Importance d’une modélisation

Il est important avant de développer d’avoir une bonne modélisation de son application.
Parfois il est nécessaire de créer son protocole.

Il existe plusieurs modèles permettant le fonctionnement de l’application :

- Modèle de communication
- Modèle d’échange
- Modèle d'exécution

Considérer les différents acteurs et les modéliser

- Capteurs
- Serveurs WEB
- Comportement Humain
- ...

Il faut également considérer les échanges entre les acteurs.

Sécurité

Il faut se poser la question : Que puis je transférer, et quoi ne PAS transférer ?

Au niveau des échanges on va avoir des flux de données (site marchand → banque → client → ...).
Pour une confirmation de paiement par exemple on choisit de multiplier les modèles (mail, SMS, ...) afin de contrer les imprévus (coupure de courant, etc).

Échanges

Deux types d’échanges :

- Directs
- Indirects

Échanges directs

Connexion directe entre les acteurs, intermédiaire nécessaire.

Échanges indirects

Addition d'un troisième acteur :
Connaissance mutuelle obligatoire des deux acteurs (logique de centralisation, concentration). Par exemple un serveur discord fonctionne en point de concentration.
Il suffit de se connecter au serveur central de discord (point de concentration).

Donc soit connexion directe, ou indirecte (utilisation d'un intermédiaire).

Séquençage d'échange

Échanges simples

Notification météo par exemple, on envoie une information, mais on attends pas (forcément) de réponse.
On peut ajouter un découpage en ajoutant une fonctionnalité d'accusé de réception (mais pas obligatoire)

Échange enchaînés

Par exemple requête à un annuaire (question- réponse).
Annuaire, base de données, …
Il est nécessaire en cas de problème d'envoyer un message d'erreur. (Navigateur, si pas de réponse, message erreur).

Importance de la signalisation dans une application

Exemple : HTTP et les codes erreur.

Échanges connectés (TCP)

Controle de présence : il faut que les parties soient là, sinon pas de communication possible.
Établissement d'un canal d'échange de données.
TCP est plein de sécurités, c'est un protocole **fiable**

Échanges non connectés (UDP)

UDP est **non-fiable**.
Le principe : Communication légère, pas de controle de présence, on ne sait pas si les données sont réceptionnées. AUCUNE INFO.
Il est possible de programmer des sécurités pour UDP, mais il en revient à la charge du développeur (ajouter un controle de présence par exemple).

UDP n’a pas de vérification de contenu. Indépendant des langages et des systèmes (comme TCP).

Exécution

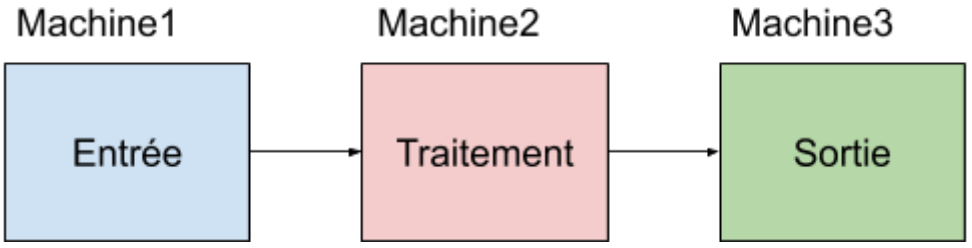
Une exécution est une opération qu’on est capable d’effectuer sur des données (une entrée et une sortie).

Exécution locale

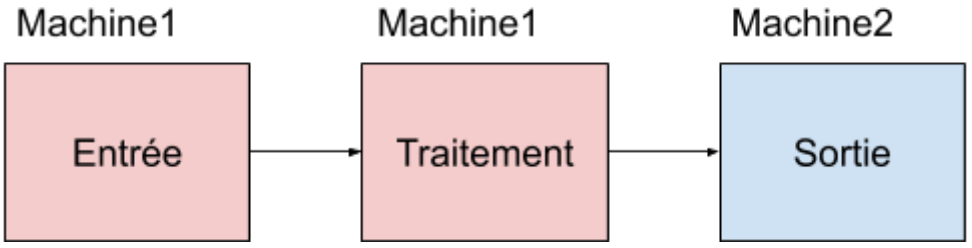
Traitement des données localisées sur le même acteur (entrée sortie sur le même acteur). C’est limité aux ressources locales.

Exécution répartie

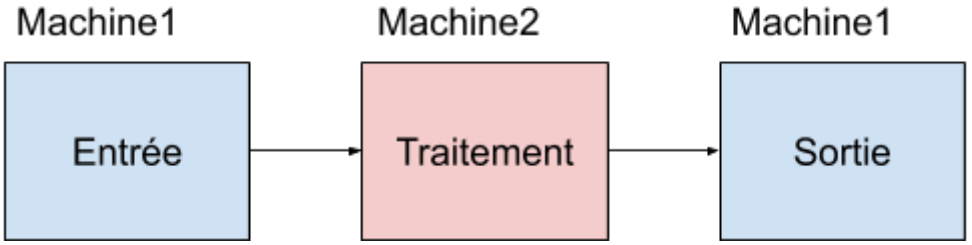
La mise en place de traitement et des données sur les différents acteurs, on va pouvoir découper et les dispatcher sur les différents acteurs (par exemple : entrée sur un machine, exécution sur l’autre, sortie sur une troisième). Ça peut-être aussi des threads.



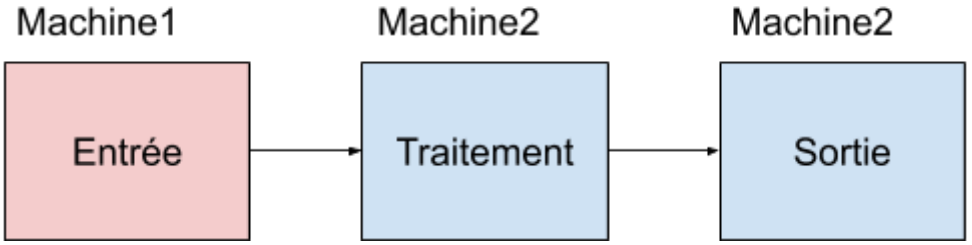
Deux acteurs :



1. Donnée source + traitement sur Machine 1
2. Donnée résultat envoyée sur Machine 2



1. Ici Donnée source envoyée sur la M2.
2. M2 traite la donnée puis la renvoie sur M1.



1. Entrée sur M1, envoi à M2
2. Traitement puis Sortie sur M2

Pourquoi des exécutions réparties ?

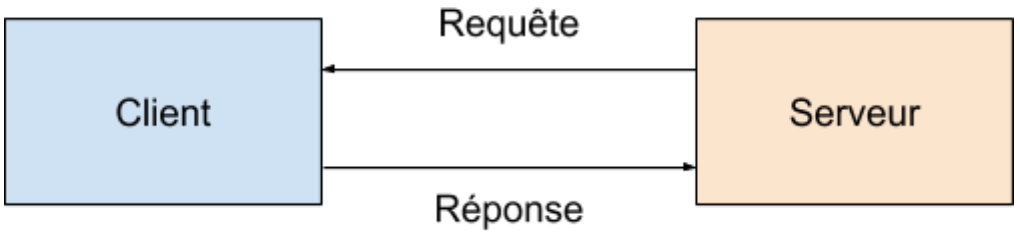
Profiter de la puissance de la machine distante (bruteforce de hash en ligne par exemple).

Modèle à message

Basé sur l'échange d'un message.
Sources émises par un acteur et traitement et donnée sortie sur un autre acteur.
Le travail de l'émetteur est d'envoyer la donnée source, le reste n'est pas de son ressort (échec de compilation, ...). Il doit simplement s'assurer que la donnée est arrivée au destinataire.

Les réseaux sociaux sont basés sur le modèle à message.
Enregistrement de logs sur machine distante : Modèle à message également. (Porte à badge électronique par exemple).

Modèle Client / Serveur



NTP : Network Time Protocol, protocole permettant de se caler sur l'heure d'un serveur distant à la microseconde près.

Un des objectifs du protocole est de normaliser les données (s'assurer que tout soit compréhensible, du genre "ß" il faut le faire comprendre à la machine).
Le protocole embarque un grand nombre de services ayant différents buts.

Il faut aussi prévenir le client de ce qu'il reçoit et adopter un comportement en fonction de la donnée envoyée (ouvrir un pdf avec un lecteur de pdf par exemple, ou simplement sauvegarder le fichier si le serveur ne connaît pas le format). Ceci est aussi une des missions du protocole.

Étude de cas : Application de tracking sportif

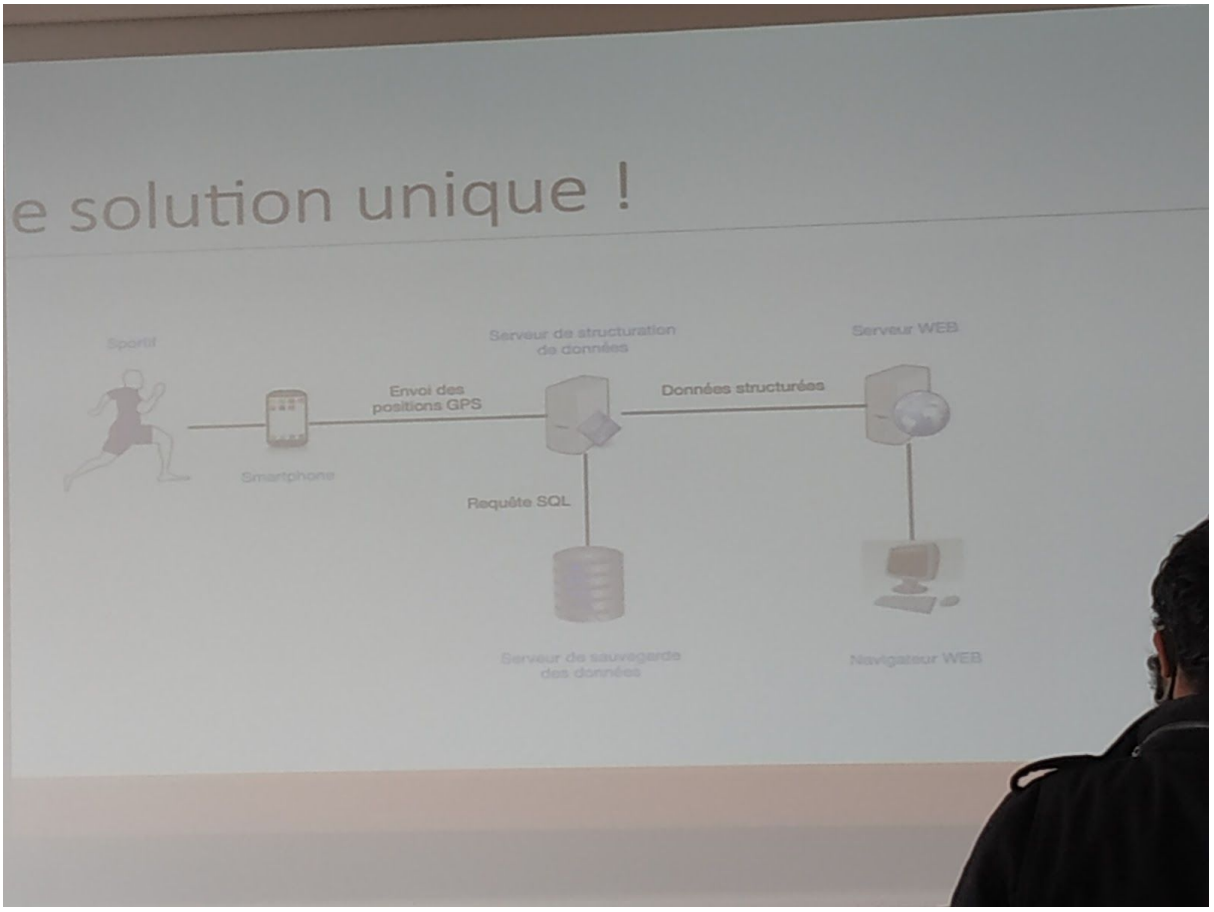
Idée principale : Vous suivre et avoir un suivi de l'activité sportive (coordonnées GPS, capteurs ,etc…). Tout au long de l'activité, des données sont envoyées au serveur.
Une fois l'activité terminé on a un bilan.

Acteur humain : Le sportif
Acteur numérique : Smartphone
Acteur numérique : Machine de sauvegarde des activités
Acteur numérique : Machine traçant les courbes et réalisant les calculs

Le smartphone ne peut pas effectuer tous les calculs (batteries, soucis de performance...)

Première phase : Le téléphone envoie des données au terminal de sauvegarde (basée sur modèle de message), le smartphone va envoyer des coordonnées GPS de façon périodique (toutes les 30 secondes pour de la course à pied par exemple). Quand le serveur de sauvegarde aura fini de recevoir les position GPS, il clôture la connexion avec le smartphone.

Phase 2 : Modèle client/serveur.


e solution unique !
Sportif
Smartphone
Envoi des positions GPS
Serveur de structuration de données
Requête SQL
Serveur de sauvegarde des données
Données structurées
Serveur WEB
Navigateur WEB