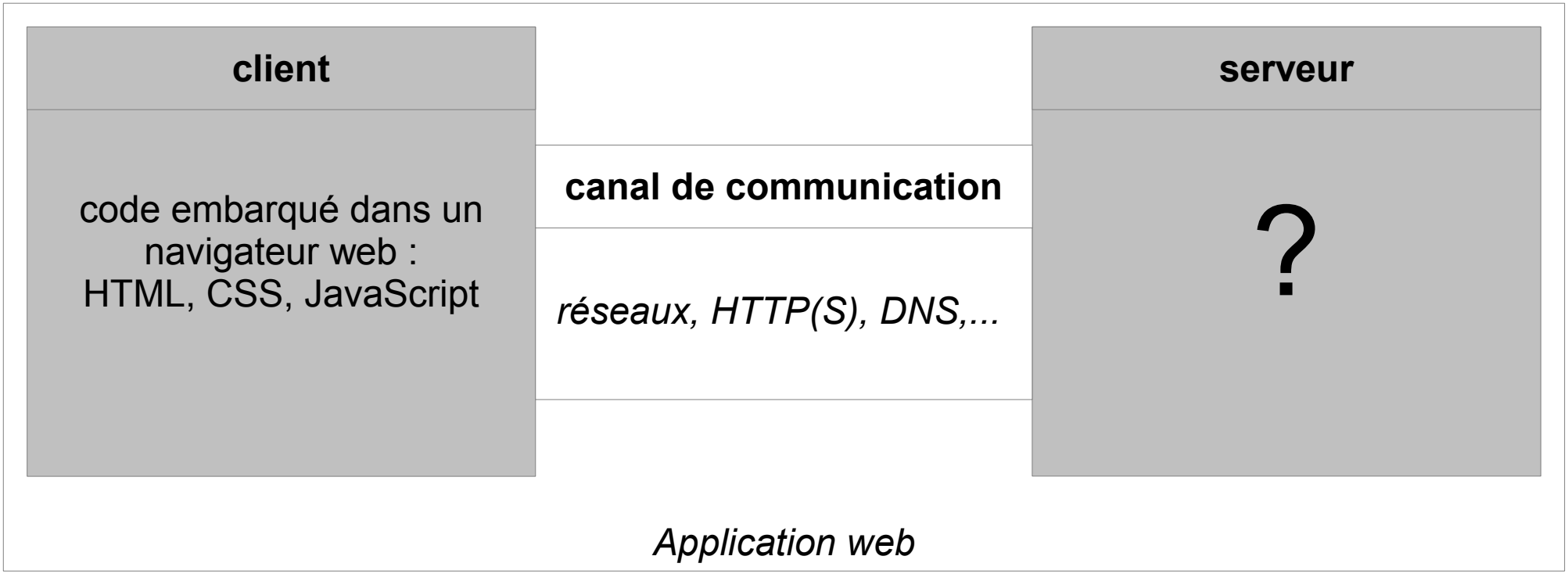
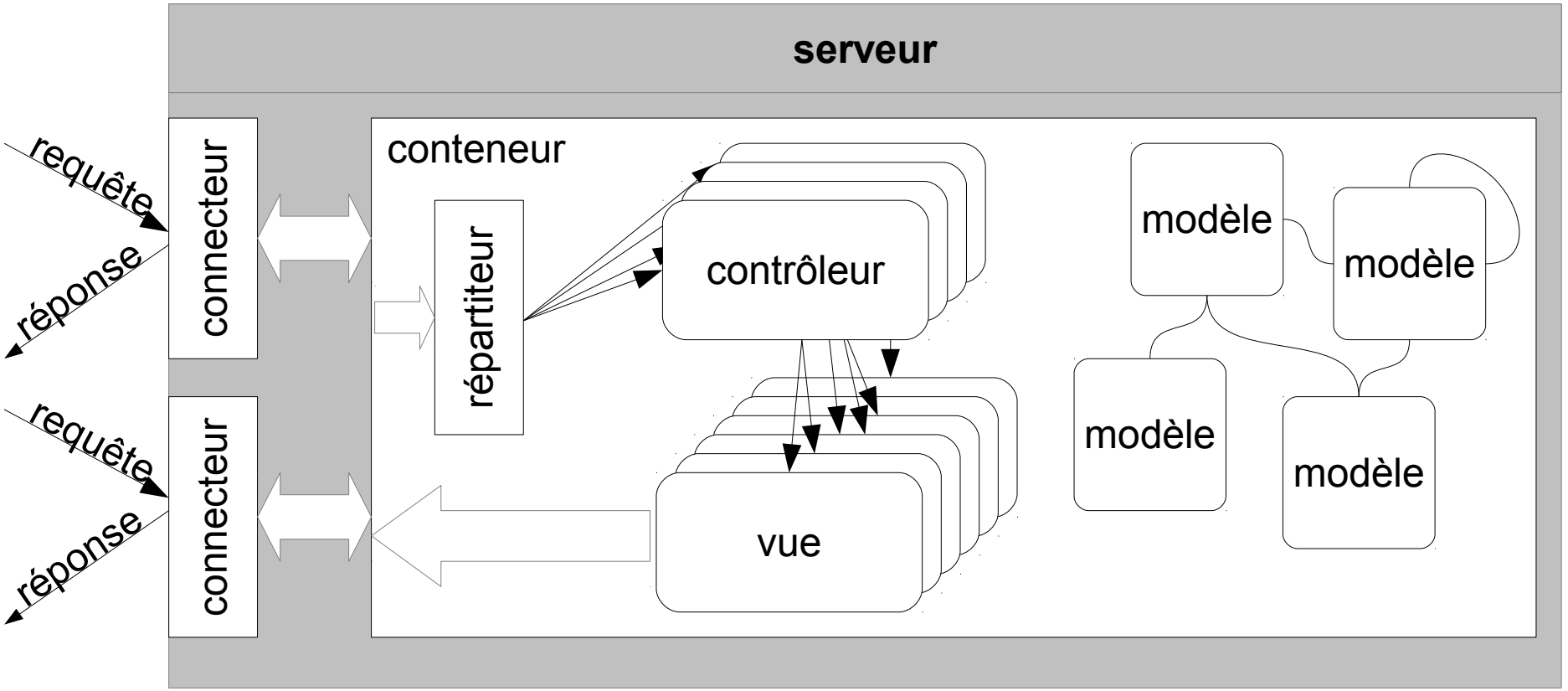
Côté serveur d'une application web



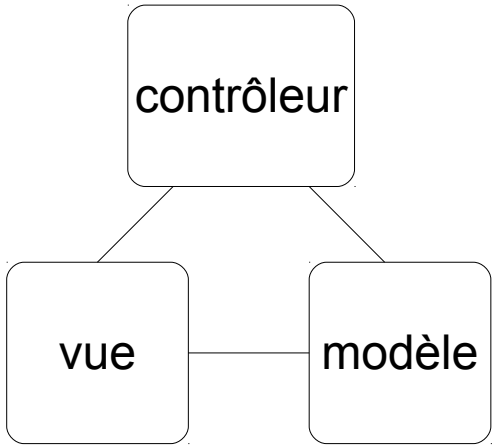
- le côté serveur est programmé avec des langages « classiques » (80% PHP, 15% .Net, 4% Java, Perl, Ruby, Python,...)
- c'est le côté passif (en attente de requêtes), sécurisé (on maîtrise le code et son exécution), centralisé (un serveur pour de nombreux clients)

Architecture d'un serveur



- les *connecteurs* assurent l'interface avec le réseau (HTTP, HTTPS)
- le *répartiteur* achemine les requêtes vers les traitements associés (en fonction du chemin, du contexte, des paramètres,...)

Paradigme Modèle-Vue-Contrôleur (MVC)



Ce paradigme sert à architecturer l'application d'une manière qui découple la logique métier de la présentation et de la logique web (technique).

- Le *modèle* définit les objets/données fonctionnels, la logique métier, les règles et les fonctions qui s'y appliquent
- Les *vues* proposent des représentations des objets du modèle
- Les *contrôleurs* gèrent les données du modèle (lecture, modification) et déterminent la vue à envoyer en réponse à une requête : ils implémentent la logique de l'application web

Exemple MVC - Modèle

- Dans une application web bancaire destinée à fournir aux clients de la banque un accès web aux comptes et aux opérations courantes, le *modèle* définirait
  - des types (classes) de données tels que Client, Compte, Opération, Solde [=données métier]
  - des types de données Utilisateur, Agent, Message [=données informatiques nécessaires à l'application]
  - des relations entre les données (p.ex. Client $\overset{0..n}{\underset{1..n}{\text{———}}}$ Compte ou Utilisateur $\overset{0..n}{\underset{0..n}{\text{———}}}$ Client ou encore Compte $\overset{1}{\underset{1}{\text{———}}}$ Solde)
  - des règles métier (p.ex. « si le solde tombe en-dessous du seuil prédéfini pour le compte associé, alors créer un message d'alerte » ou « si opération mène à un solde négatif, alors refuser »)

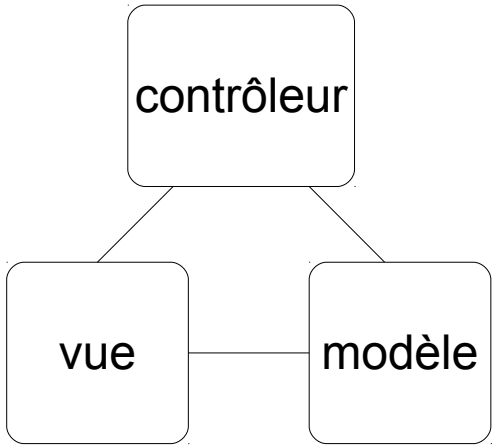
Exemple MVC - Vue

- Dans la même application bancaire, les *vues* proposeraient des représentations des objets du modèle, p.ex.
  - une synthèse client (tableau des soldes des comptes du client)
  - des relevés d'opérations par compte (listes d'opérations ou relevés en PDF)
  - possibilité d'un virement entre les comptes du client (=vue de « nouvelle opération »)
  - ...

Exemple MVC - Contrôleur

- Enfin, les *contrôleurs* assureraient que 1) la logique de l'application soit respectée, p.ex que
  - l'utilisateur soit correctement authentifié avant d'accéder aux données
  - un client n'accède qu'aux comptes dont il est titulaire
  - les opérations sur les comptes respectent la logique métier (p.ex. pas de virement depuis un PEL)
  - les vues présentées soient celles demandées par l'utilisateur
- et 2) les données du modèle soient correctement manipulées, p.ex
  - un virement entre deux comptes débite un montant d'un côté et crédite le même montant de l'autre côté,...

Métiers dans MVC



Le découpage de l'application selon MVC permet également d'identifier des parties qui font appel à différentes compétences, voire à différents métiers.

- Le *modèle* relève des domaines :
  - expert métier (p.ex. conseiller bancaire)
  - architecte des données (comment modéliser?)
  - urbaniste (comment s'inscrire dans l'existant)
- Les *vues* font appel aux compétences d'infographie et d'ergonomie
- Les *contrôleurs* nécessitent des compétences en développement web et en sécurité

MVC – Les Vues

- Les vues peuvent être implémentées entièrement sur le serveur : le serveur renvoie au client des documents HTML, CSS,... On parle de client « thin ».
- Dans cette approche, il y a souvent un langage de « templates » qui permet de décrire les vues dans un mélange de HTML et de structures programmatiques (p.ex. JSP).

- Ou alors, le serveur renvoie au client des données brutes (p.ex. en JSON ou en XML) et c'est le client qui crée les vues grâce au JavaScript (client riche, p.ex. GWT).

- La troisième voie consiste à mélanger les deux approches : le serveur fournit les « grandes » vues in extenso, et le client s'occupe de faire de petits ajustements au fil des actions de l'utilisateur.

MVC – Les tests

- Le *modèle* est généralement implémenté par des techniques de programmation classiques (classes – objets – méthodes – champs) qui permettent des tests unitaires classiques (p.ex. JUnit)
- Les *vues* nécessitent des techniques de test spécialisées pour tester le rendu final : vérifier la présence de tel ou tel élément visuel, vérifier l'apparition d'un nouvel élément après un clic,... (p.ex. Selenium)
- Les *contrôleurs* sont testés comme le modèle, avec cependant des wrappers spécialisés qui permettent de simuler des requêtes et de décoder des réponses