PresentBOX

Document d'Architecture du Logiciel

Version <1.0>

Historique des révisions

Date	Version	Description	Auteur
22/05/2022	1.0	1ére version de l'architecture logiciel	Boudouche Hamza

Table des matières

Document d'Architecture du Logiciel

1. Introduction

Ce document a pour but de présenter le logiciel traité "Gestion de classe" ainsi que les besoins fonctionnels et non-fonctionnels auxquels il répond, et de discuter en détail l'architecture choisie pour la réalisation de ses différents sous-systèmes.

2. Objectif du logiciel

2.1 Contexte

Le logiciel sujet de notre étude est une application de "Gestion de classe". C'est une application desktop cross-plateformes chargée de tout ce qui concerne les aspects de surveillance, contrôle, collaboration et évaluation d'une ou plusieurs classes. En effet, il permet la gestion des appareils électroniques utilisées par les étudiants de la classe connectés à la plateforme, surveiller les étudiants à travers leurs webcams, les évaluer par des quizs, et partager de l'écran d'un étudiant ou d'un tuteur avec les autres étudiants, ainsi que des fonctionnalités supplémentaires telles que la communication par chat, le partage d'un tableau blanc virtuel et le regroupement d'étudiants en groupes.

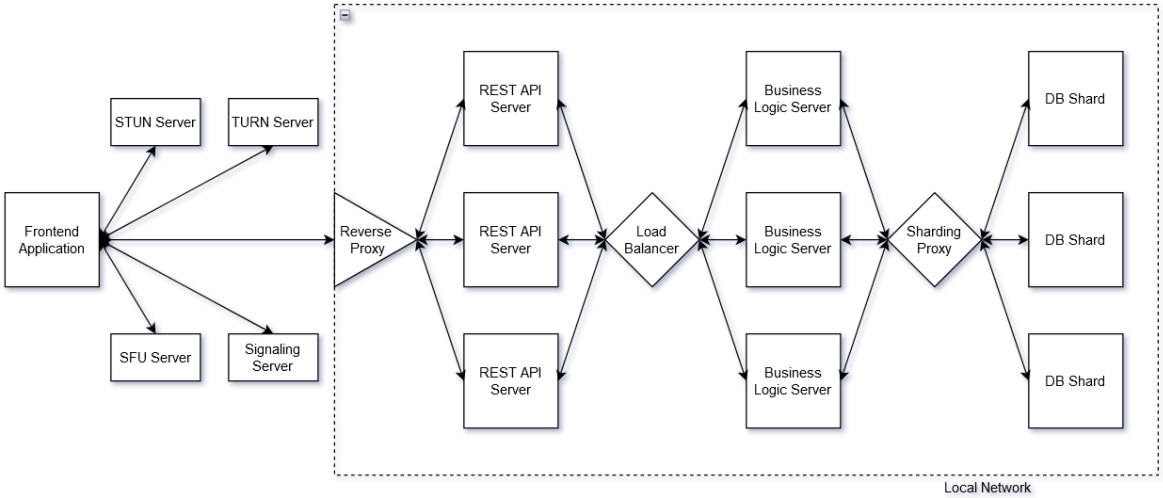
2.2 Besoins fonctionnels (du sous-système de présentation)

- Partage de l'écran du tuteur et des étudiants
- Diffusion de fichier video
- Communication par chat entre étudiants et tuteurs
- Organisation des étudiants en groupes
- Partage et édition de tableau blanc virtuel

2.3 Besoins non fonctionnels (du sous-système de présentation)

- Fiabilité de la connexion
- Scalabilité
- Facilité d'utilisation
- Sécurité et confidentialité des données personnelles
- Portabilité entre différents systèmes d'exploitation

3. Structure



C'est une architecture de type n-tiers (ici 3 tiers) qui se compose des couches suivantes:

- Couche web
- Couche logique métier
- Couche données

Cette architecture répond aux exigences de fiabilité et de scalabilité, car elle découple les composants qui sont responsables des différentes opérations au sein du système en moyennant un nombre de balanceurs de charge (load balancers) et permet aussi une redondance des services et augmente ainsi la robustesse et la fiabilité du système.

Une alternative à cette architecture est l'architecture monolithique, qui est bien plus facile à architecturer et à mettre en œuvre, mais qui se caractérise par un couplage fort des composants de l'application et une fiabilité réduite. Cette option est donc moins adaptée à notre situation.

3.1 Vue des couches

Comme expliqué précédemment, l'architecture proposée présente 3 couches :

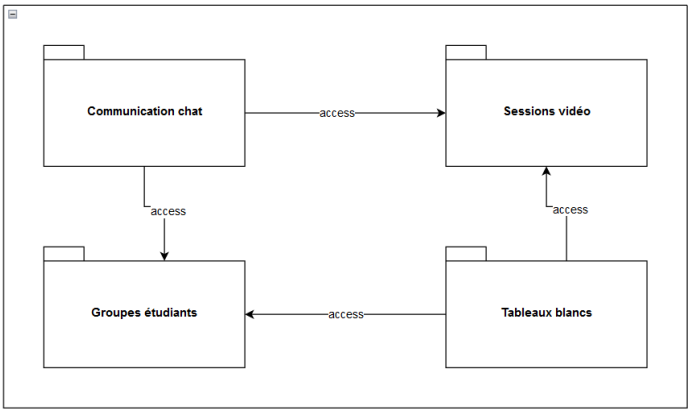
- Une couche web, qui est la première couche recevant du trafic web de la part frontend de l'application, et qui contient les endpoints d'une API de type REST. Ce choix est motivé par la robustesse des principes REST et surtout par son caractère stateless, qui a permis de mettre en œuvre une multitude de serveurs similaires dans la même couche pour rendre le système plus scalable.
- Une couche logique métier, qui contient plusieurs serveurs recevant du trafic distribué par un load balancer provenant des serveurs REST, et dont le but est de manipuler ce trafic, et communiquer avec la couche des données, pour répondre aux besoins métier de l'application.

- Une couche données, qui contient plusieurs partitions (shards) de la base de données du système. Les partitions y sont contrôlées par un proxy de partitions (sharding proxy). Ce choix de partitionnement de base de données est motivé par le besoin de scalabilité de l'application, qui ne serait pas réalisé si l'application était basée sur une seule instance de base de données.

3.2 Sous-systèmes et paquetages

Nous allons analyser chaque couche de l'architecture proposée en spécifiant les paquetages relatives à chaque sous système de la plateforme "Gestion de classe":

- Couche Web :
- Couche logique Métier :



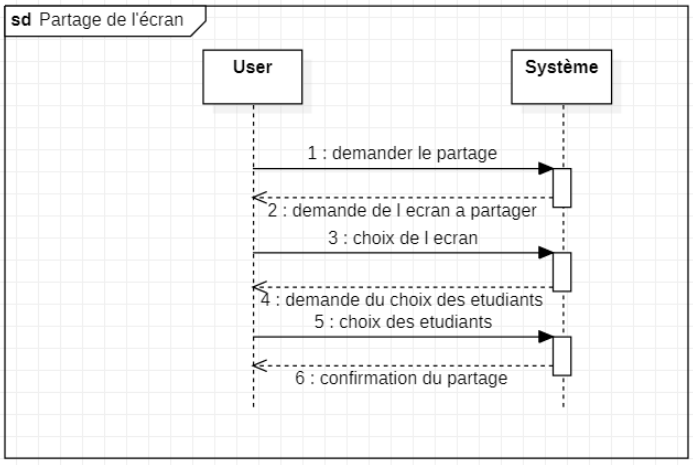
- Couche Données

3.3 Interfaces

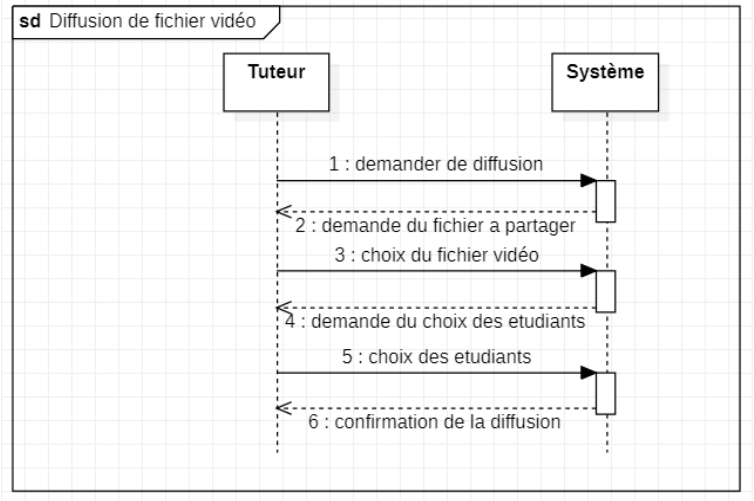
4. Comportement

4.1 Réalisation des cas d'utilisation

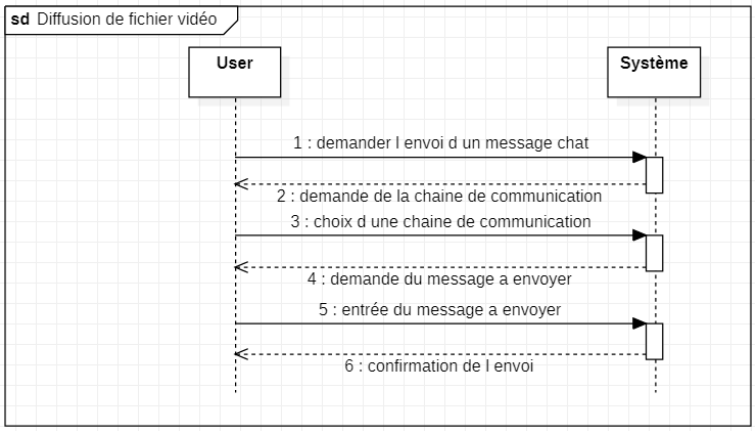
- Use case : Partage de l'écran



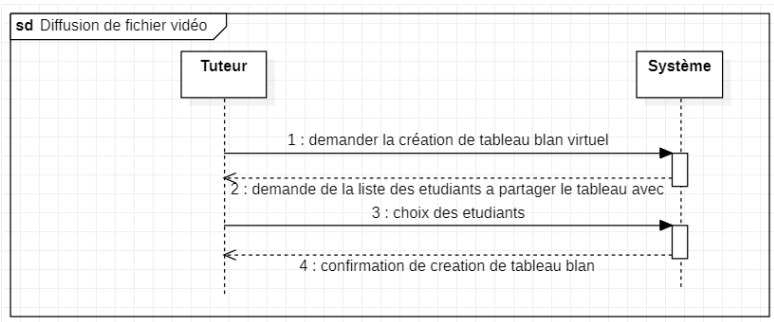
- Use case : diffusion de fichier vidéo



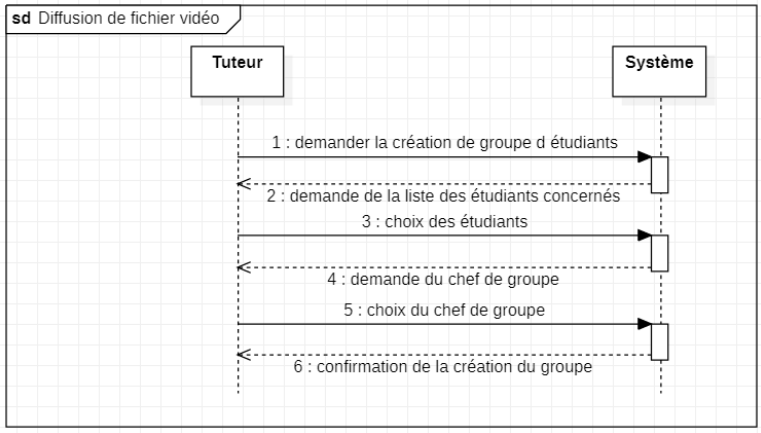
- Use case : communication par chat



- Use case : gestion de tableau blanc



- Use case : gestion des groupes d'étudiants



4.2 Mécanismes

[Présenter les mécanismes et patterns du logiciel.

Pour chacun décrire son mode d'emploi pour qu'un concepteur puisse l'utiliser facilement. Ajouter éventuellement des diagrammes d'interaction. Tracer avec les besoins non fonctionnels.]

5. Autres vues

6. Concepts du domaine

[Une description des concepts spécifiques du domaine et de leurs relations. Préciser comment les choix d'architecture préservent l'indépendance de ces concepts vis à vis de la technologie.]

7. Qualités de l'architecture

L'architecture vue précédemment présente les avantages suivants :

- Découplage des différentes couches (tiers) de l'application (couche web, métier et données)
- Mise à l'échelle simple par l'augmentation du nombre d'instances dans chaque couche de façon indépendante des autres couches

- Fiabilité en évitant les goulots d'étranglement (présence de plusieurs instances de serveurs dans chaque couche de l'architecture)

Elle peut aussi être sujet d'amélioration pour faire face aux inconvénients suivants :

- Absence de mécanisme de récupération dans le cas de plantage d'un des balanceurs de charges, qui sont les seules liaisons entre les différentes couches.

Cependant, l'architecture peut être très simplement entendue par tout type d'extension en l'intégrant comme couche supplémentaire dans l'architecture.

8. Points ouverts