#### Infos et Rôles

* **Date**: 24/02/2025
* **Heure de début : 11h10**
* **Durée : 38 min**
* **Promo :** A3
* **Tuteur** : Mohammed
* **Animateur** : Martin.A
* **Secrétaire** : Louis
* **Scribe** : Martin.W
* **Gestionnaire** : Ludovic

# Clarifier la situation

**Mots clés :**

* DMZ
* Docker/Conteneur
* Mécanisme
* Etat d’avancement
* Unité de traitement
* Gestion des ports IP
* DotNet
* Enregistrement de données
* Outils de simulation d’avancement
* Hébergement d’application

# Analyser le besoin

**Contexte :** Franck doit transmettre en temps réel via un mécanisme d’envoie concernant l’état d’avancement de la fabrication à son client, tout en hébergeant son application dans un conteneur Docker sous .NET.

**Problématique :** Comment assurer cette transmission en respectant les contraintes d’hébergement et de gestion des données ?

**Contraintes :**

* Code
* Mécanisme d’envoi (distance)
* Problèmes liés au nouvel environnement
* Transmission en temps réel

**Livrable :** Code en C#, Conteneur en DotNet (Docker)

# Généralisation

**Généralisation** : Hébergement et transmission de données en temps réel.

# Pistes de solution

* Mise en place d’un VPN (OpenVPN)
  + Firewall / Proxy
* Framework -> gère les skockets
* Protocole de communication
* Utilisation des données via une API
  + Héberger sur Docker
* CI/CD pour l’app ?
* Persistance des données (Volume Docker/BDD)
* Conteneurisation DotNet différente de Docker
* Encrypter via Manchester
  + Certif SSL
* Utilisation de WebSocket
* BDD en temps réel

# Plan D’action

1. Sélectionner la technologie de transmission de données
2. Définir les composants du sytème
3. BDD
4. Configurer Docker + DMZ + CI/CD + Sécurité (SSL, …)
5. Code (Back+Front) + implémenter les Web Sockets
6. Solution de résilience avec un CRC
7. Test de charge

Question Tuteur :

* Quelle architecture proposez-vous ?
* Quelles utilisations pour le modèle Client / Serveur ?
* Quels sont les modes de connexion ?
* Quel type de serveur choisir ?
* Quel sera le protocole de communication entre les clients et le serveur ?
* Quelles informations utiliser pour identifier un hôte ?
* Quelle structure utilise-t-on pour envoyer les données ?
* Quel serait le rôle du programme fourni ?
* Combien de programmes allez-vous développez ?
* Quel outil permet d'actualiser l'affichage périodiquement ?
* Quelle est la structure d'un socket ?
* Quelles caractéristiques pour un socket ?
* Comment créer une image Docker ?
* La gestion de l'arborescence en Docker, est-elle la même que pour Windows ?

COURS :

Simon :

UDP et TCP :

* TCP (Transmission Control Protocol) → fiable, orienté connexion.
* UDP (User Datagram Protocol) → non fiable, sans connexion.

Martin.A :

Technologie de transmission de données.

* Poolling : Rest API

Rest API : Utiliser dans le web, REST -> protocole utiliser pour le client-serveur.

D’autres protocoles : SOAP, GraphQL)

Lucas :

Architecture client-serveru.

Client : Navigateur web etc

Deux types :

* Léger : application web etc
* Lourd : logiciel bureautique, …

Différenciation :

* Différent framework
* Nombre de processus
* Plusieurs client différent (Pool de connexion plus gros)

Serveur : BDD, logiciel métier, ...

Réseau : les principaux protocoles utilisé TCP/IP, http.

Mathéo :

Sécurite :

* SSL/TLS
* Firewall/DMZ

Port dynamique :

* Résilience CRC
* HCP
* Gestion d’erreur

Vincent :

BDD :

2 types :

* SQL (PostgreSQL)
  + Application
  + Schéma définis
* No SQL
  + Flexible
  + Docker Constructeur
  + Lien entre différent objet (Réseaux Sociaux)

Serveur less : JSON, utilisé dans le NoSQL

Clément :

WebSocket : Idéal pour une communication bidirectionnelle temps réel entre le serveur et le client. Evite l’accumulation des requêtes http répétitive.

Simon :

WebSocket : Semi duplex, Async et await. Plus simple et permet de gérer plusieurs connexions.

Jérémy :

Cycle de vie d’un socket :

1. Création du socket (client et serveur).
2. Bind (côté serveur) : association du socket à une adresse IP et un port.
3. Écoute (serveur) : attente des connexions clients.
4. Connexion (client) : demande de connexion au serveur.
5. Échange de données (lecture/écriture sur le socket).
6. Fermeture (libération des ressources).

Ouverture d’un port qui n’est pas dans l’intervalle de 1 à 1024 (Simon)

DORA (Discover, Offer, Request et Acknowledge) (Simon)

Martin.A, Simon

Test de charge :

* Simuler un nombre important de connexions simultanées.
* Mesurer la latence des mises à jour.
* Optimiser la gestion de la mémoire et la scalabilité du système.
* Effectuer des tests de sécurité pour éviter toute faille exploitable.

JMeter/Elocus/K6

Lucas :

Backend (serveur)

Front (interface utilisateur)

BDD (base de données)

SSL : permet d’ajouter une couche d’authentification/certification, wildcard.

Thomas :

Azure KeyVault

CI/CD : reprendre nos tests unitaires. Automatisation des processus (Build, push, …)