Projet tutoré

Piccadilly Game

Matéo Achterfeld
Tom Georgelin
Guillaume Keller
Martin Jossic

IUT Nancy-Charlemagne 2019-2020
Pierre-André Guénégo

Table des matières :

Introduction :

Projet tutoré Piccadilly Game :

Le projet tutoré est une matière à part entière du DUT, elle s'étend sur les troisième et quatrième semestres (durant la deuxième année).

Le présent document présente ainsi le travail réalisé entre le mois d'octobre (date de réception du projet) et le mois d'avril (date du rendu final).

Présentation du projet :

Piccadilly Game est un concept d'application web proposant une interaction massive avec le public. Les utilisateurs utilisent leur périphérique pour interagir en communauté avec un support commun d'affichage (écran publicitaire, écran d'ordinateur, installation robotique, projection…). Le panel de possibilité est infini, passant de la simple application de sondage pour un grand nombre, au jeu multijoueur temporairement accessible.

Le joueur aura sur son appareil une vision personnalisée de l'application en fonction de ses données, tandis que le support central (écran le plus souvent) sera un affichage global de l'état du jeu.

Ce projet sera basé sur une architecture client-serveur. Pour avoir une vision globale et claire de notre jeu, nous présenterons celui-ci à l'aide de différents diagrammes.

Présentation de l'équipe :

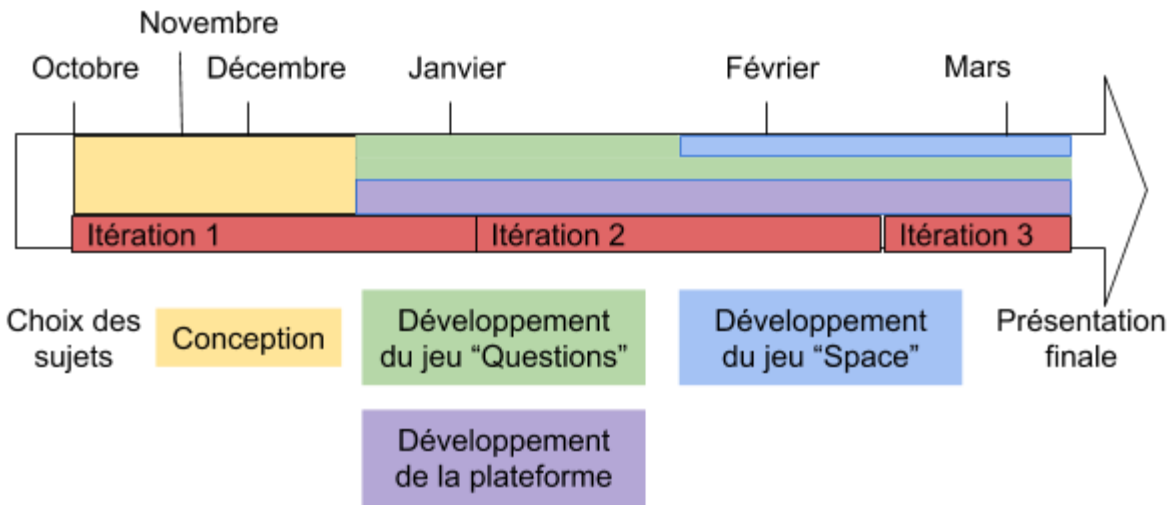
Matéo Achterfeld : Matéo a fait partie de l'équipe de conception de notre projet tutoré et a quitté l'équipe courant janvier pour compléter son quatrième semestre au Canada. Il aura donc fait partie des fondations de notre application.

Tom Georgelin : Tom développe toutes les maquettes sur lesquelles nos jeux sont basés. Il a également traité tous les aspects graphiques de l'application et a codé la partie contrôleur des deux jeux.

Martin Jossic : Développeur d'une grande partie de notre système de communication WebSocket, Martin a contribué à l'intégration des maquettes dans le jeu. De plus, il a permis le côté plug-in de notre application permettant de conserver la même plateforme de démarrage du jeu, laissant la possibilité de "brancher" rapidement des nouveaux jeux.

Guillaume Keller : Guillaume a contribué au développement des maquettes et aux intégrations avec Martin. Il vient également en soutien sur l'aspect jouabilité des jeux et a mis en place l'écran des scores ainsi que la coloration pour le jeu de fusée.

Planning de déroulement du projet :



Analyse :

Après avoir fait des recherches sur l'existant, nous avons trouvé plusieurs plateformes proches de notre concept mais aucune ne réunissait à la fois à l'aspect massif (du nombre de joueurs) et le côté "sans installation" utilisé dans notre plateforme.

C’est ainsi que Piccadilly Game s’inscrit comme une nouveauté en son genre en proposant un accès rapide depuis le navigateur ainsi que la possibilité de se joindre en grand nombre à une même plateforme de jeu.

Notre application s’articule autour des technologies du web et ce fut une contrainte forte donnée avec le sujet. De ce fait, il a fallu réfléchir en priorité à celles-ci. Par chance, le choix est assez vite fait en termes de possibilité. Le classique trio HTML[1]/CSS[2]/JS[3] a été la partie la plus facile à trouver et répond pleinement aux attentes pour ce qui est de la partie client.
En effet, à l’aide de ces technologies, il est aisé de s’assurer d’une grande compatibilité d’affichage.

La suite de notre réflexion s’est donc axée vers la stratégie à employer côté serveur. Notre formation nous ayant déjà apporté le PHP[4], il aurait été logique de l’utiliser. Cependant, l’aspect temps réel demandé pour le projet aurait vite demandé une migration de nos différents codes vers un langage permettant d’interagir rapidement. Le PHP ne permet en effet pas nativement une communication en temps réel et le serveur ne peut envoyer de données à un client sans que ce dernier ne fasse une requête. PHP est plutôt utilisé dans le cadre de sites web dynamiques ou de simple API[5] HTTP[6]. Nous ne nous sentions pas à l’aise à l’idée de créer une application temps réel avec ce langage.

Il a également été question de bases de données. Celles-ci étant une charge supplémentaire pour notre application et uniquement utilisée à des fins très peu intéressantes pour notre public, il a été décidé de ne pas garder cette idée afin de se consacrer pleinement au développement de notre plateforme.

Parmi les technologies utilisées de nos jours, nous retrouvons notamment Node.JS, il s’agit de la solution la plus adaptée à notre projet puisqu’elle permet une communication plus flexible entre les joueurs et le serveur que PHP.

[1] HTML : Langage web pour structurer les pages | cf. Lexique
[2] CSS : Langage web pour styliser les pages | cf. Lexique
[3] JS : Abréviation de JavaScript : langage de programmation pour le web | cf. Lexique
[4] PHP : Langage de programmation web coté serveur | cf. Lexique
[5] API : Interface de programmation | cf. Lexique
[6] HTTP : Protocole standard d’échange sur le web | cf. Lexique

Le protocole WebSocket[7] standardisé sur tous les navigateurs modernes est utilisable dans Node en installant une bibliothèque et il permet d'envoyer des données au joueur n'importe quand. De plus, le Framework[8] Node.JS s'utilise en JavaScript. C'est le même langage que côté client donc on se retrouve avec seulement un langage pour coder toute l'application (en considérant que HTML et CSS ne sont pas des langages de programmation).

L'API WebSocket s'apparente à l'API socket réseau classique offerte par différents langages de programmation hors web tel que Java ou le C. La communication est bidirectionnelle et suffisamment rapide pour l'utilisation qu'on en fait ici.

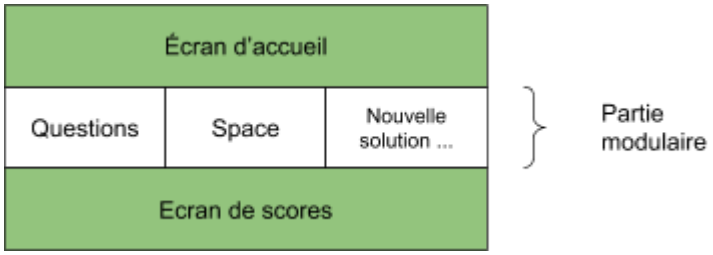
[7] WebSocket : Protocole de communication Web | cf. Lexique
[8] FrameWork : Littéralement cadre de travail, permet de faciliter le développement | cf. Lexique

Réalisations :

Généralités

La plateforme Piccadilly Game est conçue comme une solution “plug-in” grâce à la facilité d’ajouter des jeux ou applications sur la base déjà constituée. Le socle commun s’occupe de l’écran d’accueil avec la salle d’attente et l’écran de score.
Cette interface commune peut une fois la partie terminée, en redémarrer une nouvelle s’il y a assez de joueurs qui se sont joins à la partie.

De ce fait chaque nouvelle solution ne nécessite plus qu’un développement réduit puisqu’il ne reste aux programmeurs “que” l’aspect particulier du jeu à développer.



Une fois le jeu lancé, on fonctionne toujours avec une paire écran/joueur(s).

De par la philosophie de Piccadilly Game, un joueur doit avoir un écran à proximité afin de se repérer dans le(s) jeu(x) (en plus de son téléphone qui sert de manette). De ce fait, chacun peut connecter (avant le début de la partie) un écran afin de jouer en ayant un aperçu de la partie en cours à proximité. La configuration optimale restant avant tout un lieu avec de nombreux joueurs qui utilisent un même écran principal, afin d’optimiser les envois de données du serveur.

Détail des itérations :

Dès le début du projet il a été question de plusieurs stades de développement de l'application, notamment basés sur le calendrier des itérations successives (*cf. planning de déroulement du projet*).
Nous avons ainsi reçu pour consigne de commencer avec un jeu ayant un usage permissif des communications temps réel.
C'est de ce fait un jeu de sondage ou de questionnaire (en fonction de l'utilisation que l'on souhaite en faire) qui est devenu notre premier fait d'armes et qui a été présenté lors de la première itération.

Par la suite, un deuxième jeu a été requis, pour se rapprocher de l'aspect temps réel demandé.
Nous avons alors, avec les conseils de notre tuteur, décidé de mettre en place un jeu dans l'idée du best-seller qu'a été Flappy Bird[9]. Le concept étant déjà familier à une large partie de la population, ce jeu permet une rapide acquisition des réflexes pour les joueurs.
Afin de coller au mieux avec l'aspect massif du nombre de joueurs et à la jouabilité qui est fonction de la connexion des joueurs, le concept a été adapté pour être utilisé avec un curseur qui positionne le joueur verticalement (plus facile à manipuler qu'une commande de saut comme dans le jeu Flappy Bird original).
De ce fait, la deuxième itération a été dédiée à l'intégration du jeu au sein de notre plateforme commune au premier jeu (salle d'attente, affichage des scores, compte à rebours de début de partie…), en parallèle du développement du nouveau jeu lui-même.

Dès lors que l'application pouvait recevoir plusieurs jeux il a été question de pouvoir rapidement configurer le serveur. Ceci s'est fait grâce à la mise en place d'un fichier de configuration hébergé sur le serveur. Afin d'éviter les doublons, les informations de configuration doivent également se diffuser sur les clients grâce à WebSocket.

La dernière itération a été très vite tournée vers l'optimisation de l'expérience utilisateur du fait de sa durée extrêmement courte ainsi que des circonstances actuelles.

[9] https://flappybird.io

Il a donc été mis en point d'orgue le fait :

- D'aider l'utilisateur à se repérer dans le jeu "Space"
- De permettre à tous les jeux d'être intégré de la manière la plus légère possible sur le corps de Piccadilly Game
- De corriger au maximum les problèmes détectés lors des utilisations successives (bugs)
- D'automatiser la structure le plus possible (par la relance des jeux automatique)

Liste des fonctionnalités :

- Fonctionnement du jeu en temps réel comme si tout le monde jouait sur une même "console de jeu" (le serveur peut envoyer par WebSocket des données aux clients sans que celui-ci ait à envoyer de requêtes d'actualisation)

- Connexion rapide grâce à un QR Code (plus rapide qu'un lien à taper)

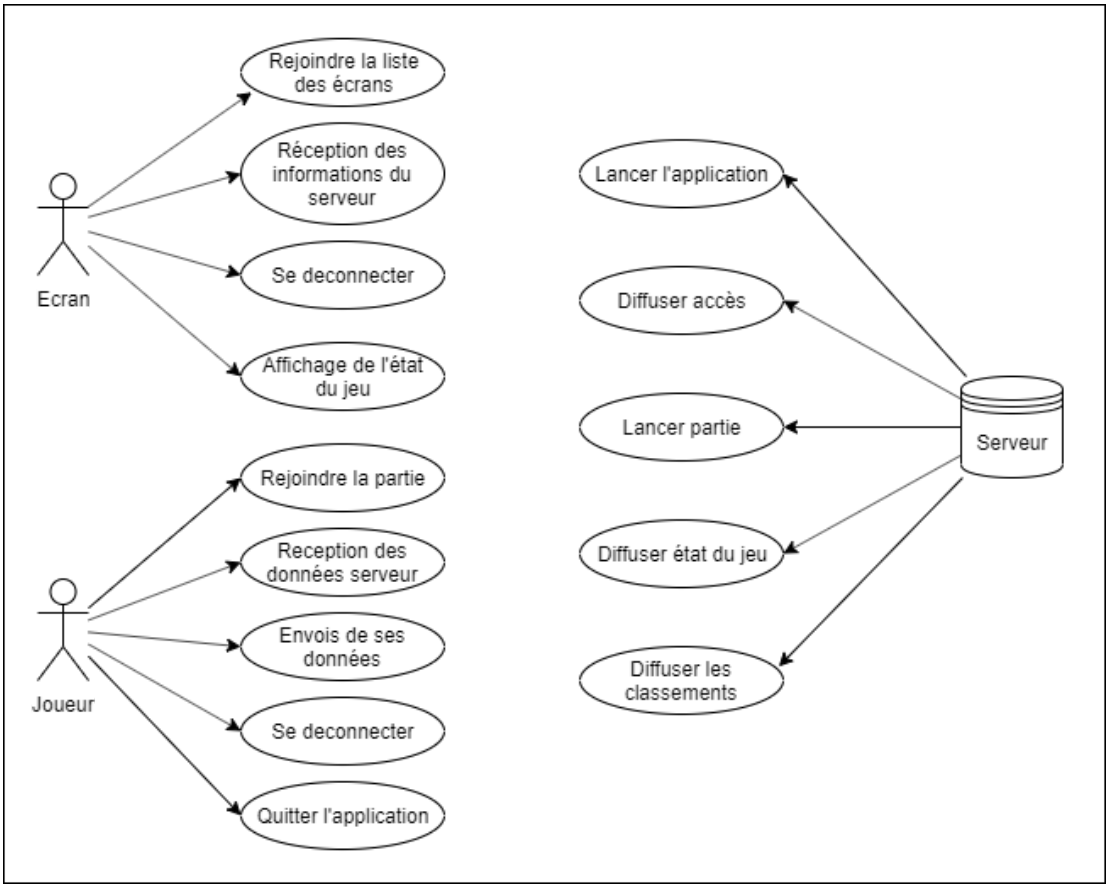
- Retransmission de l'état global de la partie sur un ou plusieurs grands écrans

- Deux jeux possibles :
  - Jeu de questions/sondage
  - Jeu type "Flappy Bird" : **Space**

- Les utilisateurs peuvent rejoindre l'ensemble des jeux en cours de partie avec une prise en compte des modifications apportées côté serveur.

Diagramme de cas d'utilisations :

Serveur et client :



Le serveur est le maître incontesté du jeu. C'est lui qui reçoit, analyse puis diffuse les informations relatives au jeu. Il est également responsable du lancement de la partie.
En effet, c'est lui qui compare le nombre de joueurs connectés sur le serveur au nombre de joueurs nécessaires pour débuter la partie. Une fois les conditions requises complétées, le jeu est lancée de manière automatique et chaque client (joueur + écran) est notifié du début de la partie.

Dans ce diagramme, on peut voir que le joueur doit pouvoir rejoindre une partie, mais également qu'il sera maître de ses actions sur le jeu (connexion, déconnexion, actions propres au jeu).

Les écrans sont les intervenants les plus passifs, ils n'ont pour seul but que l'affichage des informations envoyées par le serveur et n'envoient rien à celui-ci.

La description du serveur sera complétée par la suite avec deux diagrammes d'activités permettant de repérer de manière temporelle les différentes actions.

Nous décrirons par la suite au travers de différents diagrammes les cas d'utilisations suivants :

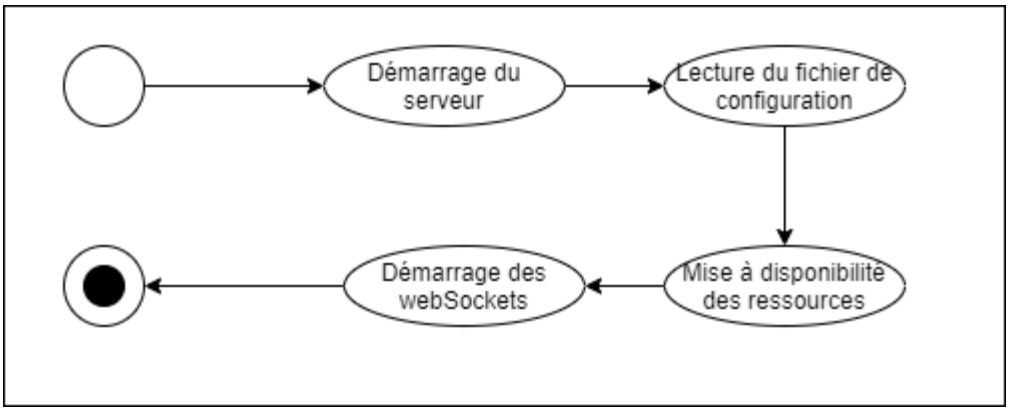
- Serveur :
  - Mise en place de la partie (Démarrage)
  - Déroulement du jeu

- Joueur :
  - Accès au jeu
  - Déroulement du jeu

Diagrammes d'activité :

Point de vue du serveur :

Démarrage du jeu



Au lancement du serveur, les données générales et propres au jeu sont initialisées en partie grâce au fichier de configuration présent sur le serveur qui comprend :

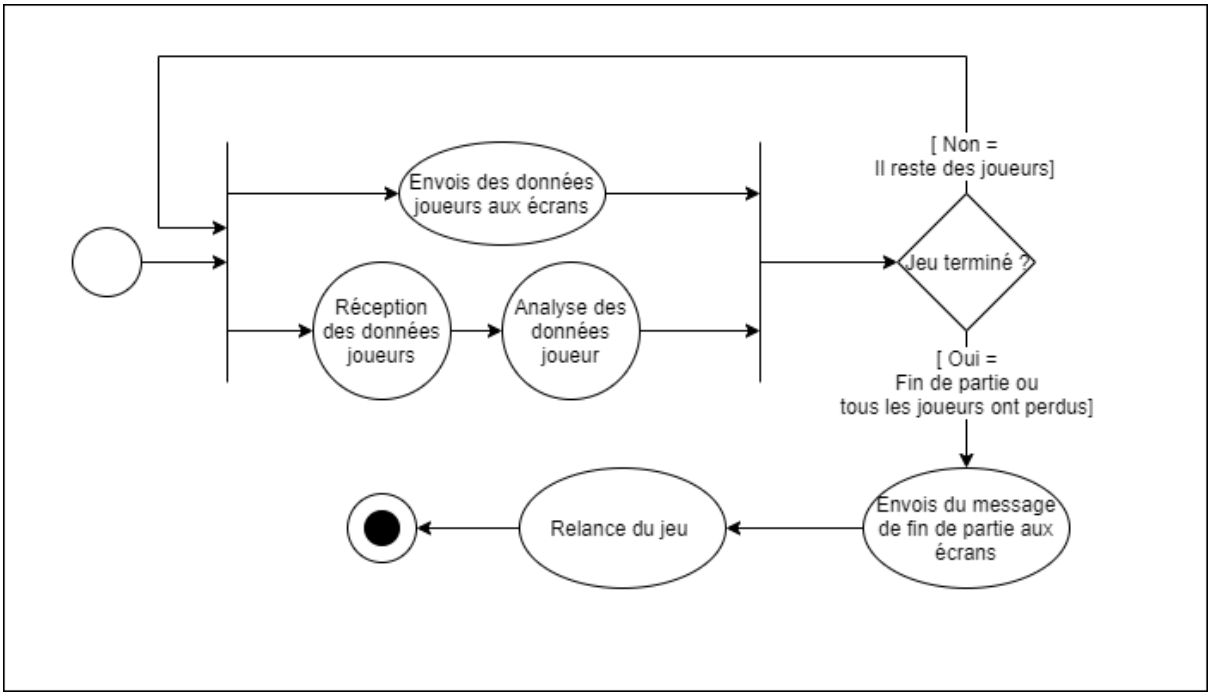
- Le port du serveur
- Le nombre de joueurs minimum pour lancer le jeu
- Le jeu à lancer
- Si c'est le jeu de questions : nombres de questions à diffuser
- La durée du compte à rebours avant le début de la partie
- L'activation d'un mode de test destiné aux développeurs

Les ressources sont identifiées et associées à des URL afin de les rendre accessibles.

Le système WebSocket sera démarré dès que ces informations seront chargées afin de ne laisser personne entrer sur le jeu avant que tout soit opérationnel. Les URL sont rendues disponibles à ce même moment.

Les informations récupérées dans le fichier de configuration permettront de créer l'interface de jeu choisie.

Déroulement du jeu



Par la suite, c'est un protocole personnalisé reposant sur WebSocket qui est utilisé pour traiter l'intégralité du fonctionnement du jeu.

Les états du jeu sont transmis du serveur aux clients (écran + joueurs) via ces mêmes WebSockets.

- Le serveur réceptionne en continu les données provenant des joueurs : communication bidirectionnelle (Serveur ↔ Joueur)
- Les écrans ont, eux, une communication unidirectionnelle (serveur → écran) uniquement.

Une fois l'application lancée, et le nombre de joueurs minimums atteint, le serveur envoie à l'écran/aux écrans un compte à rebours avant le lancement du jeu.

La boucle d'évènements du jeu est alors lancée, avec une réception constante des données des joueurs puis une retransmission de ce qui a été reçu vers l'ensemble des écrans afin d'actualiser les informations générales.

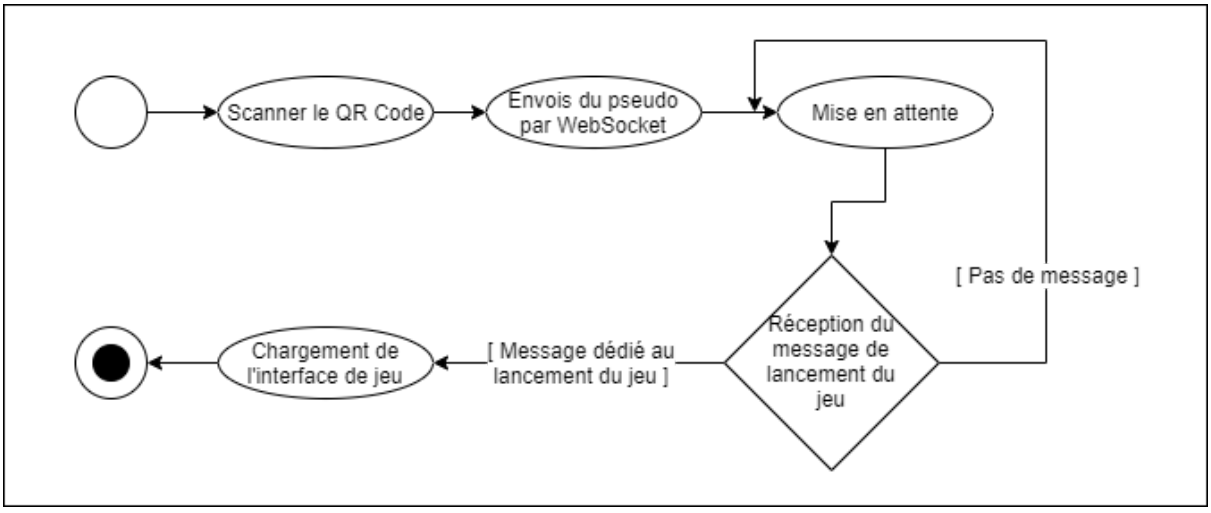
Le jeu de questions dispose également d'une réception des données de joueurs à joueurs durant la partie afin de mettre à jour les statistiques liées à la question en cours.

Une fois la partie terminée, le score individuel est diffusé aux joueurs par leur propre terminal, l'écran général lui diffuse les scores de tout le monde et les trie, permettant ainsi de voir les classements.

Après la consultation des résultats un délai est mis en place avant la relance du jeu, permettant de rejouer indéfiniment dès lors que le serveur a été lancé une fois.

Point de vue du client :

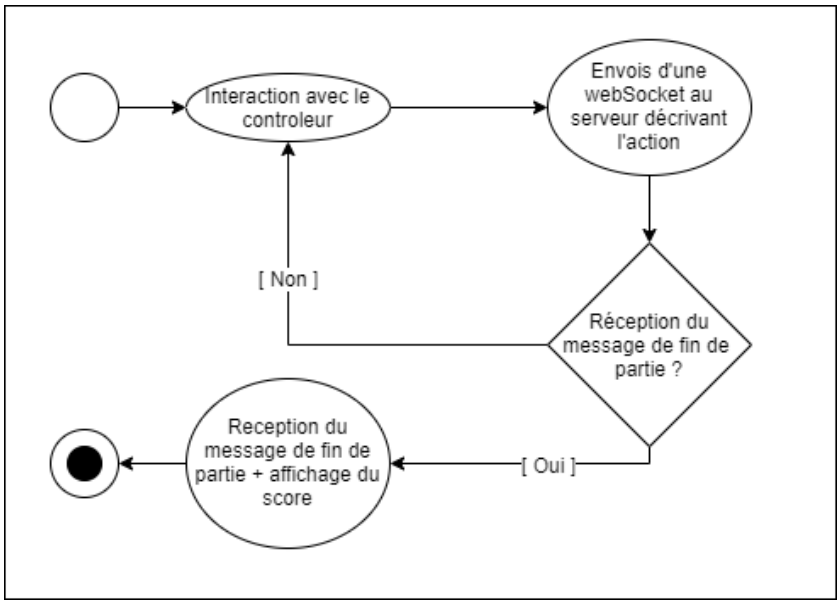
Début de la partie



L'expérience du joueur débute par le scan d'un QR Code ou bien à l'aide d'un URL disponible sur la page principale.

Le joueur se connecte sur la page et est par la suite invité à entrer un pseudo, il est alors mis en attente et peut voir en temps réel l'arrivée des autres joueurs.
Dès lors qu'il y a assez de joueurs qui ont rejoint la partie, l'affichage du joueur bascule vers une annonce du début du jeu, avant de basculer vers le jeu à proprement parlé ainsi que ses contrôleurs dédiés.

Déroulé de la partie



A partir de ce moment, à chaque interaction du joueur avec son affichage, un message WS contenant les informations de commande sera envoyé au serveur.

Au moment où la partie se termine :

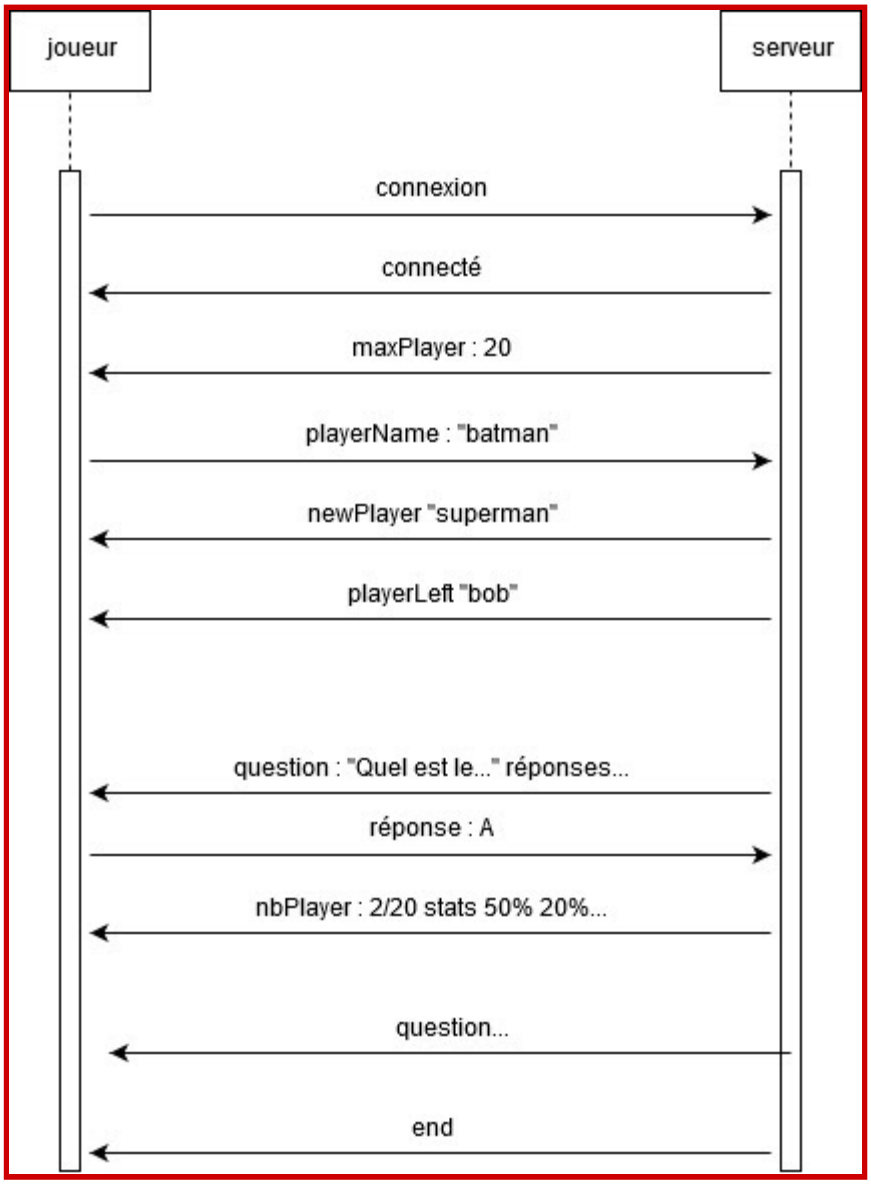
- Pour le jeu de questions : Le score est envoyé au joueur
- Pour le jeu “Space” : Un message d’attente “vous êtes mort” est envoyé pour conserver l’attention du joueur avant que tout le monde termine sa partie, par la suite un nouveau message est envoyé avec le score du joueur

Un joueur peut rejoindre la partie en cours de jeu pour les deux jeux, seule nuance, il partira avec un capital de points moins important que les autres mais ne sera pas plus pénalisé que cela.

Dans le jeu des questions, si le décompte est en cours, le joueur pourra immédiatement rejoindre la partie, dans le cas où il y a l’affichage de la réponse, celui-ci intègrera la partie quelques secondes plus tard à la prochaine question.

Pour le jeu “Space”, la fusée du joueur sera propulsée en première position sur l’écran principal.

Diagramme de séquence :
Interaction client-serveur lors d'une partie



Ce diagramme montre les messages circulants entre le client et le serveur pendant une partie avec un jeu de type QCM (comme dans l'émission "Qui veut gagner des millions"). Une série de questions est préparée sur le serveur.

Les joueurs ont un temps limité pour répondre à chaque question en choisissant la réponse A, B, C… Une fois qu'un joueur a répondu à une question, il se met en attente de la question suivante.

Pendant ce temps, il reçoit en temps réel les statistiques de réponse des autres joueurs. Sur chaque réponse, il voit le taux de clic en pourcentage (50% d'utilisateurs ont répondu B par exemple). Quand le temps de réponse est écoulé, on passe à la question suivante.

Explication de la séquence de communication

- "***connexion***" et "***connecté***" : étape de connexion WebSocket.

- "***minPlayer***" et "***maxPlayer***" : avant que la partie se lance, le joueur est mis dans une "***salle d'attente***" où il voit le nombre de joueurs ayant rejoint la partie, le message "***minPlayer***" permet simplement au client de connaître le nombre de joueur minimal pour afficher sur la page "***encore n joueur(s) manquant(s)***". Le message "***maxPlayer***" permet de connaître le nombre maximum de joueurs dans la partie et d'afficher ***"x/20 joueurs connectés***".

- "***playerName***" est envoyé au serveur pour indiquer le pseudo choisi par le joueur.

- "***newPlayer***" est reçu quand un nouveau joueur rejoint la partie. On donne avec ce message le pseudo du joueur s'étant connecté. Le client maintient en local la liste des joueurs connectés et en déduit le nombre de joueurs présents à afficher dans "***x/20 joueurs connectés***".

- "***playerLeft***" est reçu quand un joueur a quitté la partie. On reçoit avec ce message le nom du joueur s'étant déconnecté pour que le client l'efface de sa liste de joueurs interne.

- "***question***" est reçu quand une nouvelle question est lancée et est accompagné des réponses possibles à la question parmi lesquels l'utilisateur fera son choix. Ce message met fin à la question précédente (il peut être envoyé avant la fin du temps si tous les joueurs ont répondu avant l'échéance). Quand on est en "***salle d'attente***", ce message lance aussi la partie en plus de donner la première question (avec les réponses possibles).

- "***réponse***" est envoyé pour indiquer au serveur la réponse choisie par l'utilisateur (A, B, C…). Une fois qu'il a répondu, l'utilisateur ne peut pas modifier sa réponse.

- "***nbPlayer…***" contient les données sur les réponses données par les autres joueurs à savoir le nombre de joueurs ayant répondu et combien de joueurs (en pourcentage) ont cliqué sur les réponses A, B, C... Ces données ne sont reçues qu'une fois que l'utilisateur a lui-même répondu à la question en cours.

  On retourne à "***question***" jusqu'à ce qu'il n'y ait plus de questions.

- "***end***" est reçu quand toutes les questions ont été jouées. La partie est alors terminée.

Recensement, évaluation des risques et retour sur expérience

Connexion des utilisateurs :

Des utilisateurs avec des connexions différentes doivent avoir accès au jeu sans qu'on ait de problèmes :

- Différents types de connexions ont été testés (Wi-Fi, 4G, H+…) au travers de différents types d'installations (réseau local, hébergement sur serveur distant).

- Les différences de débit entre les joueurs n'ont pas démontrés de résultats significatifs pour abandonner un type de connexion. Il semble que l'intégralité des joueurs pourront avoir une connexion satisfaisante malgré les différences entre les connexions (le faible "poids" des messages WS est très certainement la partie permettant cette variance insignifiante)

Surcharge du serveur :

- Le serveur doit être proportionné à la charge qu'on lui impose (limitation du nombre de joueurs).
- Durant nos tests, nous avons rarement testé avec plus de 10 joueurs. À ce jour nous n'avons pas eu de latence due à la surcharge du serveur.

Latence durant l'application :

- Si le serveur éprouve des difficultés à répondre à temps, il faut prévoir une méthode pour diminuer ce temps.
- L'envois n'étant pas fait à intervalle régulier, le serveur n'envoie des données que lorsqu'il est nécessaire d'en envoyer, ce qui allège la quantité de données à envoyer.

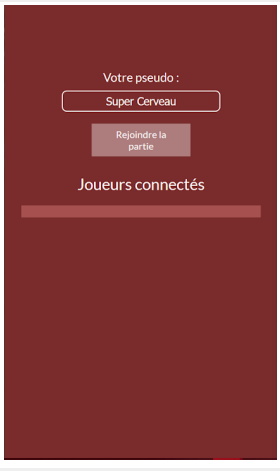
Développement des cas d'utilisation développés

Résumé des cas d'utilisations

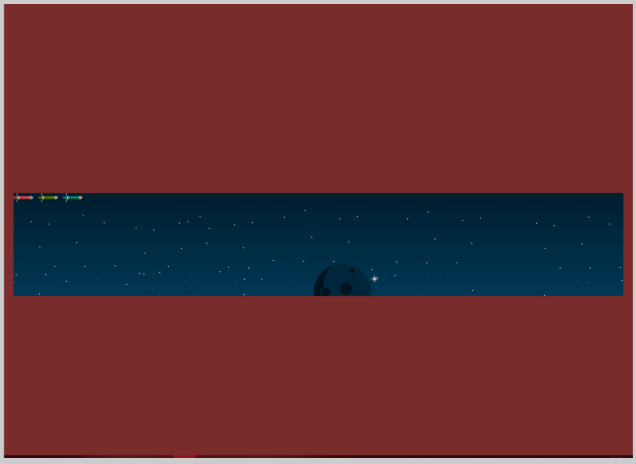
(Utilisateur)	*(Serveur)*
Rejoindre la partie Réception des données du serveur Envois de ses propres données Se déconnecter Quitter l'application *(Écran)* Rejoindre la liste des écrans Recevoir les données de jeu Quitter la liste des écrans Afficher l'état du jeu	Lancer l'application Diffuser les accès Lancer la partie Diffuser l'état du jeu Diffuser les classements

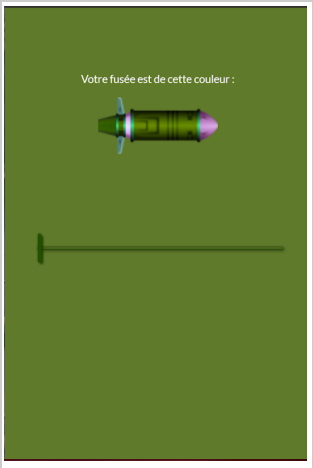
Visuels de l'application :

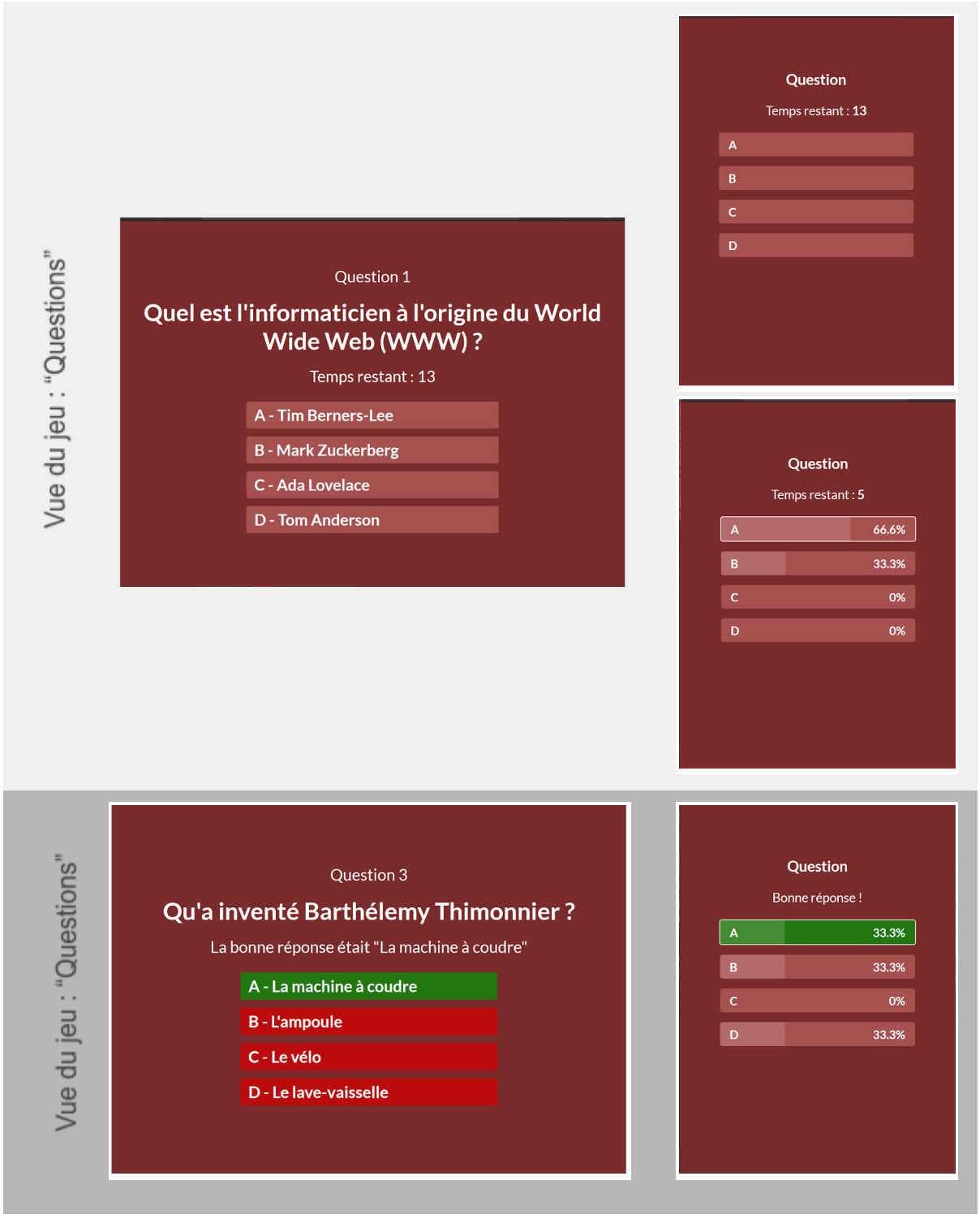
Page d'accueil

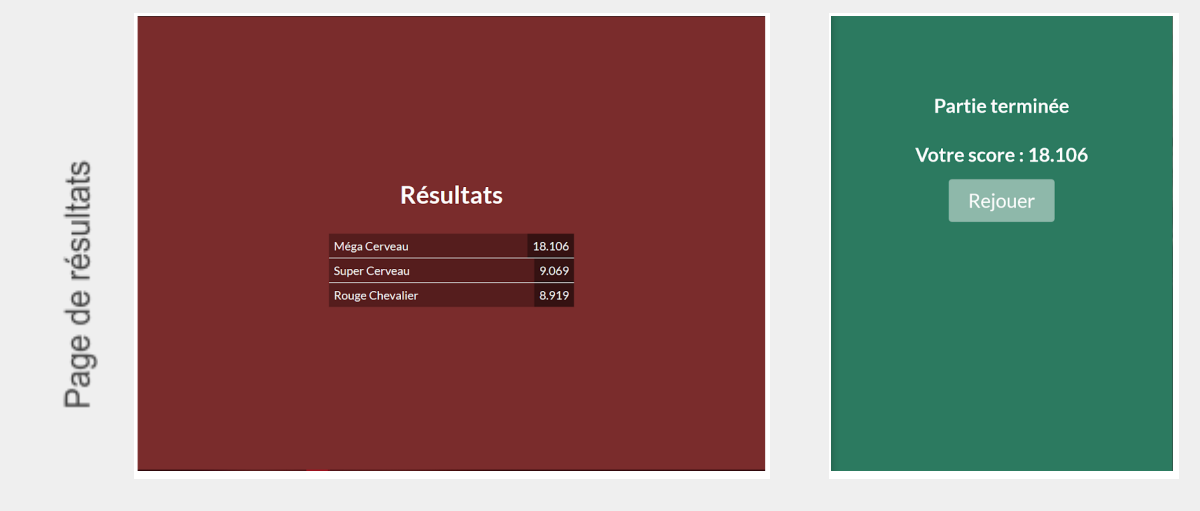


Vue du jeu : "Space"






Vue du jeu : "Questions"
Question 1
Quel est l'informaticien à l'origine du World Wide Web (WWW) ?
Temps restant : 13
A - Tim Berners-Lee
B - Mark Zuckerberg
C - Ada Lovelace
D - Tom Anderson
Question
Temps restant : 13
A
B
C
D
Question
Temps restant : 5
A 66.6%
B 33.3%
C 0%
D 0%
Vue du jeu : "Questions"
Question 3
Qu'a inventé Barthélemy Thimonnier ?
La bonne réponse était "La machine à coudre"
A - La machine à coudre
B - L'ampoule
C - Le vélo
D - Le lave-vaisselle
Question
Bonne réponse !
A 33.3%
B 33.3%
C 0%
D 33.3%



Conclusion :

En l'état notre projet pourrait être repris par une équipe sans grandes difficultés grâce au côté adaptatif qui a été mis en place dans les itérations précédentes. A l'aide de la documentation produite il faudra donc s'approprier la manière de produire le contenu spécifique pour un nouveau jeu.

Le projet Piccadilly Game aura été une belle opportunité pour découvrir les technologies du web ainsi que les possibilités offertes par celles-ci.
Le fait de ne passer que par le navigateur à la place d'une application dédiée nous a semblé être une idée novatrice d'un point de vue utilisateur, en effet, on trouve ici une certaine liberté sur l'utilisation, la mise en place est très rapide et n'impose pas de garder la plateforme sur l'appareil.
De plus, les langages web ont pour avantage d'être compris par tout type de périphérique (smartphones, tablettes, ordinateurs…).

Un projet d'une telle envergure reste quelque chose de nouveau au cours de notre scolarité et nous a permis de nous approprier de nouvelles méthodes de travail par rapport à nos habitudes.
Entre autres, autour de la gestion du temps, avec les 3 itérations, mais aussi avec une phase de conception longue, bien souvent abordée avec trop de légèreté sur les projets de plus petites envergures.

L'utilisation de logiciels spécialisés dans la gestion de projet ainsi que des patrons de conception ont aussi été la manière la plus efficace de s'organiser et de partager notre travail dans le groupe.

Nous avons également au sein de ce projet, su remarquer la grande différence entre les premières esquisses et la réalisation finale. Bien que proche l'une de l'autre, ces deux versions restent néanmoins différentes grâce à la maturité grandissante du projet au cours des 6 mois de travail.

L'approche du côté temps réel, n'ayant pas été abordé souvent au cours de notre cursus, nous a également permis d'ajouter des compétences non négligeables pour le futur.
De la même manière l'utilisation de Node.JS, une seconde manière de programmer côté serveur, est une compétence qui nous sera, à coup sûr bénéficiable durant la suite de notre carrière.

Le jeu n'étant qu'un prétexte pour mettre en application toutes les technologies et tous les aspects techniques, il reste cependant une très bonne idée dans l'optique du partage de notre création.
En effet, il sera facile de regrouper autour de soi de nombreuses personnes (famille, amis, …) afin de tester notre application dans un premier temps, puis, même de l'utiliser pour ajouter un moment de convivialité et de compétition.

Nous tenons à remercier M. Guénégo pour le sujet ainsi que sa grande implication dans le projet, en nous stimulant avec de nouvelles idées de manière très fréquente.

Lexique :

HTML :

HTML signifie « HyperText Markup Language » qu'on peut traduire par « langage de balises pour l'hypertexte ». Il est utilisé afin de créer et de représenter le contenu d'une page web et sa structure.

CSS :

Cascading Style Sheets (CSS) est un langage de feuille de style utilisé pour décrire la présentation d'un document écrit en HTML ou en XML (on inclut ici les langages basés sur XML comme SVG ou XHTML). CSS décrit la façon dont les éléments doivent être affichés à l'écran, sur du papier, en vocalisation, ou sur d'autres supports.

Javascript (JS) :

JavaScript (qui est souvent abrégé en « JS ») est un langage de script léger, orienté objet, principalement connu comme le langage de script des pages web.

PHP :

Le PHP, pour Hypertext Preprocessor, désigne un langage informatique, ou un langage de script, utilisé principalement pour la conception de sites web dynamiques.

API :

Une API est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. API est un acronyme anglais qui signifie « Application Programming Interface », que l'on traduit par interface de programmation d'application.

WebSocket (WS) :

WebSocket est une technologie évoluée qui permet d'ouvrir un canal de communication bidirectionnelle entre un navigateur (côté client) et un serveur

Node.js :

“Node.js is an open-source and cross-platform JavaScript runtime environment. It is a popular tool for almost any kind of project!”

Node.js est un environnement d'exécution JavaScript open-source et inter-plateforme. C'est un outil populaire pour pratiquement tous les types de projets.

Framework :

Un Framework (ou infrastructure logicielle en français) désigne en programmation informatique un ensemble d'outils et de composants logiciels à la base d'un logiciel ou d'une application.

Bibliographie :

https://developer.mozilla.org/fr/docs/Web/HTML
https://developer.mozilla.org/fr/docs/Web/CSS
https://developer.mozilla.org/fr/docs/Web/JavaScript

https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203597-php-hypertext-preprocessor-definition/

https://www.redhat.com/fr/topics/api/what-are-application-programming-interfaces

https://developer.mozilla.org/fr/docs/Web/API/WebSockets_API
https://nodejs.dev
https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203355-framework/