Équipe 111

PolyQuiz

Protocole de communication

Version 2.0

Historique des révisions

Date	Version	Description	Auteur
2024-09-14	1.0	Interfaces/Protocoles avant ajouts des fonctionnalitées	Simon Cloutier
2024-09-27	1.1	Ajout des Protocoles pour les nouvelles fonctionnalités	Simon Cloutier
2024-09-27	2.0	Vérification et corrections	Damaris Calestrov Manel Abroudj Lina Belloui Simon Cloutier Aymen Ghodbane Motassembellah Mohamed Bassiouni

Table des matières

Protocole de communication

1. Introduction

Ce document a pour but de décrire les divers protocoles de communication qui seront utilisés au sein de ce projet, ainsi que les entités qui seront échangées au sein des interactions entre les clients et le serveur.

2. Communication client-serveur

Pour parvenir à établir une communication entre les clients et le serveur, il a été décidé d'utiliser des requêtes HTTP ainsi que des websockets.

Les applications clientes "desktop" ainsi que le serveur étant programmés en Typescript avec le cadriciel Angular, la librairie "socket.io" sera utilisée pour mettre en place les connexions sockets. Lorsqu'il est question des clients mobiles, puisque nous avons pris la décision d'utiliser Kotlin comme langage de programmation, nous avons opté d'utiliser la librairie io.socket".

Pour les requêtes HTTP, nous avons décidé d'opter pour la librairie "http" d'angular. Pour les requêtes HTTP pour Kotlin, nous utiliserons "okhttp3". Outre les communications avec le serveur, les clients communiqueront aussi avec le service d'Okta "Auth0." Les requêtes utilisent le format JSON pour transmettre les différentes valeurs. Nous allons utiliser le standard REST pour effectuer ces premières.

Nous allons utiliser le protocole de communication WebSocket pour les fonctionnalités de parties, d'ajout d'amis de manière dynamique lorsque les utilisateurs concernés par les fonctionnalités sont connectés, ainsi que les fonctionnalités de chat. En effet, nous avons besoin d'avoir des communications bidirectionnelles pour accomplir ces tâches, et les websockets sont la solution pour y parvenir.

Nous allons utiliser le protocole HTTP pour mettre en place les fonctionnalités de créations de parties, de création de Quiz, de personnification des comptes, de système de monnaie ainsi que de création de compte. Ces fonctionnalités ne nécessitent pas une communication continue entre les divers partis.

3. Description des paquets

3.1 Paquets HTTP

3.1.1 Paquets HTTP Games

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	/games	—	Récupérer la liste de tous les jeux	—	Game[]	Succès : 200 Échec : 404, 409
POST	/games/title	—	Vérifier qu'un jeu existe avec son titre	title : string	titleExist : boolean	Succès : 201 Échec : 404
POST	/games	—	Ajouter un nouveau jeu à la liste	Game	Game	Succès : 201 Échec : 404
PUT	/games	—	Modifier un jeu	Game	—	Succès : 200 Échec : 404
POST	/games/title/:title	—	Vérifier qu'un jeu existe avec son titre	{title : string}	titleExist : boolean	Succès : 201 Échec : 404
GET	/games/title/:title	title: string	Vérifier qu'un jeu existe avec	—	Game	Succès : 200

			son titre			Échec : 404
GET	/games/:id	id: string	Récupérer un jeu selon son id	—	Game	Succès : 200 Échec : 404
DELETE	/games/:id	id: string	Supprimer un jeu selon son id	—	—	Succès : 200 Échec : 404
PATCH	/games/:id/ update-visibility	id: string	Mettre à jour la visibilité du jeu selon son id	boolean	—	Succès : 200, 204 Échec : 403, 404
DELETE	/games/tag/:tag	tag: string	Enlever un tag de la base de données	—	—	Succès : 200, 204 Échec : 400, 404
POST	/game/:id/rate/	id: string	Évaluer un match	—	MatchEvaluation	Succès : 200, 201 Échec : 400, 404

3.1.4 Paquet HTTP Session

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour

DELETE	/session/:userId	userId: string	Changer la session sur la base de donnée pour la mettre comme étant conclue	—	Session | null	Succès : 200 Échec : 404
GET	/session/history/:userId	userId: string		—	SessionHistory[]	Succès : 200 Échec : 404
POST	/session/create/:userId	userId: string	Créer un object Session sur la base de donnée avec le temps	—	Session	Succès : 201 Échec : 404

3.1.2 Paquet HTTP Comptes

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	polyquiz.ca.auth0.com/ authorize	—	Envoyer la requête pour s’authentifier	—	—	Succès : 302 Échec : 400, 401
GET	polyquiz.ca.auth0.com/ v2/logout	—	Effacer tous les matchs de la base de données	—	—	Succès : 200 Échec :

						400
POST	polyquiz.ca.auth0.com/oauth/token	—	Créer un nouveau match	—	—	Succès : 201 Échec : 403, 500
PATCH	polyquiz.ca.auth0.com/api/v2/users/:USER_ID	USER_ID: string	Changer le nom, le nickname et l'avatar d'un compte	name: string; nickname: string; picture: URL;	user_id: string name: string nickname: string picture: string (URL)	Succès : 200, 204 Échec : 400, 401 403, 404
GET	/accounts	—	Retourne les informations de tous les comptes	—	Account[]	Succès : 200 Échec : 500
GET	/accounts/:userId	userId: string	Retourner les informations du compte a l'aide du userId	—	Account	Succès : 200 Échec : 400, 404, 500
GET	/accounts/:pseudonym/pseudonym/	pseudonym: string	Retourner les informations du compte à l'aide du pseudonyme	—	Account	Succès : 200 Échec : 400, 404, 500

POST	/accounts/pseudonym	—	Retourner les informations du compte à l'aide du pseudonyme. Utilisé dans le cas où des caractères spéciaux sont dans la requête.	string	Account	Succès : 201 Échec : 400, 404, 500
PATCH	/accounts/:userId/pseudonym	userId: string	Changer le username en utilisant le noveau passé dans le body	string	Account	Succès : 200 Échec : 400, 404, 500
POST	/accounts/batch	—	Retourner les informations de tous les comptes passés dans le body	string[]	Account[]	Succès : 201 Échec : 400, 404
GET	/accounts/:userId/friends/	userId: string	Accéder à la liste d'amis	—	string[]	Succès : 200 Échec : 404
GET	/accounts/:userId/friend-requests	userId: string,	Envoyer toutes les requêtes des amis venant des autres utilisateurs	—	FriendRequestDTO[]	Succès : 200 Échec : 404
GET	/accounts/:userId/	userId: string,	Envoyer toutes	—	string[]	Succès :

	friend-requested		les requêtes des amis envoyés aux autres utilisateurs			200 Échec : 404
GET	/accounts/:userId/ blockedUsers	userId: string,	Envoyer tous les identifiants des utilisateurs bloqués	—	string[]	Succès : 200 Échec : 404
PATCH	/accounts/:userId/ lang/:lang	userId: string, lang : string	Changer la langue d'affichage	—	Account	Succès : 200 Échec : 401, 404
PATCH	/accounts/:userId/ theme/:themeVisual	userId: string, theme : string	Changer le thème	—	Account	Succès : 200 Échec : 401, 404
PATCH	/accounts/:userId/ ownedThemes	userId: string	Changer les thèmes auxquels l'utilisateur a accès.	ThemeVisual[]	Account	Succès : 200 Échec : 401, 404
POST	/accounts	—	Créer un nouveau compte	CreateAccountDTO	Account	Succès : 201 Échec : 500
PATCH	/accounts/:userId/avatar	userId: string	Changer l'avatar du joueur à l'aide	string	Account	Succès : 200

			du URL passé dans le body			Échec : 401, 404
PATCH	/accounts/:userId/ ownedAvatars	userId: string	Changer les avatars que la personne possède à l'aide de la valeur mise dans le body	string[]	Account	Succès : 200 Échec : 401, 404
Patch	/accounts/:userId/money	userId: string	Changer le montant d'argent que la personne possède à l'aide de la valeur mise dans le body	number	Account	Succès : 200 Échec : 401, 404
GET	/accounts/:id/stats/	id: string	Retourner les statistique du compte	—	gamesPlayed : number; gamesWon: number; avGoodQuestion: number; avGameLength: number;	Succès : 200 Échec : 400, 404
GET	/accounts/:id/games/	id: string	Retourner l'historique des parties du compte	—	Match[]	Succès : 200 Échec : 400, 404
GET	/accounts/avatars	—	Retourner une	—	string[] (URL)	Succès :

			liste d'avatar prédéfinie			200 Échec : 400, 404
GET	/accounts/:id/avatar/	id: string	Retourner l'avatar de l'utilisateur	—	string	Succès : 200 Échec : 400, 404
PATCH	/accounts/:id/avatar/	id: string	Changer l'avatar de l'utilisateur	string	—	Succès : 200, 204 Échec : 400, 401, 404

3.1.3 Paquet HTTP Match

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
GET	/matches	—	Récupérer la liste de tous les matches	—	CurrentMatchesDTO[]	Succès : 200 Échec : 404
DELETE	/matches	—	Enlever tous les matchs du serveur. On	string	string	Succès : 200

			fournit un password dans le body pour s'assurer de la sécurité.			Échec : 401, 500
POST	/matches/match	—	Créer un nouveau match	—	Match	Succès : 201 Échec : 500
POST	/matches/join-match	—	Laisser le joueur rejoindre une partie à l'aide de l'information mise dans le body, et renvoie un feedback	JoinMatchDto	String	Succès : 201 Échec : 400, 403, 404
GET	/matches/match/validity/: accesCode	accessCode : string	Récupérer la validité d'un code d'accès	—	boolean	Succès : 200 Échec : 404
GET	/matches/match/accessibility/ :accessCode	accessCode : string	Récupérer l'accessibilité à un match	—	boolean	Succès : 200 Échec : 404
PATCH	/matches/match/accessibility/ : accessCode	accessCode : string	Modifier l'accessibilité	—	—	Succès : 200, 204

			d’un match			Échec : 404
GET	/matches/match/:accessCode	accessCode : string	Récupérer un match à partir de son code d’accès	—	Match	Succès : 200 Échec : 404
DELETE	/matches/match/:accessCode	accessCode : string	Effacer un match à partir de son code d’accès	—	—	Succès : 200, 204 Échec : 404
POST	/matches/match/playerName Validity	—	Vérifier l’existence d’un nom de joueur	Validation	boolean	Succès : 201 Échec : 404
POST	/matches/match/player	—	Ajouter un joueur à la liste des joueurs du match	UpdateMatch	—	Succès : 201 Échec : 404
POST	/matches/match/:accessCode/ history	accessCode : string	Valider et sauver l’historique d’un match	—	—	Succès : 201 Échec : 404
GET	/matches/history	—	Retourner l’historique de tous les matchs	MatchHistory[]	—	Succès : 200 Échec :

						404
DELETE	/matches/history	—	Supprimer tous les historiques de match	—	—	Succès : 200 Échec : 404

3.1.5 Paquet HTTP Friends

Méthode	URI	Paramètres URI	Description	Corps de la Requête	Corps de la Réponse	Code(s) de retour
POST	/friends/send/:senderId/ :receiverId	senderId: string receiverId: string	Envoyer une requête d'ami	—	—	Succès : 201 Échec : 404
POST	/friends/accept/:requestId	requestId: string	Accepter une requête d'ami	—	—	Succès : 201 Échec : 404
POST	/friends/reject/:requestId	requestId: string	Refuser une requête d'ami	—	—	Succès : 201 Échec : 404
DELETE	/friends/remove/:userId/	userId: string	Enlever un	—	—	Succès :

	:friendId	friendId: string	ami			200 Échec : 404
POST	/friends/:userId/ :blockedUserId	userId: string blockedUserId : string	Bloquer un utilisateur n'ayant pas fait de demande d'ami	—	—	Succès : 201 Échec : 404
POST	/friends/blockFriend/:userId/ :blockedFriendId	userId: string blockedUserId : string	Bloquer un ami	—	—	Succès : 201 Échec : 404
POST	/friends/blockUserWithPendingRequest/:userId/:otherUserId	userId: string otherUserId: string	Bloquer un quelqu'un qui nous a envoyé une demande d'ami	—	—	Succès : 201 Échec : 404

3.2 Paquets WebSockets

3.2.1 - Paquet Websockets Clavardage

Événement	Source	Description	Données	Événements potentiellement déclenchés
sendMessage	Client	Envoyer un message dans le Clavardage	Message	chatMessage
chatMessage	Serveur	Notifier de la réception d'un nouveau message dans le Clavardage	Message	-

sendVoiceMessage	Client	Envoyer un message vocal dans le Clavardage	VoiceMessage	voiceMessageSent
voiceMessageSent	Serveur	Notifier de la réception d'un nouveau message vocale dans le Clavardage	VoiceMessage	-
getChatRoomHistory	Client	Envoyer une requête pour avoir les messages de la salle de Clavardage moins récents	EarliestMessage	oldChatRoomMessage/ oldChatRoomMessageFail
oldChatRoomMessage	Serveur	Envoyer les messages envoyés précédemment dans la salle de Clavardage au client	PastMessages	-
oldChatRoomMessageFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de l'accès aux messages de la salle de Clavardage	{errorMessage: String}	-
createChatRoom	Client	Envoyer une requête pour créer une nouvelle salle de Clavardage	{chatRoomId: string}	newChatRoom/ newChatRoomFail
newChatRoom	Serveur	Informer les clients de l'ouverture d'une nouvelle salle de Clavardage	{chatRoomId: string}	-
newChatRoomFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de la création de la salle de Clavardage	{errorMessage: String}	-
closeChatRoom	Client	Envoyer une requête pour créer une nouvelle salle de Clavardage	{chatRoomId: string}	chatRoomClosed/ chatRoomClosedFail
chatRoomClosed	Serveur	Informer les clients de la	{chatRoomId: string}	-

		fermeture d'une salle de Clavardage		
chatRoomClosedFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de la fermeture d'une salle de Clavardage	{errorMessage: String}	-
joinChatRoom	Client	Envoyer une requête pour rejoindre une nouvelle salle de Clavardage	{chatRoomId: string}	chatRoomJoined/ chatRoomJoinedFail
chatRoomJoined	Serveur	Envoyer au joueur les informations liées à la salle de Clavardage	Channel	-
chatRoomJoinedFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de son ajout dans une salle de Clavardage	{errorMessage: String}	-
leaveChatRoom	Client	Envoyer une requête pour quitter une salle de Clavardage	{chatRoomId: string}	chatRoomLeft/ chatRoomLeftFail
chatRoomLeft	Serveur	Informer le cleint du succès de la requête	-	-
chatRoomLeftFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de son départ de la salle de Clavardage	{errorMessage: String}	-
switchToIntegratedChat	Client	Informer le serveur du changement de l'interface de clavardage	-	integratedChatSwitched/ integratedChatSwitchedFail
integratedChatSwitched	Serveur	Informer le client du succès de la requête	-	-

integratedChatSwitchedFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de son changement d'interface de Clavardage	-	-
switchToWindowedChat	Client	Informer le serveur du changement de l'interface de clavardage	-	windowedChatSwitched/ windowedChatSwitchedFail
windowedChatSwitched	Serveur	Informer le clint du succès de la requête	-	-
windowedChatSwitchedFail	Serveur	Informer le client ayant fait la requête d'une erreur lors de son changement d'interface de Clavardage	-	-
requestFriend	Client	Requête pour envoyer une demande d'ami	Account	friendRequested
friendRequested	Serveur	Notifier de la demande d'ami	Account	—

addFriend	Client	Requête pour envoyer l'acceptation/refus de la demande d'ami	{
			Account
			accept: boolean
			}
				friendAdded

friendAdded	Serveur	Notifier de l'acceptation/ refus de la demande d'ami	Account	—
blockAccount	Client	Requête pour bloquer un utilisateur	Account	accountBlocked
accountBlocked	Serveur	Enlever le compte de la liste d'ami	Account	—

3.2.2 - Paquet Websockets Match

Événement	Source	Description	Données	Événements potentiellement déclenchés
getMatchRooms	Client	Envoyer une requête pour avoir toutes les Matchs ouverts	-	matchRoomsSent
matchRoomsSent	Serveur	Envoyer les Matchs disponibles	MatchRoom[]	-
matchRoomOpened	Serveur	Notifier de l'ajout d'une nouvelle salle de Match	MatchRoom	-
matchRoomUpdated	Serveur	Notifier d'un changement au niveau de la salle de Match	MatchRoom	-
matchRoomClosed	Serveur	Notifier de la fermeture d'une salle de Match	{matchRoomId: string}	-
matchRoomStarted	Serveur	Notifier du départ d'un Match	{matchRoomId: string}	-
joinMatchRoom	Client	Demander à rejoindre la salle d'un match	PlayerRequest	newPlayer/newObserver, chatRoomJoined
newPlayer	Serveur	Notifier l'ajout d'un joueur dans la salle d'attente d'un match	Player[]	-
newObserver	Serveur	Notifier l'ajout d'un observateur dans la partie	Observer[]	
createTeam	Client	Envoyer une requête pour créer une équipe dans la salle d'attendre d'un match	{teamName: string?}	teamCreated
teamCreated	Serveur	Notifier l'ajout d'une	{	-

		équipe d'un joueur dans la salle d'attente d'un match	teamName: string? playerName: string }	
switchPlayerTeam	Client	Envoyer une requête pour changer d'équipe dans la salle d'attendre d'un match	{teamName: string?}	playerTeamSwitched

playerTeamSwitched	Serveur	Notifier le changement d'équipe d'un joueur dans la salle d'attente d'un match	{
			teamName: string?
			playerName: string
			}
				-

switchQuestion	Client	Envoyer aux joueurs une demande pour passer à la prochaine question	Room	nextQuestion
nextQuestion	Serveur	Notifier les joueurs du passage à la prochaine question	-	-
updateAnswer	Client	Envoyer une demande de changement des réponses d'un joueur après le changement de sélection des choix de réponses	UpdateAnswerRequest	answerUpdated
answerUpdated	Serveur	Notifier l'organisateur d'un changement dans la sélection de choix de réponse d'un joueur	PlayerAnswers[]	-
startTimer	Client	Envoyer une demande au serveur pour démarrer le timer	TimerRequest	newTime
newTime	Serveur	Notifier les joueurs et organisateur du changement de la valeur	TimerRequest	-

		du timer à chaque seconde		
stopTimer	Client	Envoyer une demande su serveur pour arrêter le timer	Room	-
cancelGame	Client	Envoyer une demande d'annulation du match par l'organisateur à partir de la salle d'attente	Room	-
gameCanceled	Serveur	Notifier les joueurs de l'annulation du match par l'organisateur dans la salle d'attente	-	-
finishMatch	Client	Envoyer une demande de la fin du match par l'organisateur durant la partie	Room	matchFinished, chatRoomLeft
matchFinished	Serveur	Notifier les joueurs de la fin du match par l'organisateur durant la partie	-	-
beginMatch	Client	Envoyer par l'organisateur une demande aux joueurs pour joindre le match commencé	Room	joinMatch
joinMatch	Serveur	Notifier les joueurs du début du match	Match	-
removePlayer	Client	Envoyer une demande pour retirer un joueur du	PlayerRequest	playerRemoved, chatRoomLeft

		match		
playerRemoved	Serveur	Notifier les joueurs et l'organisateur du départ d'un joueur	Player[]	-
updatePlayerScore	Client	Envoyer une demande pour mettre à jour le score d'un joueur dans le match stocké dans le serveur	QuestionRequest	updatedPlayerScore
updatedPlayerScore	Serveur	Notifier du changement de score pour un joueur afin de synchroniser les scores entre les matchs stockés côté serveur et côté client	Player	-
setFinalAnswer	Client	Envoyer qu'une réponse est finale	UpdateAnswerRequest	finalAnswerSet
finalAnswerSet	Serveur	Notifier l'organisateur que la réponse du joueur est finale	PlayerAnswers	-
playerLeftAfterMatchBegun	Client	Envoyer au serveur qu'un joueur a quitté le match après que ce dernier ait commencé	QuestionRequest	playerDisabled, chatRoomLeft
playerDisabled	Serveur	Notifier qu'un joueur n'est plus actif (a quitté le match pendant le déroulement de ce dernier)	Player	-
allPlayersResponded	Serveur	Notifier l'organisateur que tous les joueurs ont déclaré leurs réponses	-	-

		finales pour arrêter le timer et passer aux résultats de la question		

3.3 Interfaces

Nom	Description	Structure
Account	Interface pour l'information public d'un utilisateur	interface Account { _id: string; userId: string; pseudonym: string; email: string; avatarUrl: string; themeVisual: ThemeVisual; lang: Language; gamesPlayed: number; gamesWon: number; avgQuestionsCorrect: number; avgTimePerGame: number; matchHistory: PlayerMatchHistory[]; money: number; friends: string[]; friendRequests: FriendRequestData[]; friendsThatUserRequested: string[]; visualThemesOwned: ThemeVisual[]; avatarsUrlOwned: string[]; }
Answer	Interface pour les réponses	{ text: string; isCorrect: boolean; playerId: string; didPlayerAnswer: boolean; questionId: string; point: number; lastAnswerTime: Date; final: boolean; points: number; }
Channel	Interface qui contient les précédent	{

	chats, le nom de la salle de clavardage et le type de salle	messages: Message[] roomType: ChatRoomType owner: string players: string[] }

ChatRoomInfo	Interface qui contient les informations rudimentaires des channels	{
		messages: Message[]
		roomType: ChatRoomType
		}

ChatRoomType	Énum qui contient les divers types de salles. Pour référence :
	- general : Chat de base accessible par tous.
	- public : Chat créé par un utilisateur, accessible par tous.
	- match : Chat créé par une partie, accessible seulement par les personnes étant dans la partie.
		{
		General : “general”;
		Public : “public:
		Match : “match”
		}

ChatRoomMessageData	Interface étant utilisée pour échanger les messages entre les joueurs et le serveur	{
		chatRoomName: string,
		data: ChatRoomMessage
		}

ChatRoomMessage	Interface étant utilisée pour contenir les informations liées à un message	{
		userID: string,
		time: string,
		data: string
		}

ChatRoom	Interface contenant les informations d’un Channel, en incluant aussi le nom de ce dernier de manière à l’identifier	{ chatRoomName: string messages:ChatRoom Message[] roomType: ChatRoomType owner: string players: string[]

		}

ChatAccessibilityRequest	Interface étant utilisée pour activer ou désactiver les joueurs pouvant interagir avec le chat	{
		matchAccesCode: string
		name: string
		player: Player[]
		}

ChatRoomInfo	Interface servant à mettre en relation une room et son type	{
		chatRoomName: string,
		chatRoomType: ChatRoomType
		}

ChoiceCount	Interface servant à donner combien de choix ont été fait pour une réponse	{
		choice: Choice,
		nSelected: number
		}

Choice	Interface servant à contenir les informations liées à un des choix d'une réponse	{
		text: string,
		isCorrect: boolean
		}

CreateGameDTO	Informations utilisées dans la création d'un jeu	{ id: string; title: string; description: string; duration: number; lastModification: string; questions: Question[]; isVisible: boolean; creator: string, difficultyMap: {key: string, value: number}[] interestMap: {key: string, value: number}[] durationMap: {key: string, value:

		number}[] }

DisplayableMatchHistory	Interface pour montrer l'historique de partie	{
		matchAccesCode: string,
		bestScore: number,
		startTime: Date,
		nStartPlayer: number,
		gameName: string
		}

FriendRequestData	Interface pour communiquer les requêtes d'ami	{
		requestId: string,
		sendBasicInfo: Partial<Account>
		}

GameEvaluation	Interface pour communiquer les évaluations de partie	{
		difficulty: string,
		interest: string,
		duration: string,
		rating: string,
		gameId: string
		}

Game	Interface contenant toute l'information quant à une partie, incluant sa visibilité ainsi que ses évaluations.	{ id: string, title: string, description: string, duration: number, lastModification: string, questions: Question[], isVisible: boolean, creator: string, difficultyMap: {key: string, value: number}[] interestMap: {key: string, value: number}[] durationMap: {key: string, value:

		number}[] rating: {key: string, value: number}[] }

Language	Interface contenant toutes les langues admises au sein de l'application	{
		EN = 'en'
		FR = 'fr'
		}

Match	Interface contenant toutes les informations en lien avec le déroulement d'un Match, dont entre autre le prix, l'était du timer, les joueurs et les équipes présent dans le match.	{
		game: Game.
		begin: string,
		end: string,
		bestScore: number,
		accessCode: string,
		testing: boolean,
		players: Player[],
		observers: Observer[],
		managerName: string,
		managerId: string,
		isAccessible: boolean,
		isFriendMatch: boolean,
		bannedNames: string[],
		playerAnswers: PlayerAnswers[],
		panicMode: boolean,
		timer: number,
		timing: boolean,
		isTeamMatch: boolean,
		isPricedMatch: boolean,
		priceMatch: boolean,
		nbPlayersJoined: number,
		teams: Team[],
		currentQuestionIndex: number,
		isEvaluatingQrl: boolean
		}

MatchRoom	Interface de match pour la liste des matchs disponibles	{
		name: string;
		playerNum: number;
		accessCode: string;
		price: int
		}

MatchEvaluation	Interface pour l'évalutation d'un match	{
		{criteria: string
		choice: string
		}[];
		rating: number
		}

MatchHistory	Interface contenant les informations plus détaillées sur les diverses parties.	{
		matchAccesCode: string,
		bestScore: number,
		startTime: Date,
		nStartPlayer: number,
		gameNameL sting
		}

MatchRouteParams	Interface pour la création de partie pour décrire si elle est une partie en mode test ou non	{
		id: string,
		testing: boolean
		}

Message	Interface pour l'envoie de messages	{
		playerName: string,
		matchAccessCode: string,
		time: string,
		data: string,
		}

ObserverQuitRequest	Interface utilisé pour demander au serveur de quitter une partie lorsqu'on est un observateur	{
		observerName: string,
		accessCode: string
		}

Observer		{
		name: string,
		observerName: string
		}

Player	Interface des joueurs dans une partie	{
		name: string;
		isActive: boolean;
		score: number;
		nBonusObtained: number;
		}

PlayerAnswers	Interface pour communiquer les réponses d'un joueur	{
		name: string;
		QCManswers: Choice[];
		QRLanswer: string;
		QREanswer: number;
		lastAnswerTime: string;
		final: boolean;
		questionId: string;
		obtainedPoints: number;
		}

PlayerRequest	Interface pour transférer les données en lien avec le joueur pour les événements durant le déroulement de la partie	{
		roomId: string;
		name: string;
		hasPlayerLeft?: boolean;
		}

PlayerMatchHistory	Interface pour faire montrer les parties jouées.	{
		gameName: string,
		datePlayed: string,
		won: boolean
		}

Question	Interface des questions dans un jeu	{ id: string; type: string;

		text: string; points: number; choices: Answer[]; timeAllowed: number; correctAnswer: number | null, lowerBounds: number | null, upperBounds: number | null, tolerance: number | null, image: string?; }

QuestionRequest	Interface pour transférer les données nécessaires pour mettre à jour les informations pour une question durant une partie	{
		matchAccessCode: string;
		player: Player;
		questionId: string;
		}

QrlInteraction	Interface pour communiquer l'intéraction avec un question QRL	{
		player:Player,
		hasInteracted:boolean
		}

Room	Interface pour finir	{
		id: string;
		}

SocketToUser	Interface contenant un mapping entre le socketId et le userId	{
		socketId: string,
		userId: string,
		}

StatisticsProfilDto	Interface contenant les statistiques des joueurs	{
		numberPlayedGames: number,
		numberWonGames: number,
		averageCorrectAnswers: number,
		averageTimePerGameMinute:
		number avnumberPlayedGames:
		}

StopServerTimeRequest		{
		roomId: string,
		isHistogramTimer: boolean
		}

ThemeVisual	Interface contenant les diverses valeurs des thèmes	{
		DARK = ‘dark’,
		LIGHT = ‘light’
		CHRISTMAS = ‘christmas’
		VALENTINE = ‘valentines’
		}

TimerRequest	Interface pour transférer les données relatives au temps	{
		roomId: string;
		timer: number;
		timeInterval?: number;
		}

Token	Interface contenant le token pour s’identifier auprès du serveur	{ token: string }

UpdateAnswerRequest	Interface pour transférer les données relatives aux réponses des joueurs	{
		matchAccessCode: string;
		playerAnswers: PlayersAnswers
		}

UpdateMatch	Interface pour mettre à jour une valeur dans un match	{
		accessCode: string;
		player: Player;
		observers: boolean;
		}

Validation	Interface pour valider si le nom d’un joueur existe	{
		accessCode: string;
		name: string;
		}