Fais-moi un dessin

Protocole de communication

Version 2.0

Historique des révisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Auteur** |
| 2021-02-09 | 1.0 | Première ébauche du document | Augustin Bouchard |
| 2021-02-10 | 1.1 | Ajout des deux diagrammes + texte dans la section communication client-serveur | Simon Ayotte |
| 2021-02-17 | 1.2 | Ajout plusieurs descriptions de paquets | Guilhem Dubois |
| 2021-02-18 | 1.3 | Ajout diagrammes paquets + diagrammes web sockets | Simon Ayotte |
| 2021-02-19 | 1.4 | Version finale avant la remise d’appel d’offre | Équipe 205 |
| 2021-04-19 | 2.0 | Version finale avant la remise finale | Équipe 205 |

Table des matières

[**1. Introduction**](#_heading=h.30j0zll) **5**

[**2. Communication client-serveur**](#_heading=h.1fob9te) **5**

[2.1 REST API, CRUD et protocole HTTP](#_heading=h.3hcgn5u4gsvf) 5

[2.2 Communication avec WebSockets](#_heading=h.i5549tuxceoy) 6

[**3. Description des paquets**](#_heading=h.gnzalzenq40s) **8**

[3.1 Connexion de l’utilisateur](#_heading=h.13ysyahh06fm) 8

[3.2 Création de la fenêtre externe](#_heading=h.yqhi07xxbksk) 9

[3.3 Inscription d’un utilisateur](#_heading=h.6r45f5lwsa8l) 10

[3.4 Déconnexion d’un utilisateur](#_heading=h.k8yej8kf1dsx) 11

[3.5 Canaux de clavardage](#_heading=h.534uj5tc52wq) 13

[3.5.1 Réception de l’historique des messages d’un canal de discussion](#_heading=h.rgpoz7ou8dh4) 13

[3.5.2 Réception des canaux de discussion](#_heading=h.5zo2n17kk5pr) 13

[3.5.3 Création d’un canal de discussion](#_heading=h.6o60othryqh4) 14

[3.5.4 Suppression d’un canal de discussion](#_heading=h.x2e2urtgvlwj) 16

[3.5.5 Joindre un canal de discussion](#_heading=h.pqa2nlsscezj) 16

[3.5.6 Quitter un canal de discussion](#_heading=h.l0gdx9d0tfyh) 18

[3.5.7 Envoi/Réception de messages](#_heading=h.nhx1efq9jeb) 20

[3.6 Lobbies](#_heading=h.ttbudcsecggc) 22

[3.6.1 Réception des lobbies disponibles](#_heading=h.7wo0u3k6ik0f) 22

[3.6.2 Création d’un lobby](#_heading=h.llho8x8hplg2) 23

[3.6.3 Suppression d’un lobby](#_heading=h.bqnzq6qn3hpx) 23

[3.6.4 Joindre un lobby](#_heading=h.vk013ob3qb1g) 25

[3.6.5 Quitter un lobby](#_heading=h.ne9dbn7gkwlz) 26

[3.7 États dans une partie](#_heading=h.jd78w398420l) 26

[3.7.1 Début d’une partie](#_heading=h.psn3zvxptfn6) 26

[3.7.2 L’utilisateur devine le mot dessiné](#_heading=h.qgw8d4zc70cu) 27

[3.7.3 L’utilisateur demande un indice pour le mot dessiné](#_heading=h.rg33h96cd3ug) 28

[3.7.4 L’utilisateur envoie le signal qu’il est prêt à débuter la nouvelle round dans la partie](#_heading=h.2qpqnb4o1eod) 29

[3.7.5 L’artiste envoie le signal qu’il est prêt](#_heading=h.8j1e7x4n9z46) 31

[3.7.6 L’utilisateur envoie un J’aime au dessinateur](#_heading=h.e5rvobvgbp5o) 32

[3.7.7 L’utilisateur envoie un Je n’aime pas au dessinateur](#_heading=h.kbdwnxht2nsd) 32

[3.7.8 Fin d’une manche](#_heading=h.9svlew2xg26i) 33

[3.7.9 Fin d’une partie](#_heading=h.eda9673110og) 34

[3.8 Envoie des dessins dans la partie](#_heading=h.ybo1yjnovrc6) 34

[3.8.1 Premier point d’un trait](#_heading=h.4lk8c6o5xrtn) 34

[3.8.2 Point au milieu d’un trait](#_heading=h.g3xhzg6mdmjs) 35

[3.8.3 Dernier point d’un trait](#_heading=h.wdiojfvpug23) 36

[3.8.4 Effacer un trait dans le dessin](#_heading=h.5hzrklbv6ms7) 37

[3.8.5 Annuler le dernier trait](#_heading=h.v4wh0waxnm5o) 38

[3.8.6 Refaire du dernier trait annulé](#_heading=h.qm8a0ec4enn5) 39

[3.9 Paire mot-image](#_heading=h.89snsy1u5z3r) 40

[3.9.1 Envoie des infos de dessin paire-mot-image](#_heading=h.8lb7jm5zk0oa) 40

[3.9.2 Envoie d’un path paire-mot-image](#_heading=h.pg2dvnucbmpa) 40

[3.9.3 Mot quickdraw](#_heading=h.xsnugizi07bl) 40

[3.10 Leaderboard](#_heading=h.frnup0ec08c) 41

[3.11 Profile](#_heading=h.ag628hmv3s9q) 42

[3.11.1 Identité de l’utilisateur](#_heading=h.b2xxddvkbi9b) 42

[3.11.2 Historique de connexion](#_heading=h.r83dazhju7hz) 43

[3.11.3 Historique de partie](#_heading=h.fckutc2ynht9) 43

[3.11.4 Statistiques](#_heading=h.drq6l22jnao) 45

Protocole de communication

# 1. Introduction

Le présent document présente le protocole de communication de notre application *Fais moi un dessin.* On y présente des diagrammes facilitant la compréhension des communications client-serveur.

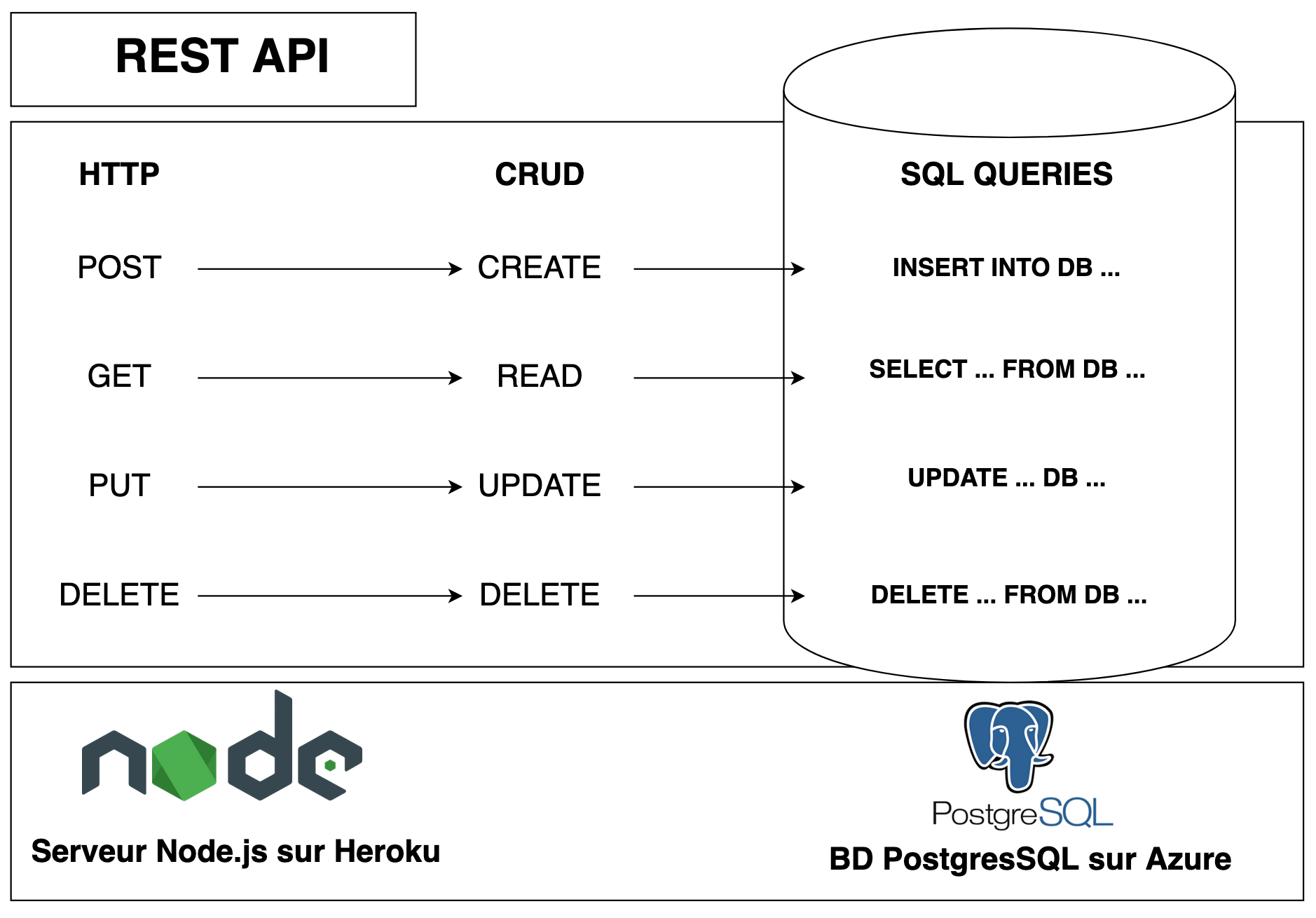
Dans la partie sur 2. sur la Communication client-serveur il sera question de l’architecture générale du protocole de communication de notre application ainsi que des principes fondamentaux pour la communication comme l’utilisation des WebSockets avec la librairie Socket.Io ainsi que de l’utilsatoin des requêtes HTTP.

La dernière section du document servira à illustrer comment les principes fondamentaux expliqués dans la section 2 seront utilisés concrètement dans notre application pour implémenter les divers fonctionnalités requises.

# 2. Communication client-serveur

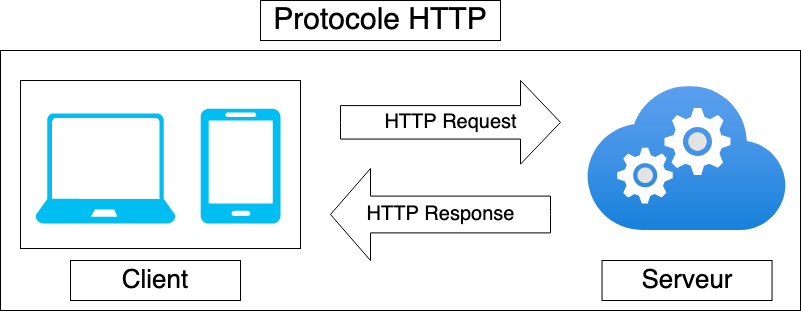
## 2.1 REST API, CRUD et protocole HTTP

La communication entre le serveur et les clients se fait principalement avec les requêtes HTTP (POST, GET, PUT, DELETE) qui sert d’interface entre notre serveur node.js et les clients (Electron/Android). Ces opérations font partie de la famille des opérations CRUD (CREATE, READ UPDATE, DELETE).



Le client peut donc envoyer des informations au serveur par l’entremise des requêtes POST. Il peut également recevoir de l’information du serveur par l’entremise de requête GET. Les requêtes sont par la suite gérer par le serveur qui s’assure de valider et d’envoyer des queries à la BD PostgreSQL si il le faut. Les requêtes suivent donc le protocole HTTP.

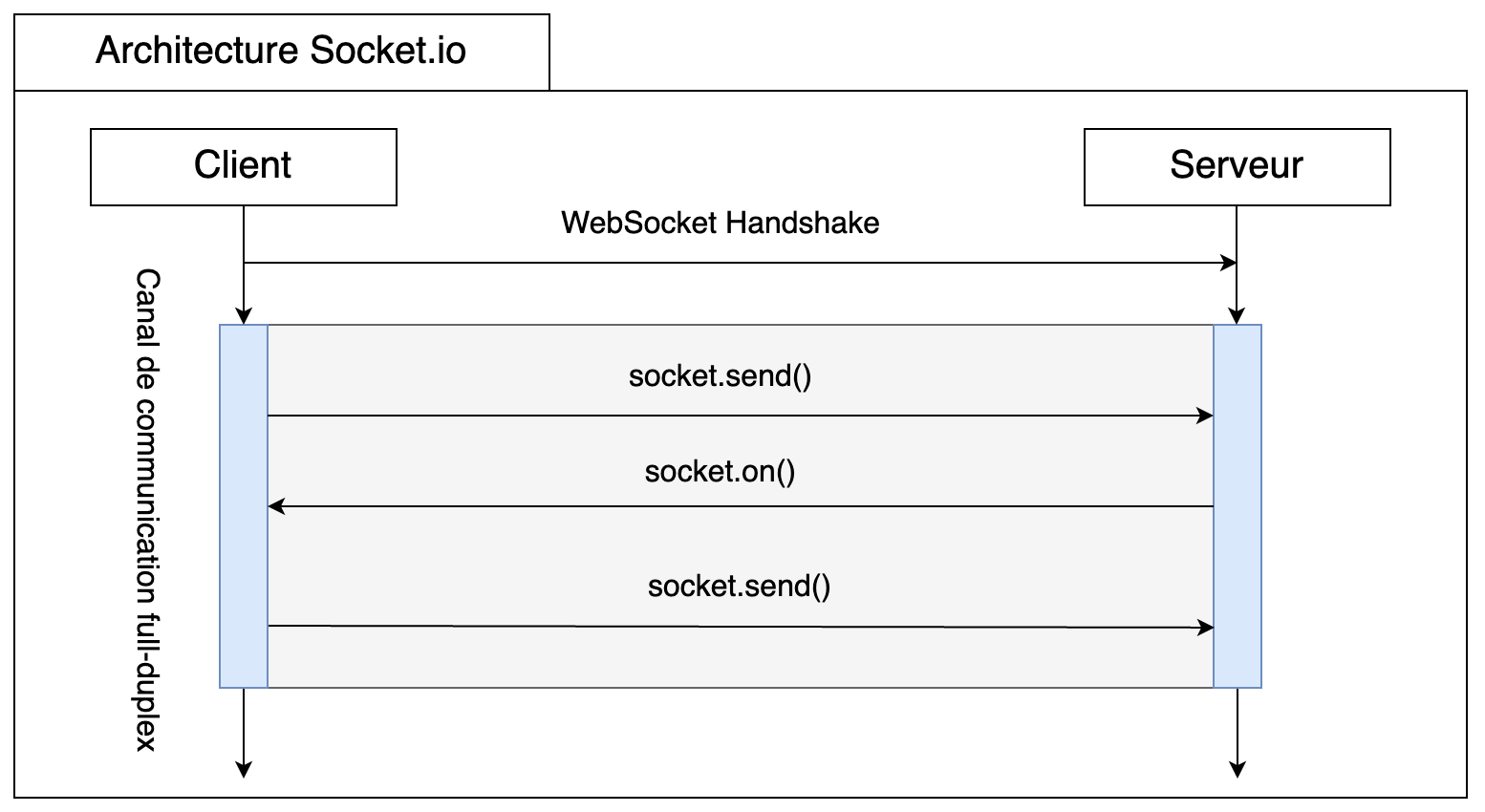
Pour la majorité des opérations entre les clients et le serveur où le client a besoin d’un accès à une information du serveur ou bien d’une validation du serveur, nous allons utiliser les ree



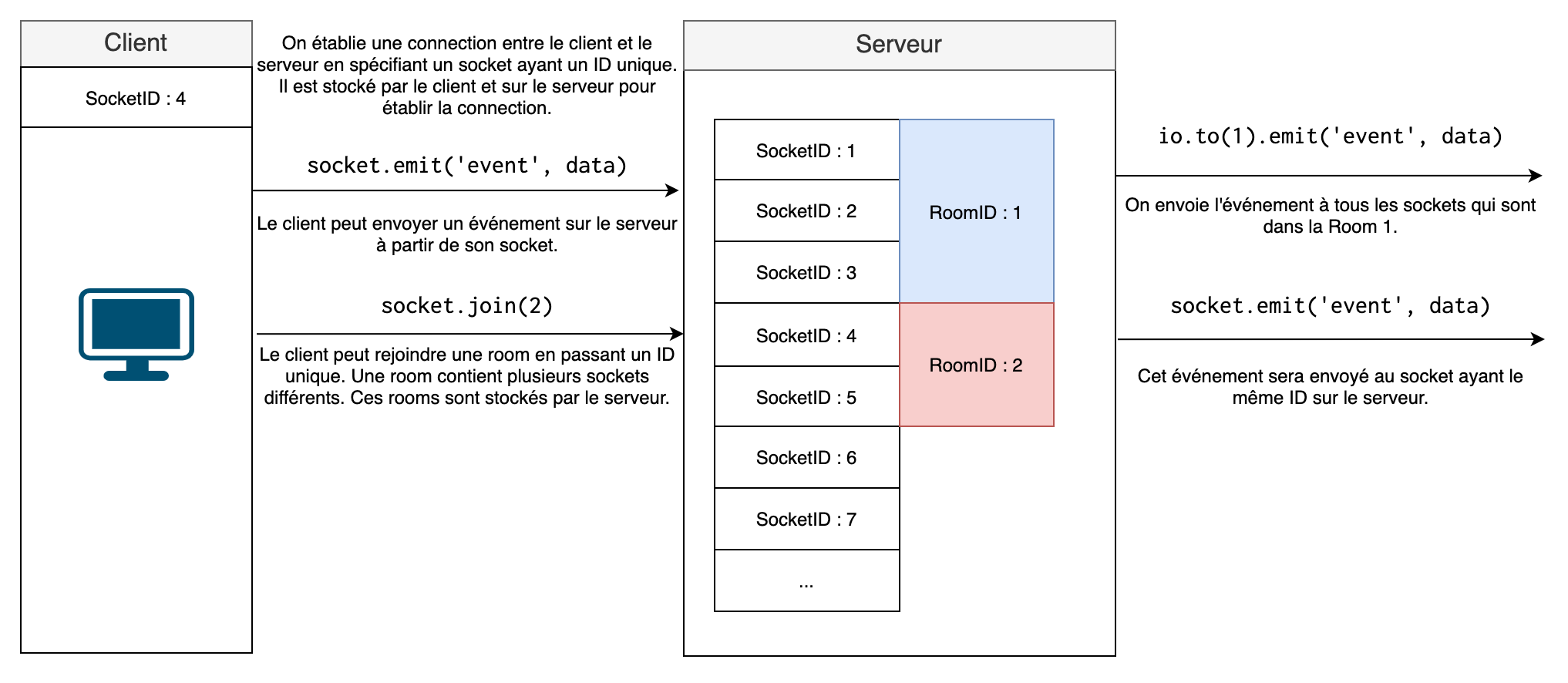
# 

## 2.2 Communication avec WebSockets

Lorsque nous avons besoin d’effectuer plusieurs opérations de communication entre le client et le serveur dans un court laps de temps, nous optons plutôt pour l’utilisation des WebSockets dans notre application à l’aide de la librairie [Socket.io](https://socket.io/docs/v3/). Nous allons particulièrement faire usage des WebSockets pour la fonctionnalité du clavardage ainsi que pour la transmission en temps réel des éléments SVG sur le canevas de dessin dans une partie.



Les WebSockets nous permettent d'établir un canal de communication full-duplex entre le client et le serveur ce qui facilite la transmission de données pour certaines fonctionnalités. Socket.io fonctionne par événements, le client envoie donc des données sous la forme d'événements qui sont écoutés par le serveur. Par la suite, le serveur peut répondre au client.



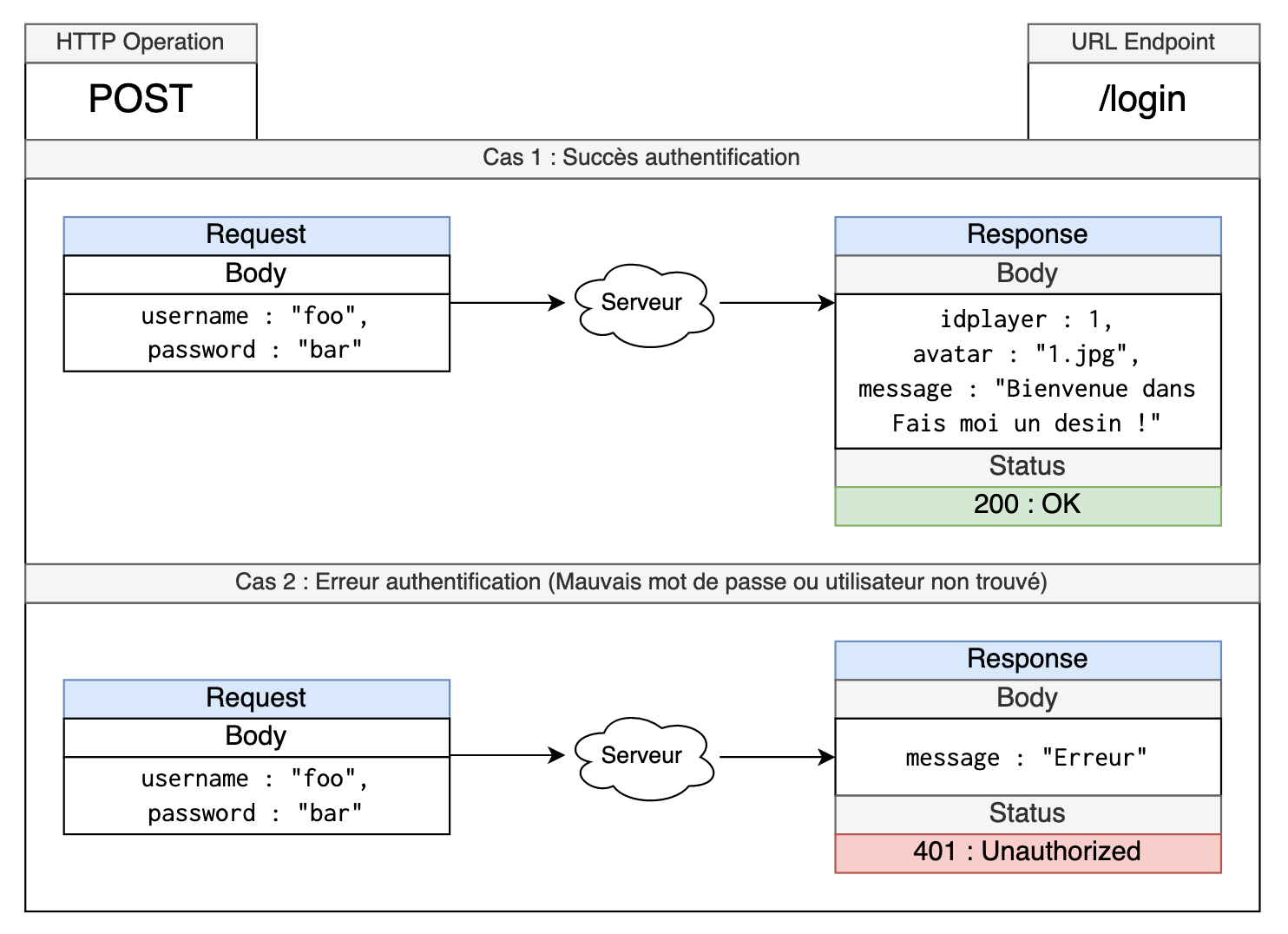
Il est aussi important de parler du concept de Room qui nous sera bien utile pour les fonctionnalités de Fais-moi un dessin. En effet, il est possible de créer une room contenant plusieurs sockets différents afin d’envoyer des événements seulement au groupe de sockets faisant partie de la room. Nous allons principalement utiliser les Room pour encapsuler les parties ainsi que les canaux de discussion. De cette façon, les événements d’une partie seront seulement envoyés aux clients faisant partie de la partie.

# 

# 3. Description des paquets

Pour l'envoi des requêtes et des réponses HTTP, notre application privilégie le format JSON comme structure de communication étant donné que c’est une structure extrêmement flexible et facile d’utilisation, particulièrement avec le serveur et le client lourd qui sont écrit en TypeScript.

## 3.1 Connexion de l’utilisateur

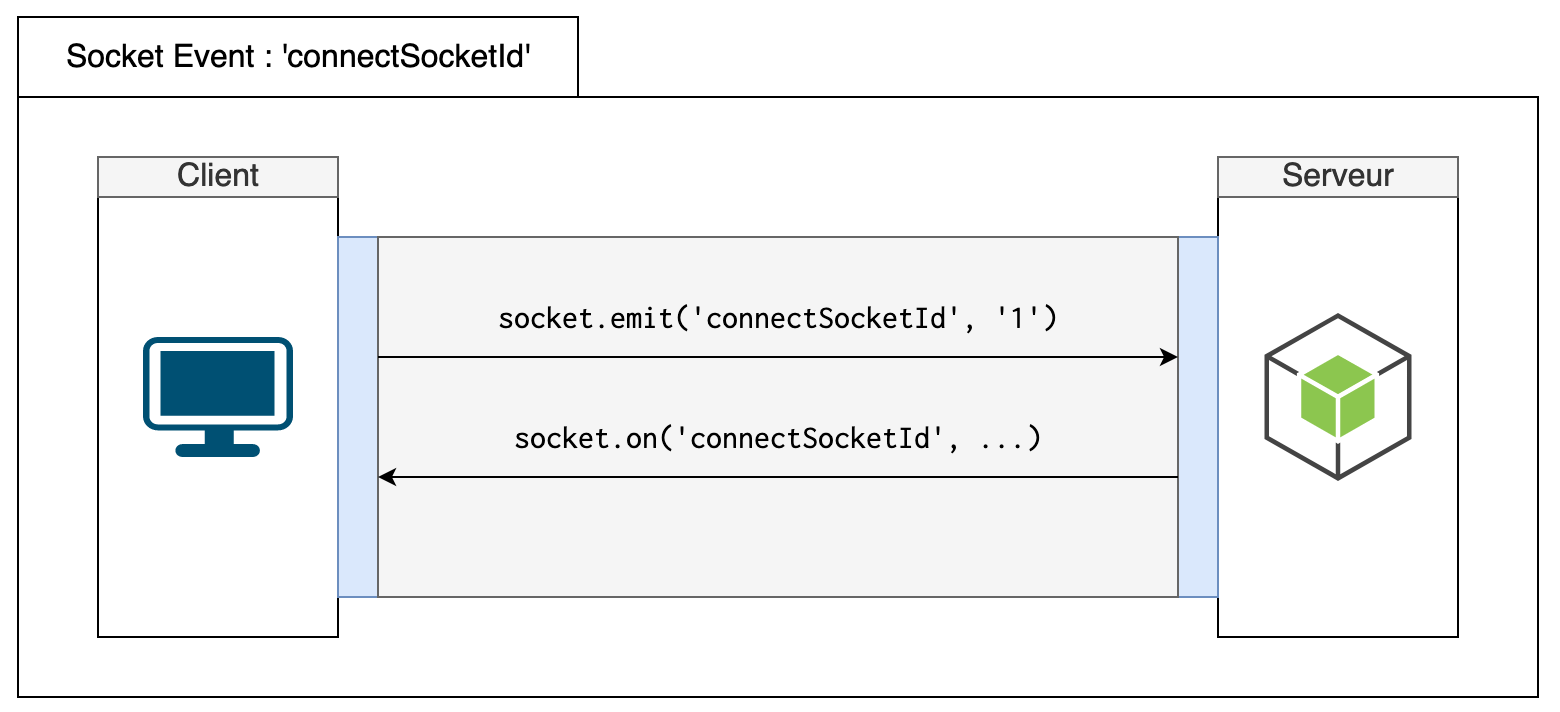


Une fois confirmé, le client démarre une connexion par WebSocket au serveur et envoie un événement afin que le serveur puisse associer la connexion à l’utilisateur:

Événement: connectSocketid

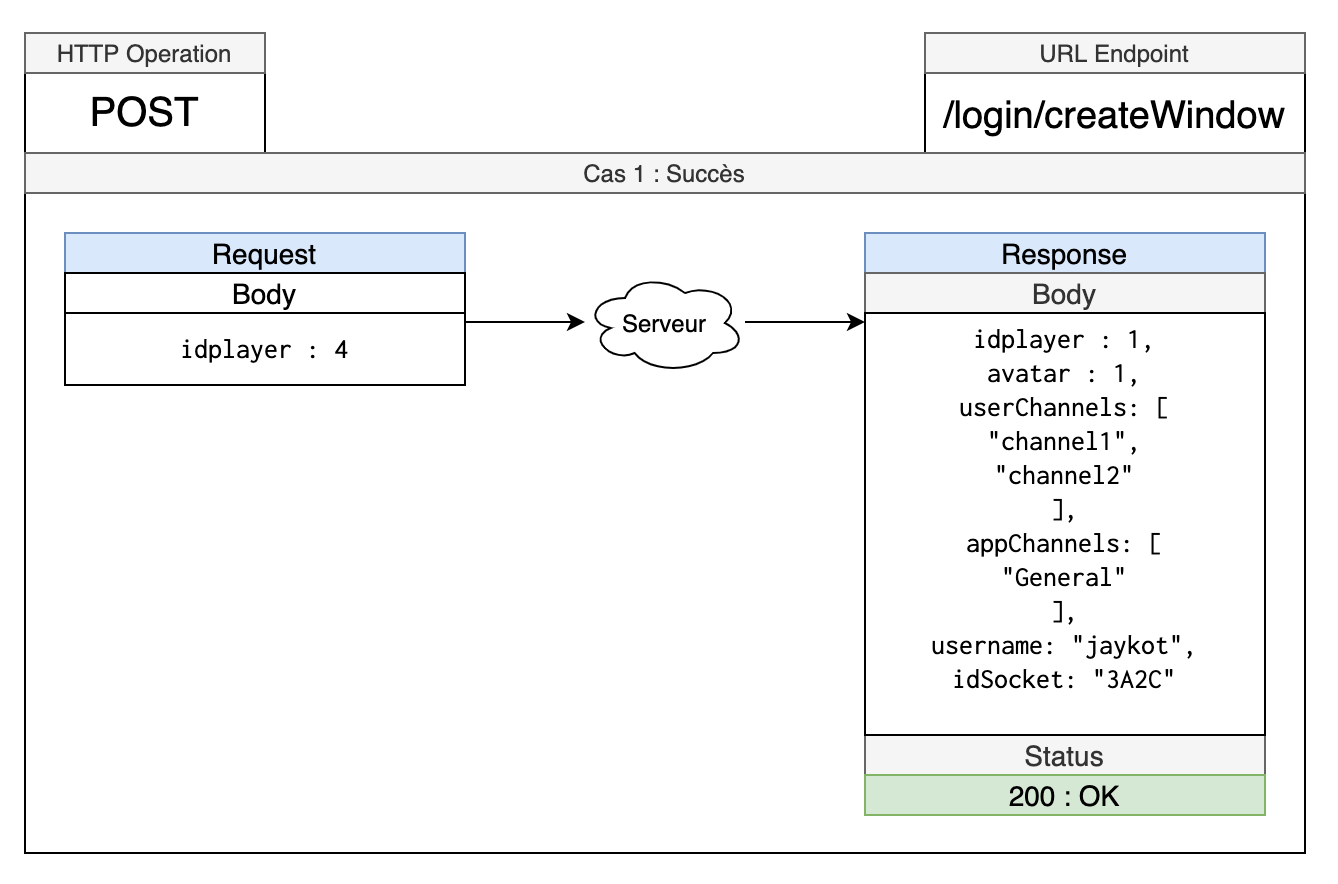
Méthode de requête: WebSocket

Data: idplayer: identifiant de l’utilisateur



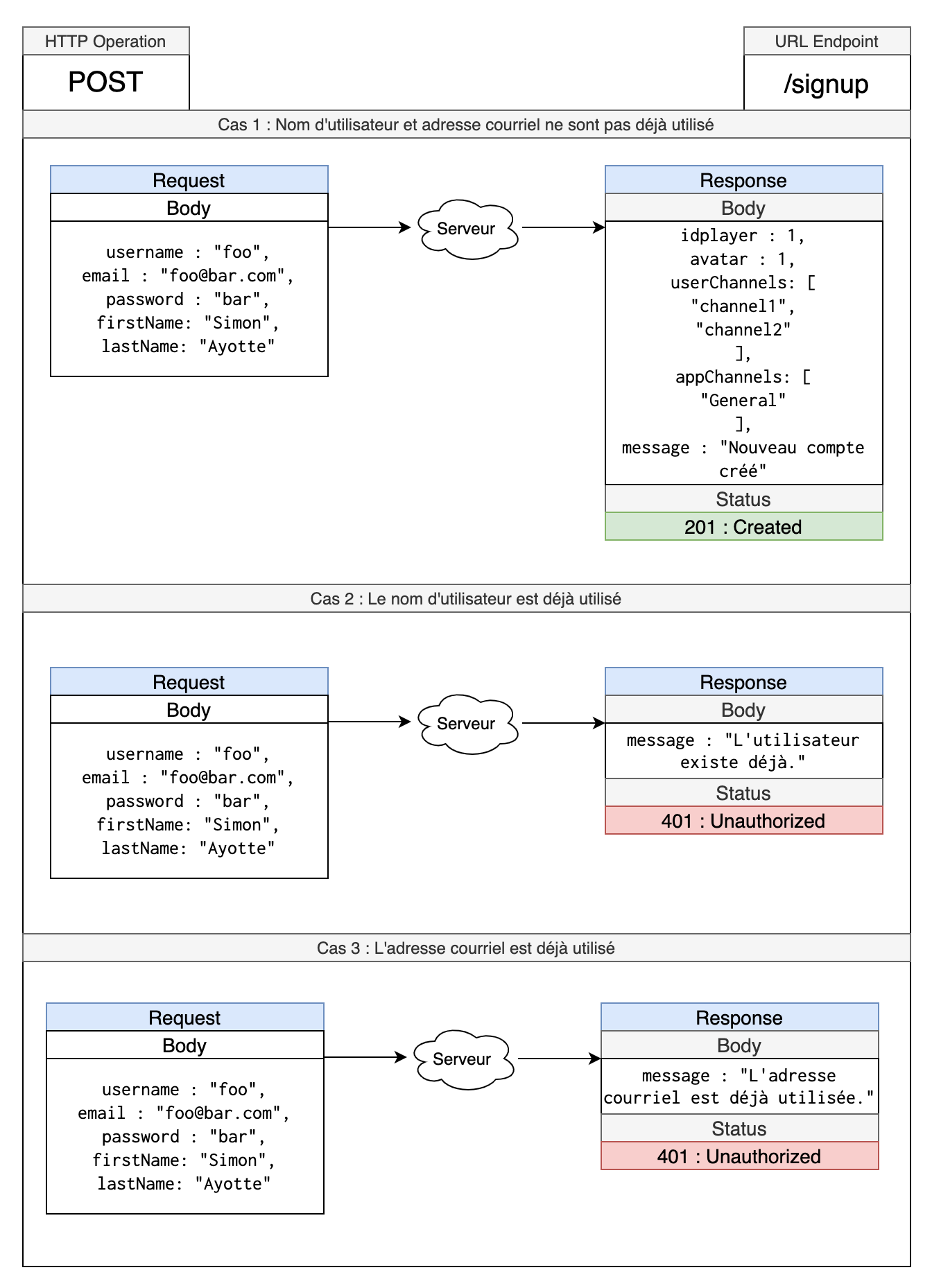
Par la suite, le serveur reconnaîtra que tout message envoyé par cette connexion WebSocket correspond à cet utilisateur.

## 3.2 Création de la fenêtre externe



Cette requête nous permet d’ouvrir un nouvelle instance de l’application pour le mode de fenêtrage externe avec les données nécessaires sans devoir passer par le sign-in.

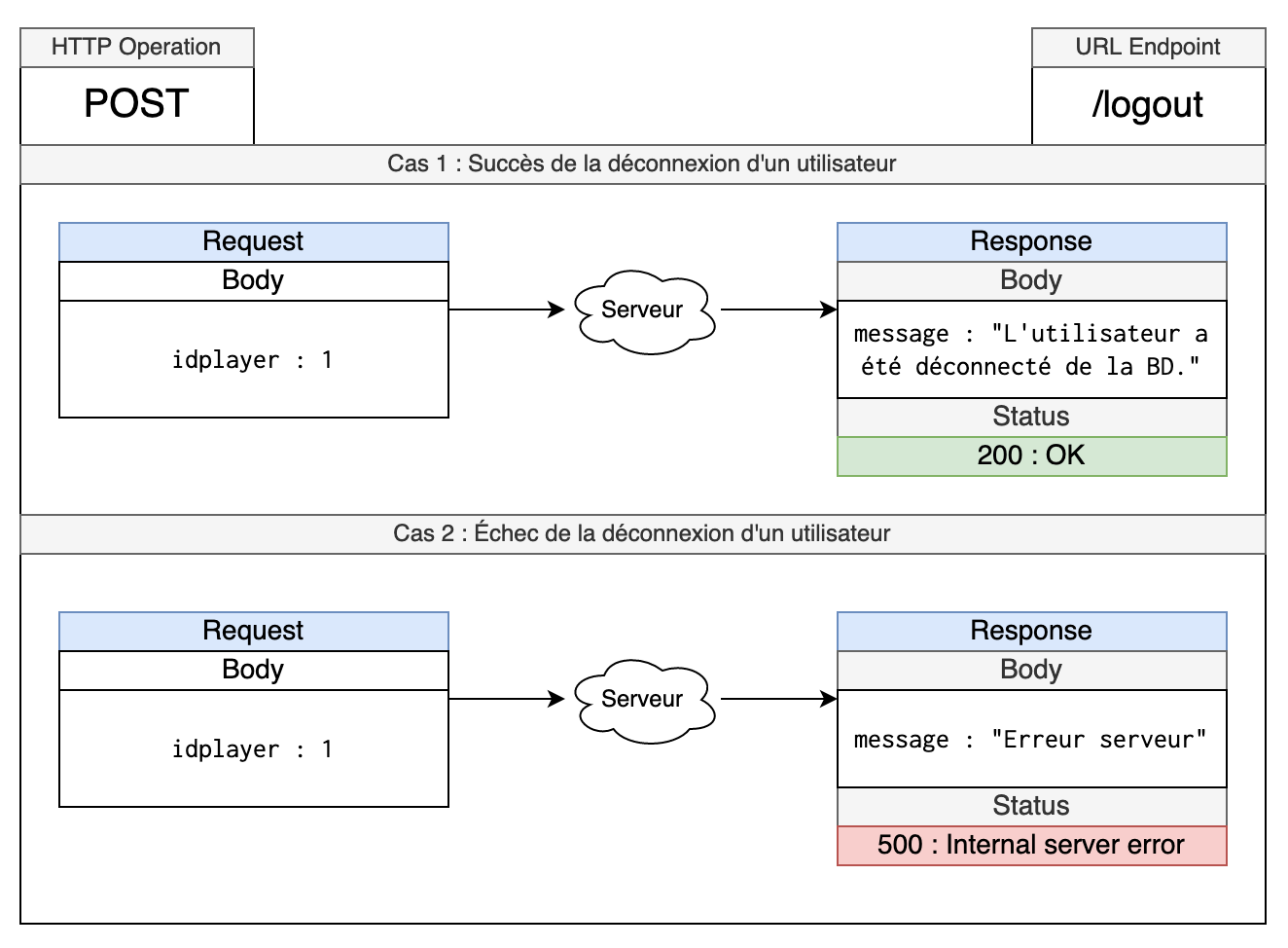
## 3.3 Inscription d’un utilisateur



L’événement “connectSocketid” est ensuite envoyé tel qu’expliqué dans la section précédente.

## 3.4 Déconnexion d’un utilisateur

Quand l’utilisateur se déconnecte du serveur manuellement, on effectue une requête POST pour changer son status de connexion dans la base de données.



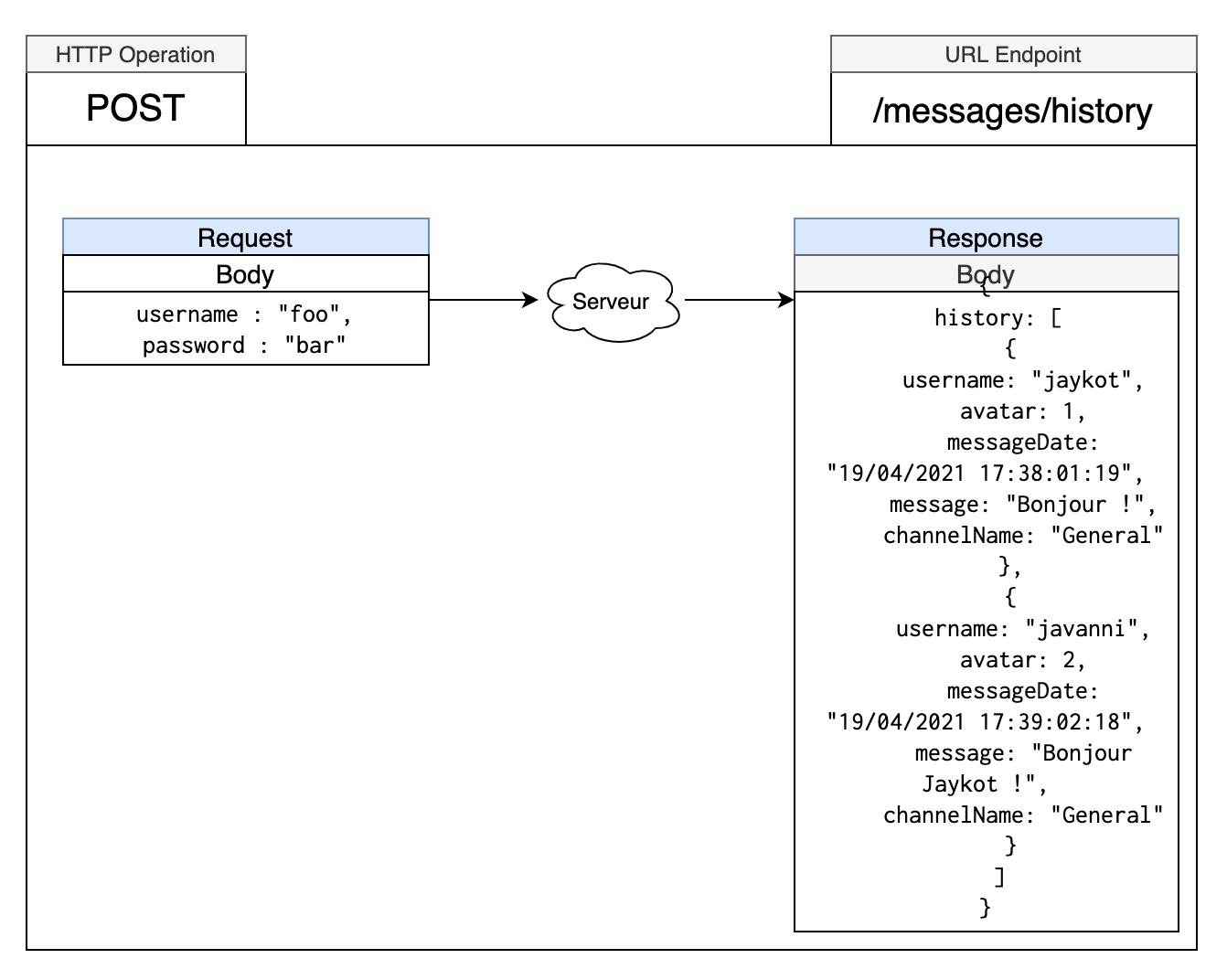
Lorsque l’utilisateur se déconnecte automatiquement (par exemple , en fermant la fenêtre du client lourd), un événement socket est automatiquement appelé où l’on termine la connexion du client au socket.



## 

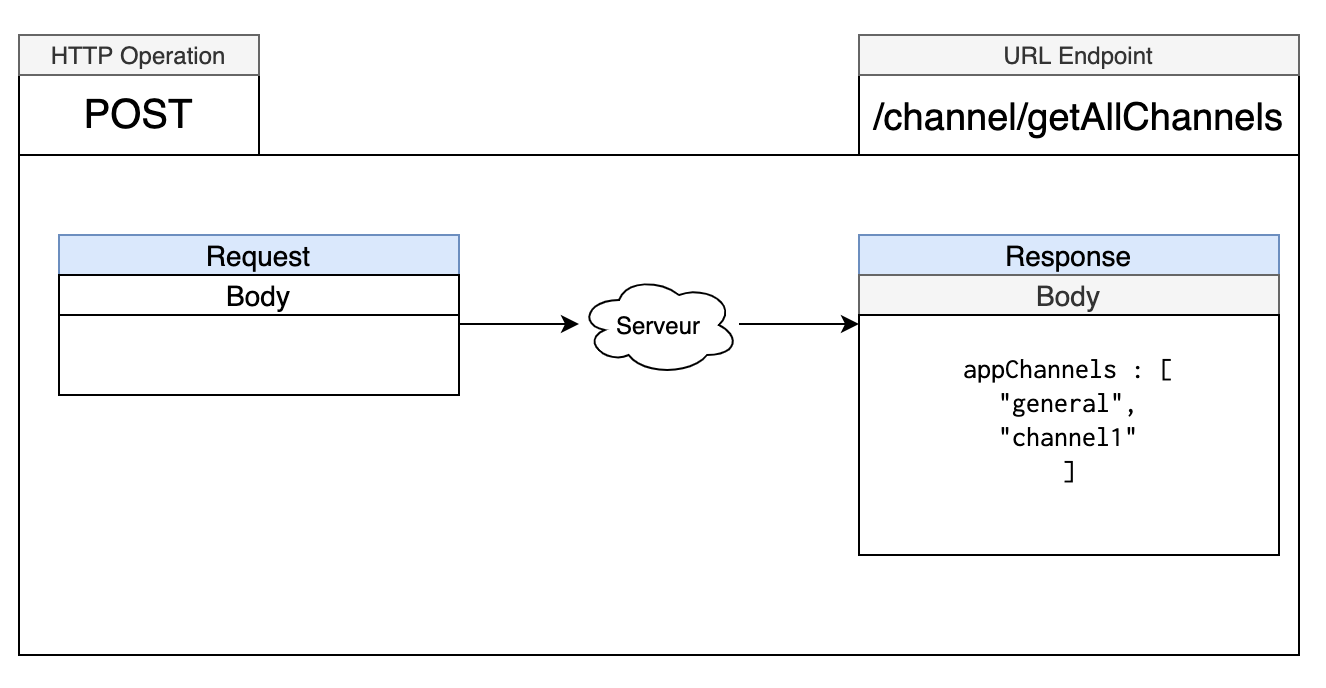
## 3.5 Canaux de clavardage

## 3.5.1 Réception de l’historique des messages d’un canal de discussion



## 3.5.2 Réception des canaux de discussion

Cette requête permet d’avoir tous les canaux de discussion actifs sur le serveur.

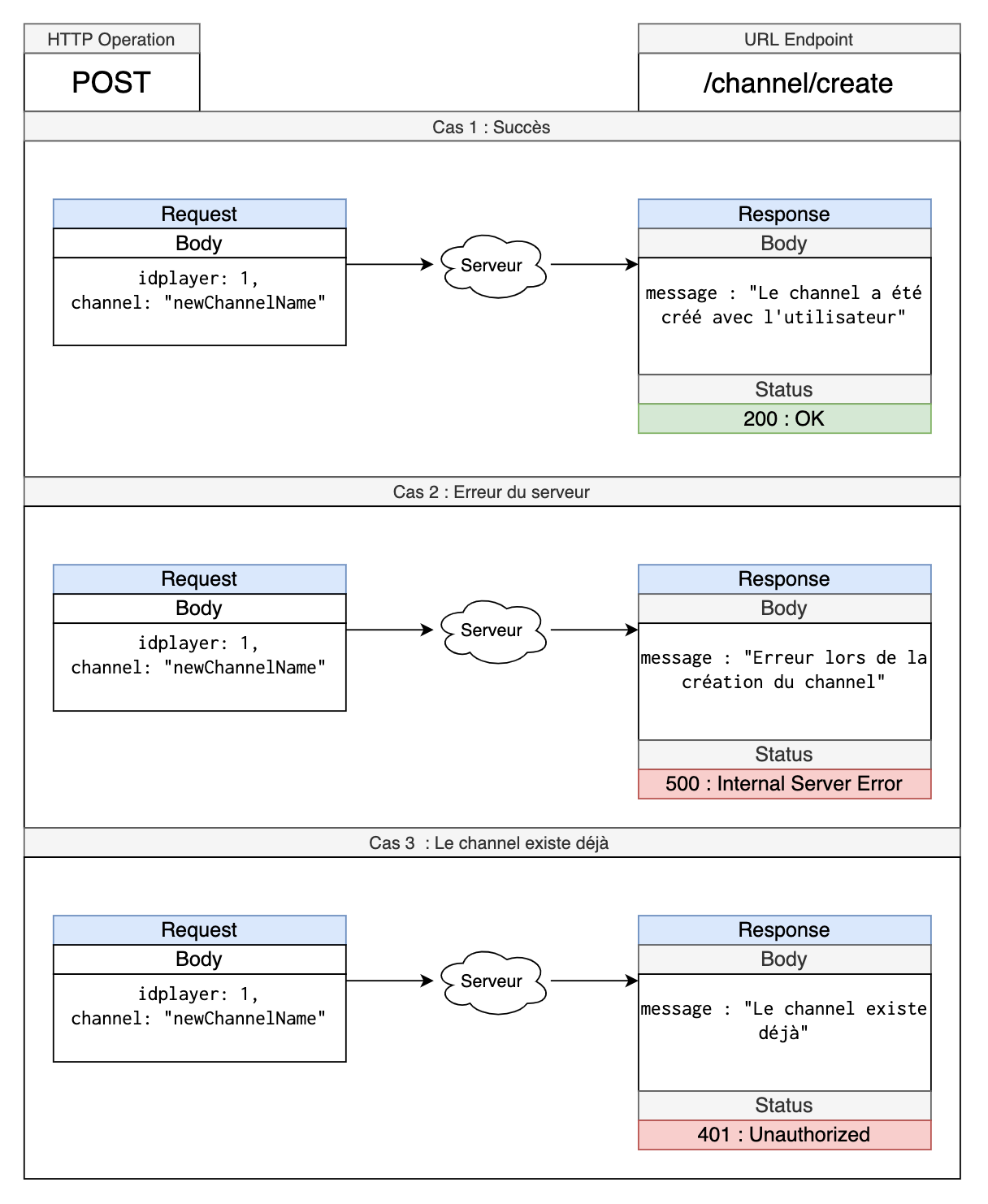


Pour chaque canal, le client écoute un événement pour recevoir les messages du canal (voir section 3.9).

## 

## 3.5.3 Création d’un canal de discussion

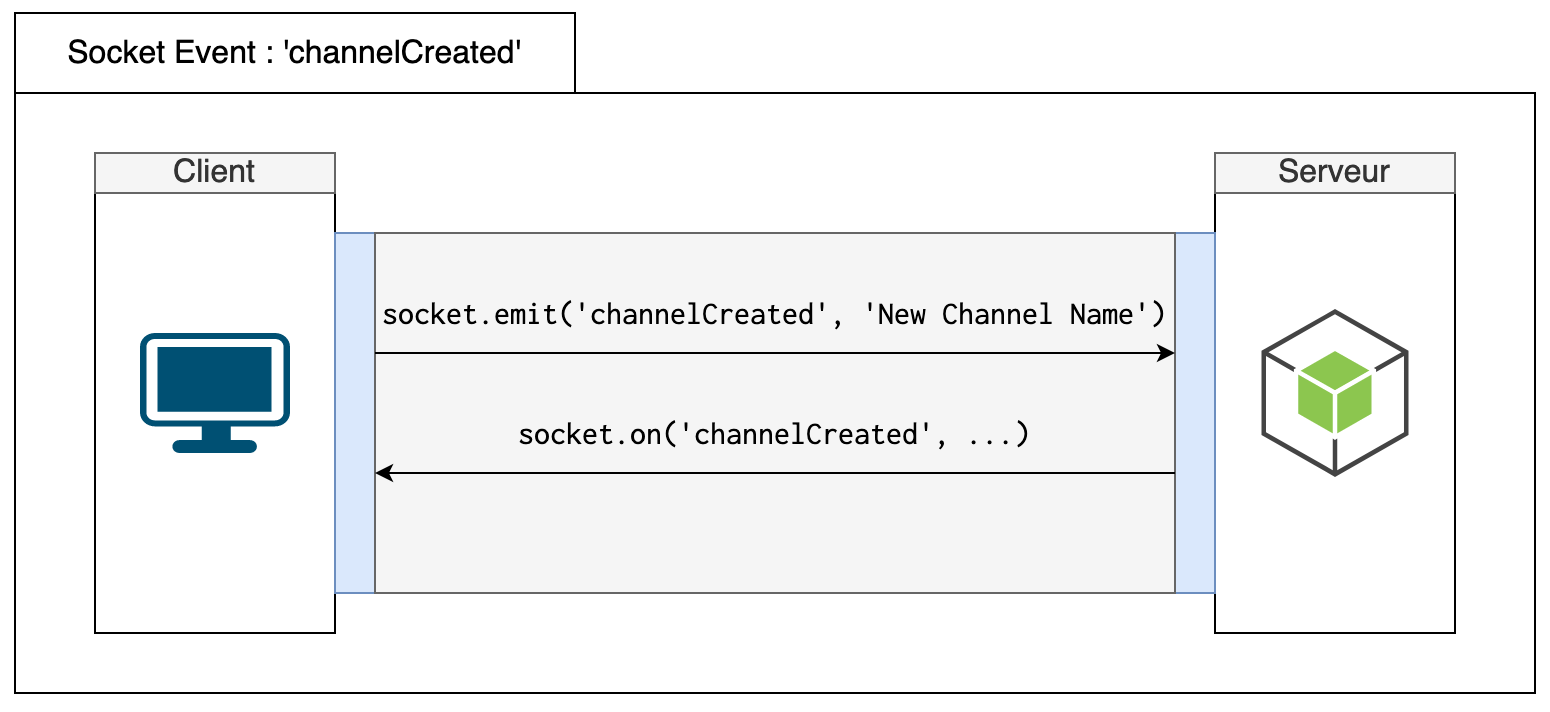
Pour les opérations sur les canaux de discussion, nous avons des requêtes POST permettant d’ajouter l’information des canaux à la base de données ainsi que des évènements pour avoir une certaine fluidité dans notre application au niveau de la connexion.



Événement: channelCreated

Méthode de requête: WebSocket

Data: name: nom du canal



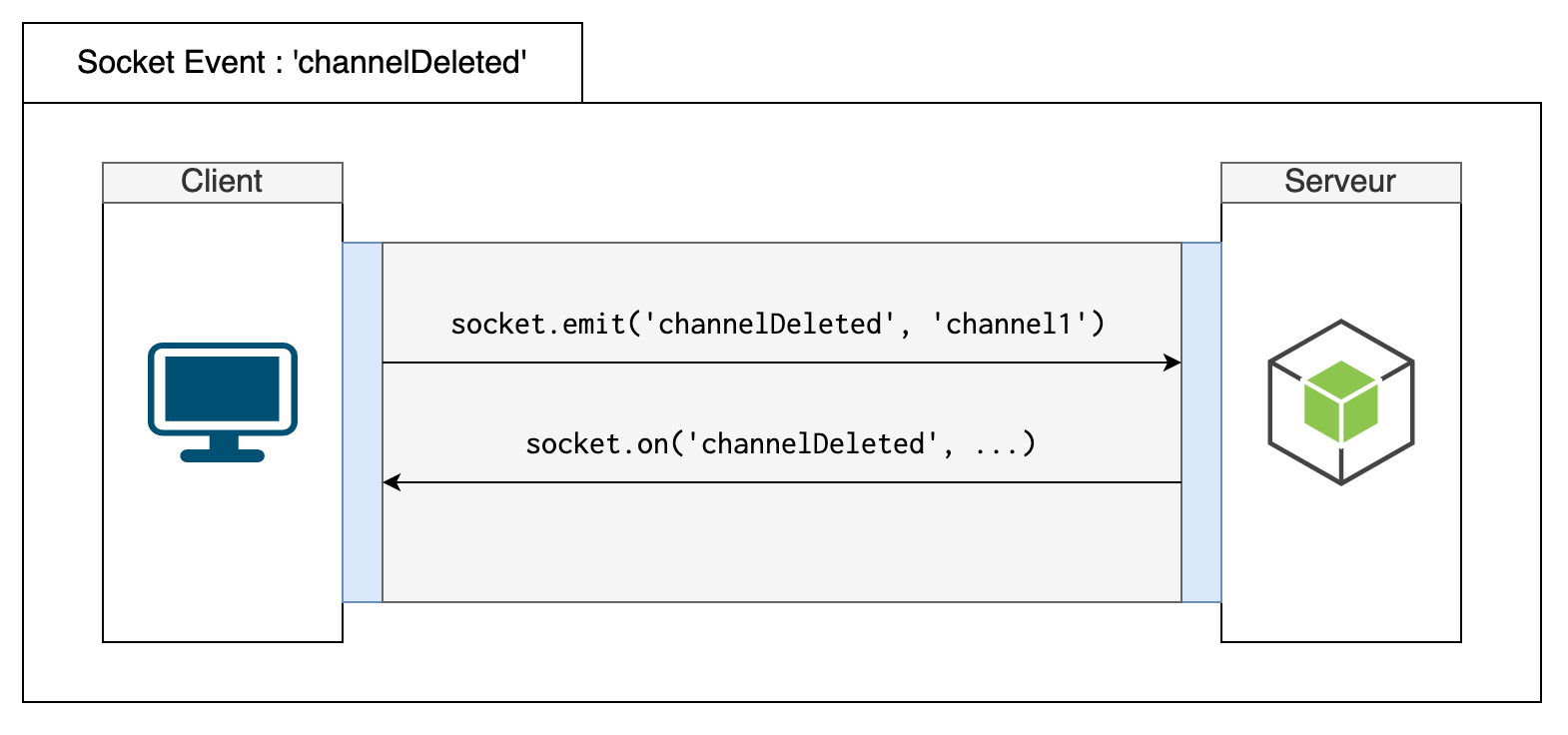
Par la suite, le client rejoint automatiquement le canal (voir section 3.7).

## 3.5.4 Suppression d’un canal de discussion

Événement: channelDeleted

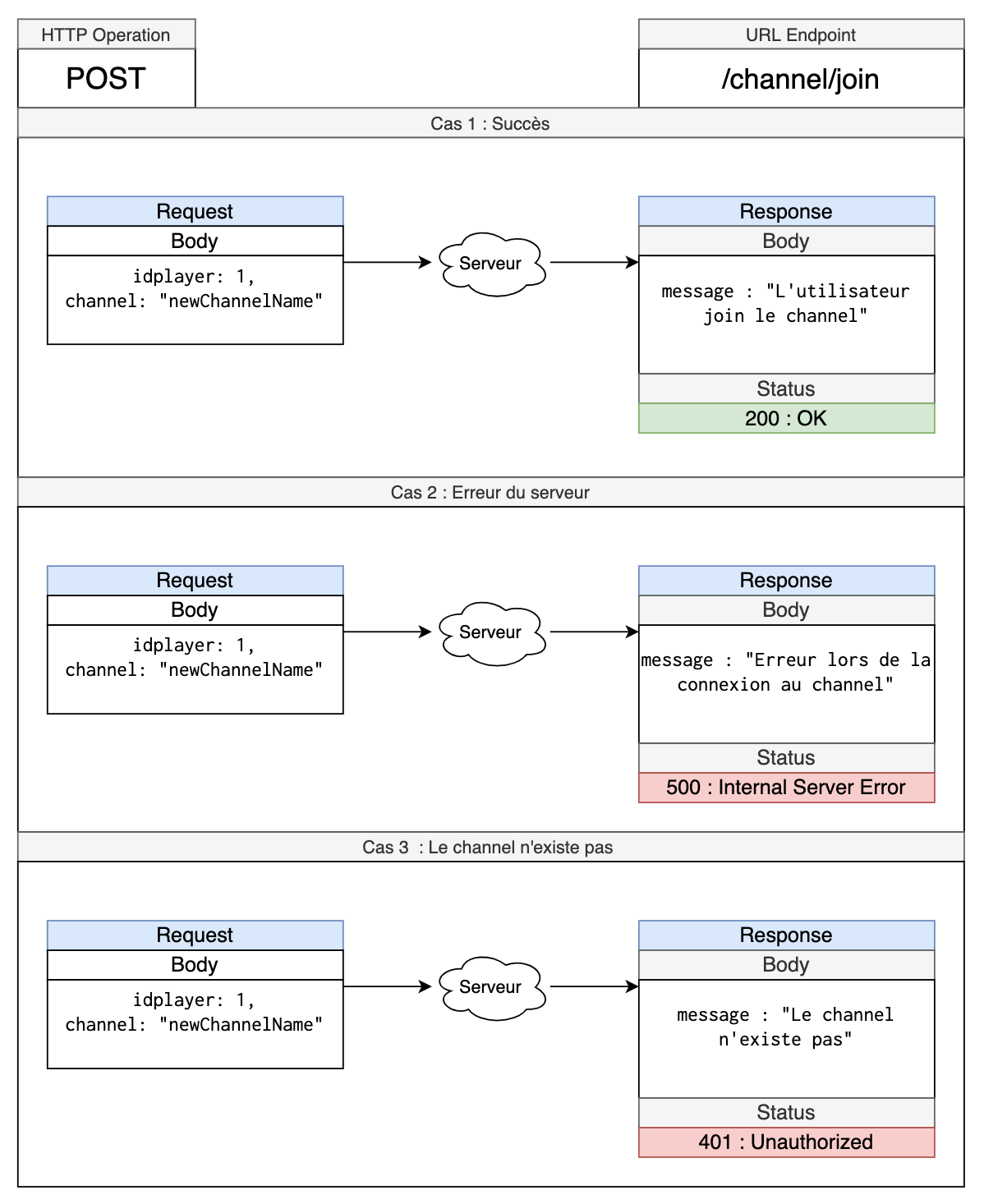
Méthode de requête: WebSocket

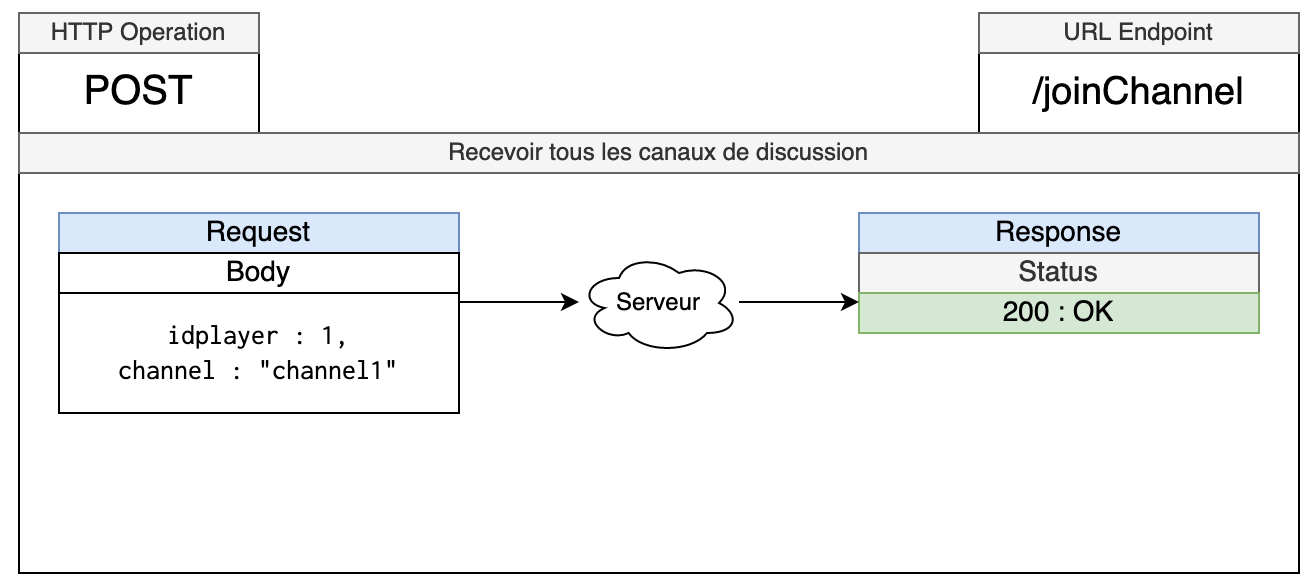
Data: name: nom du canal



Le client arrête aussi d’écouter l'événement qui reçoit les messages du canal (voir section 3.9).

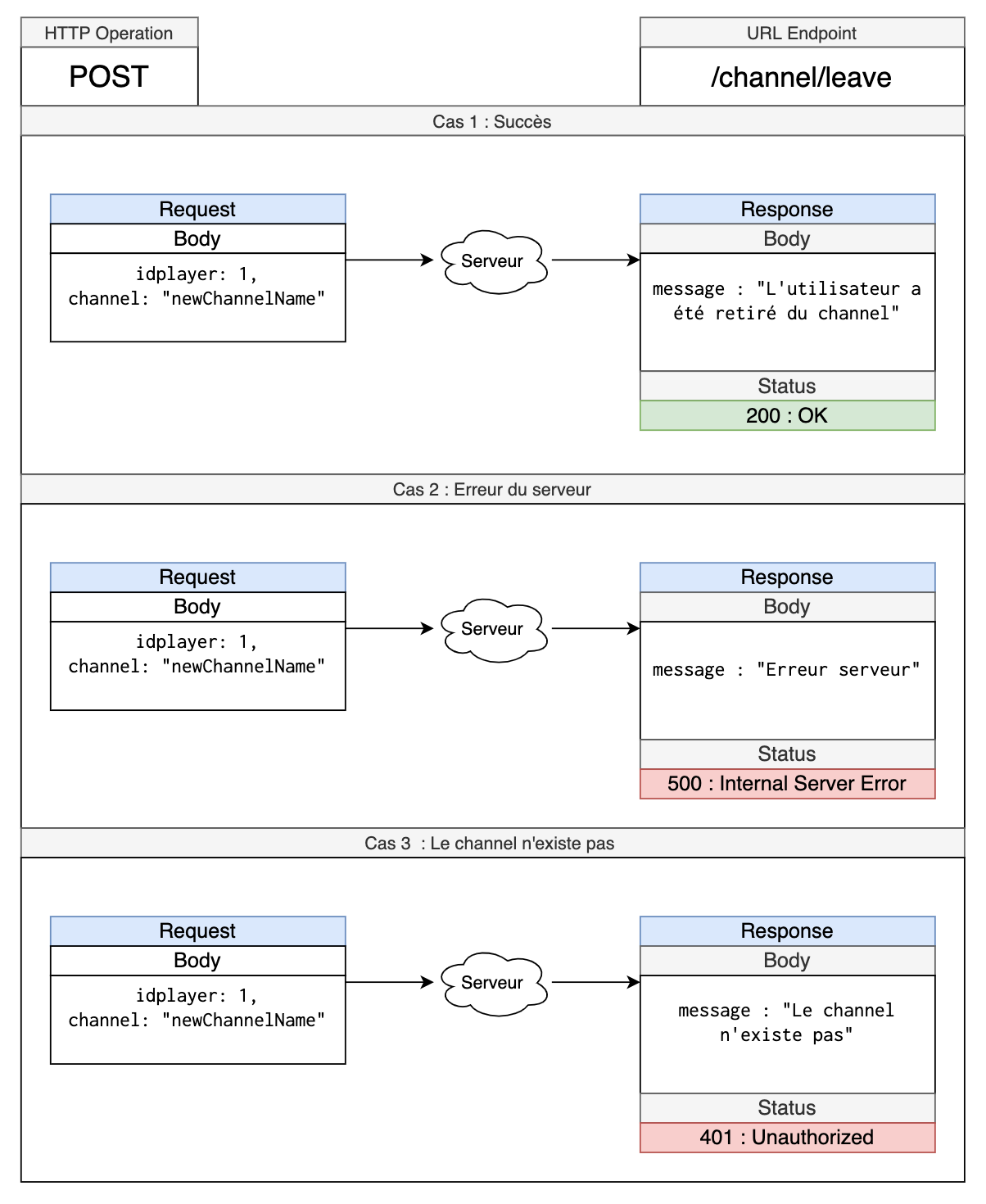
## 3.5.5 Joindre un canal de discussion

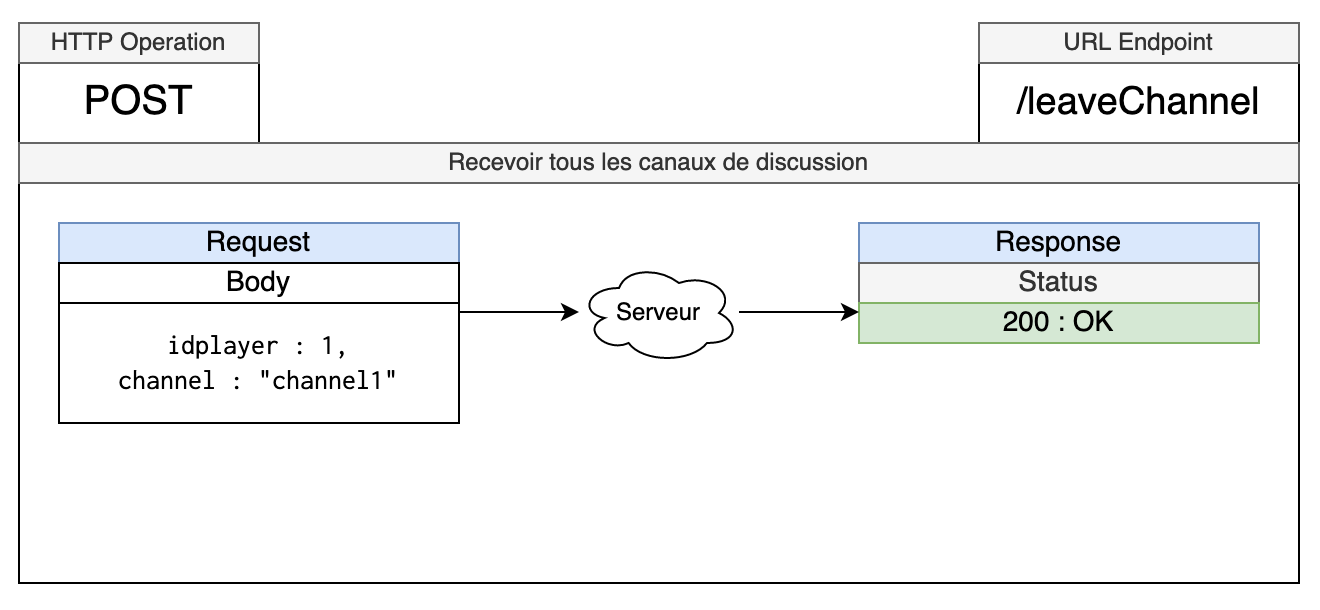




Par la suite, le client écoute l’événement pour recevoir les messages du canal (voir section 3.9).

## 3.5.6 Quitter un canal de discussion





Le client arrête aussi d’écouter l'événement qui reçoit les messages du canal (voir section 3.9).

## 3.5.7 Envoi/Réception de messages

Chaque client passe par un même événement de WebSocket pour envoyer/recevoir des messages d’un canal de discussion. Le serveur reçoit le socket et le broadcast à tous les clients observant cet événement. Les clients observant l’événement font tous partie de la Room de Socket associé au canal de discussion. Le serveur s’occupe de censurer le message ainsi que d’ajouter un timestamp avant de l’envoyer à tous les clients faisant partie de la room.

Événement: “ chatMessageXX ” où XX est l’identifiant du canal de discussion

Data: {

message: contenu du message

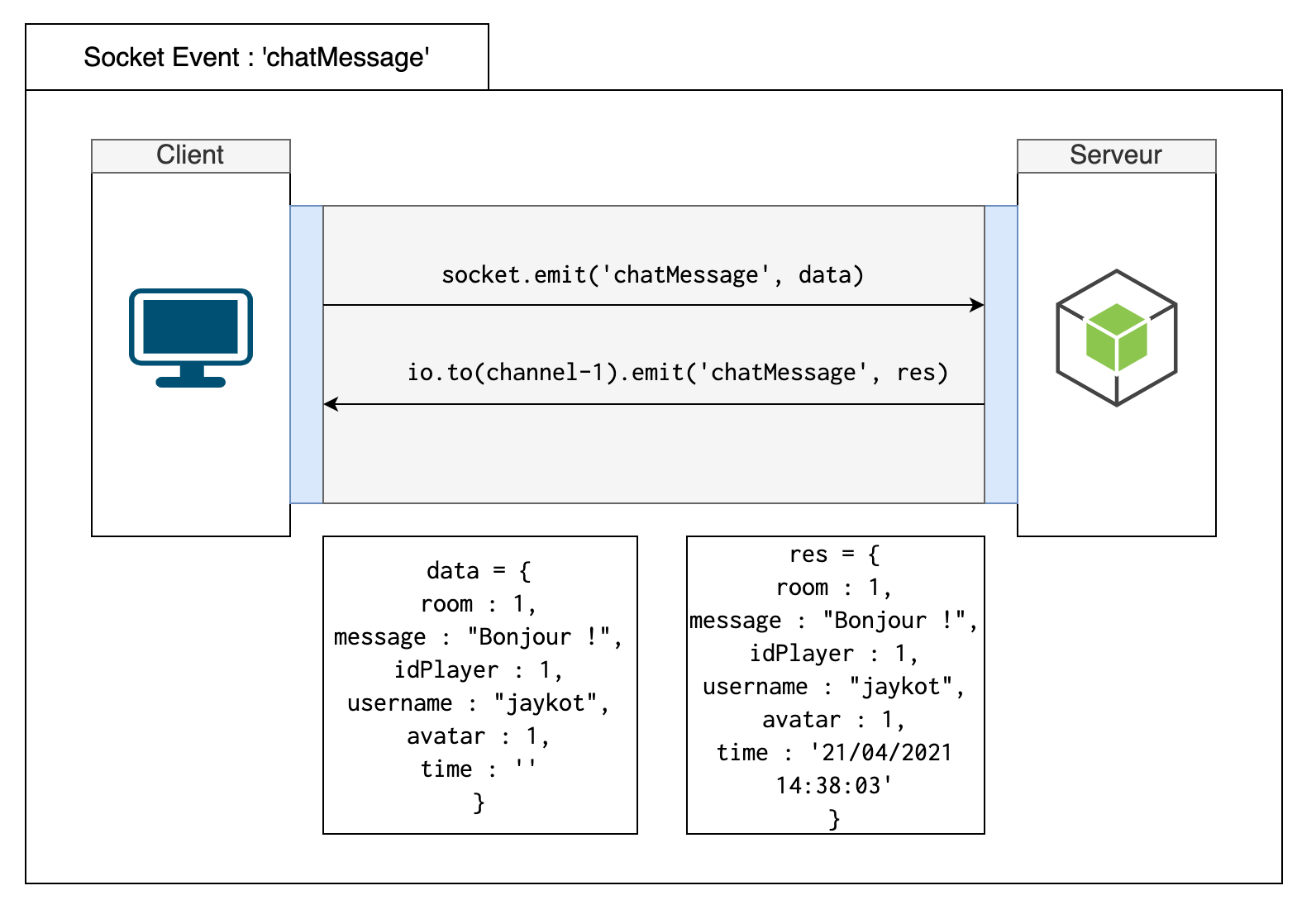
username: nom d’utilisateur de l’expéditeur

avatar: nom du fichier d’image de l’avatar de l’expéditeur

time: temps d’envoi sous format heure:minute:seconde

room : id de la room du canal de discussion

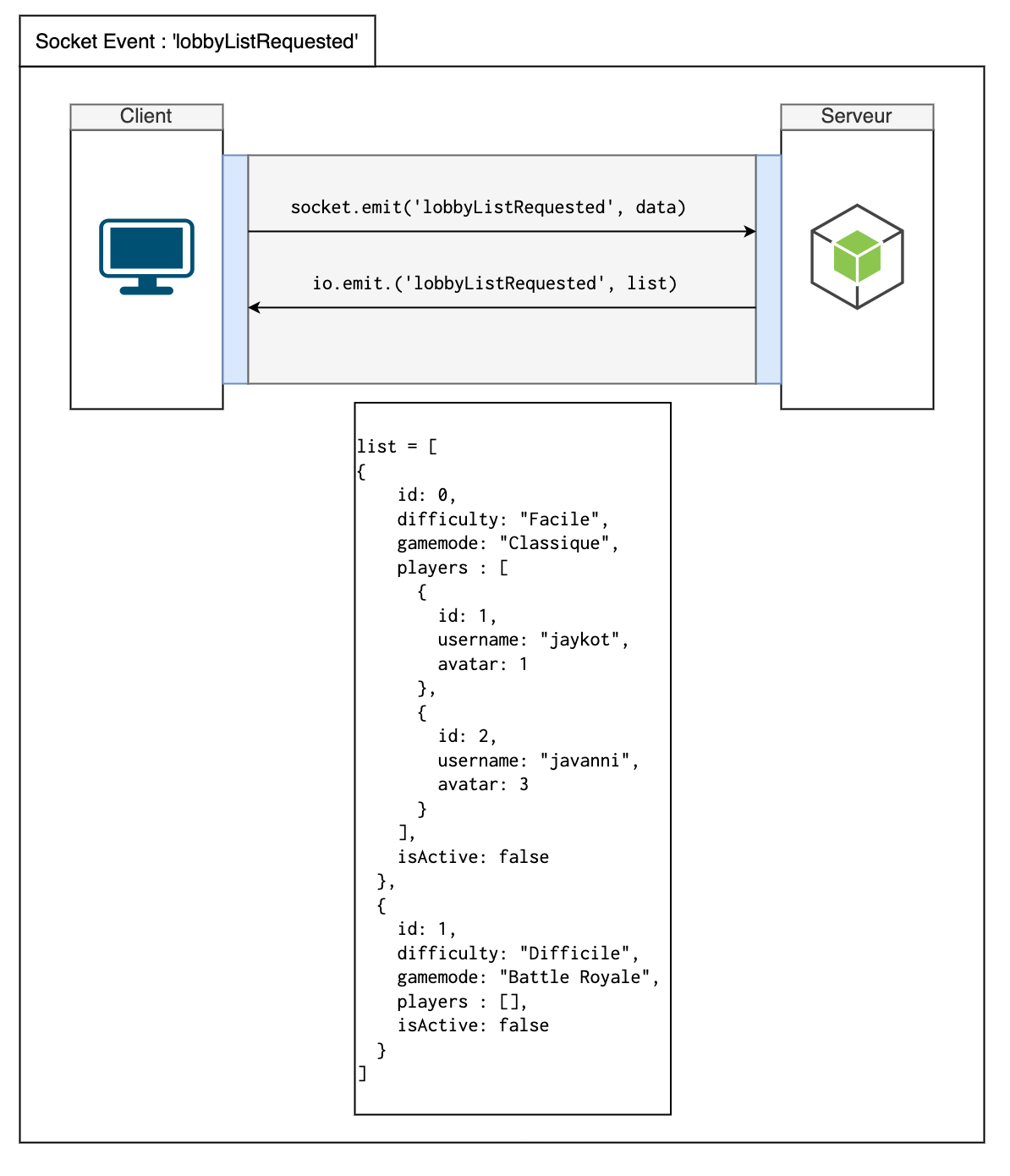
}



## 

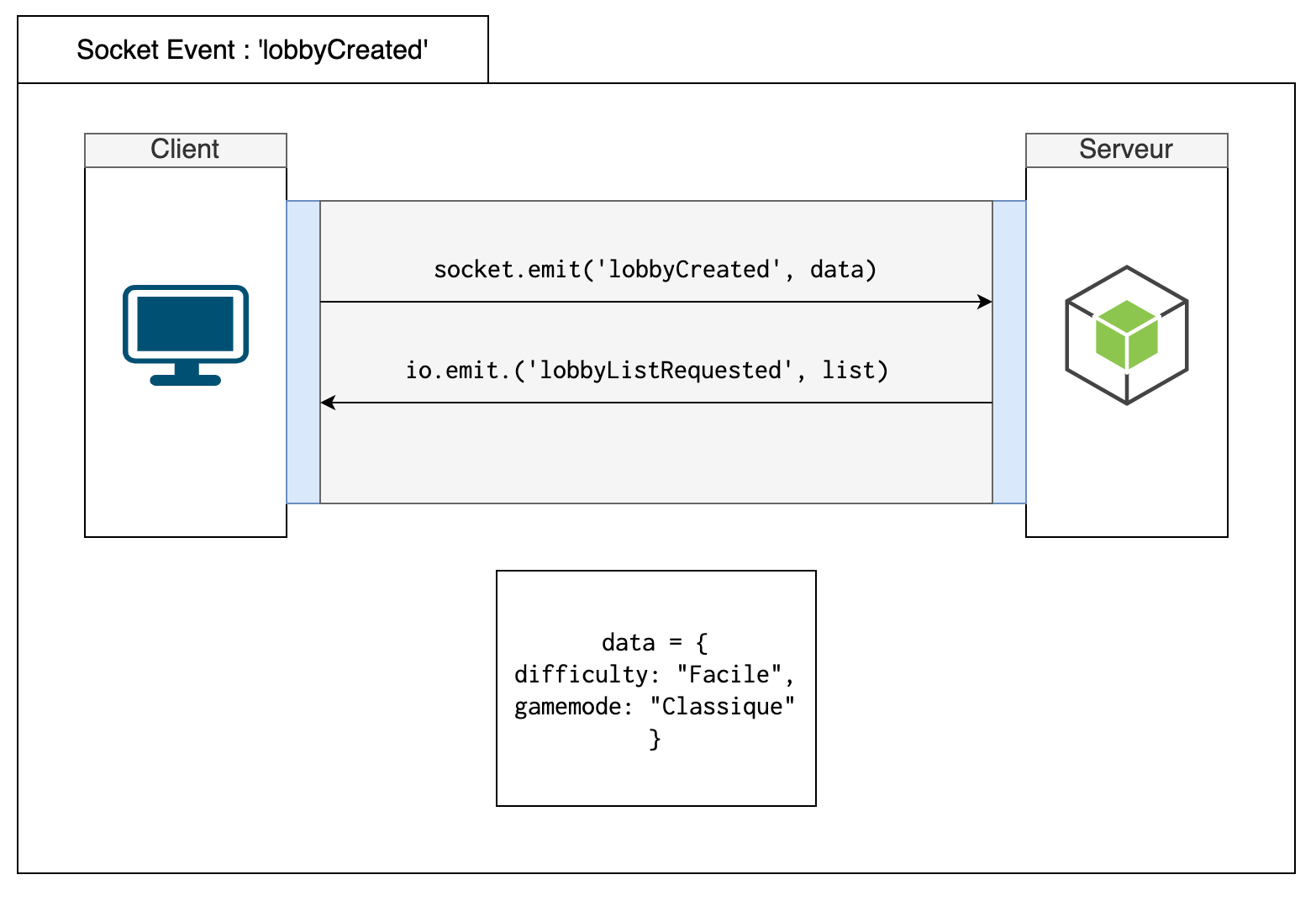
## 3.6 Lobbies

## 3.6.1 Réception des lobbies disponibles

L’application comporte un événement pour mettre à jour la liste des lobbies sur les clients. 

Cet événement est envoyé à chaque modification que l’on apporte à la liste de lobbies. La liste envoyé est la liste de tous les lobbies qui est stocké par le serveur.

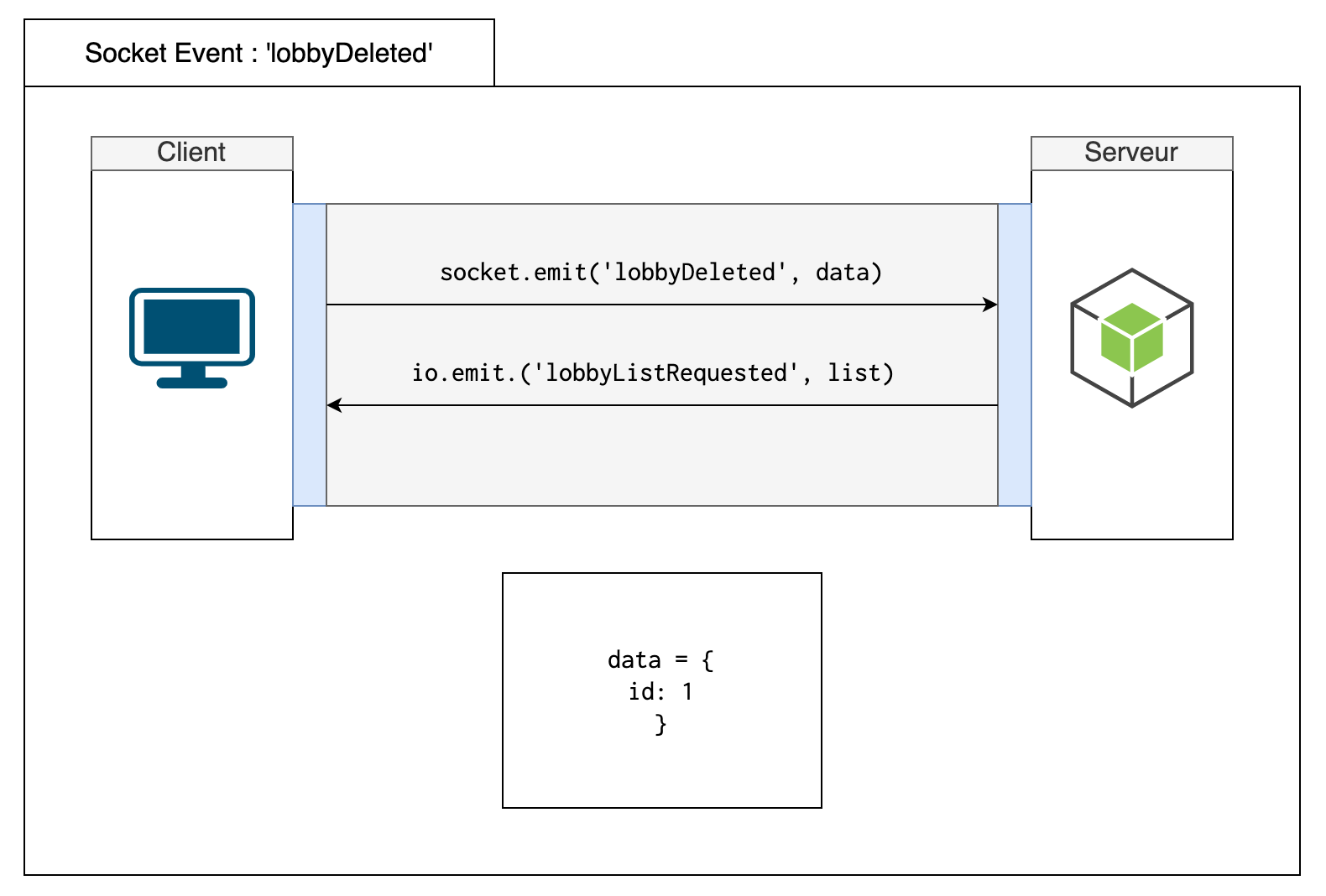
## 3.6.2 Création d’un lobby



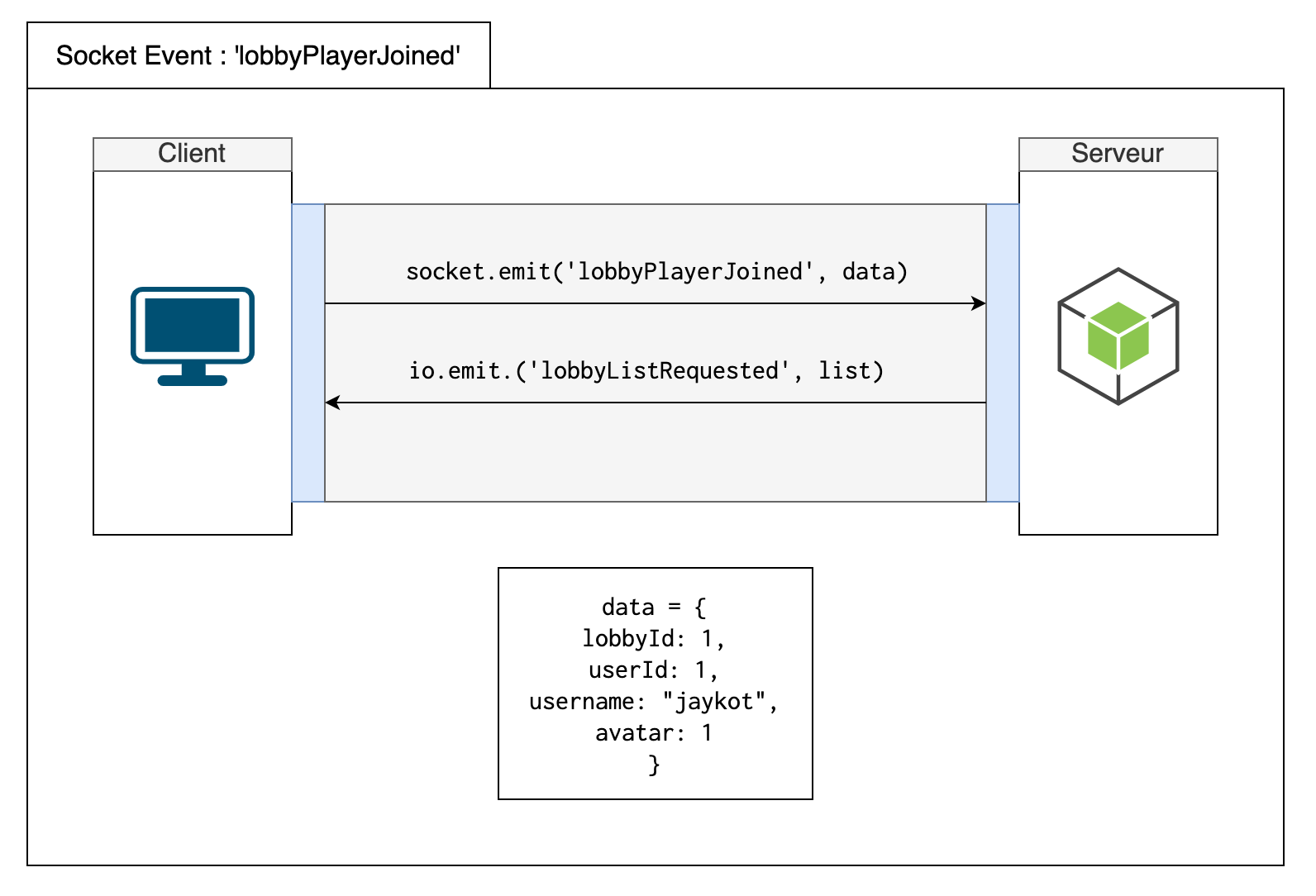
Suite à la création d’un lobby, on envoie la nouvelle liste des lobbies mise à jour contenant le nouveau lobby créé à tous les clients avec l’event LobbyListRequested.

## 

## 3.6.3 Suppression d’un lobby

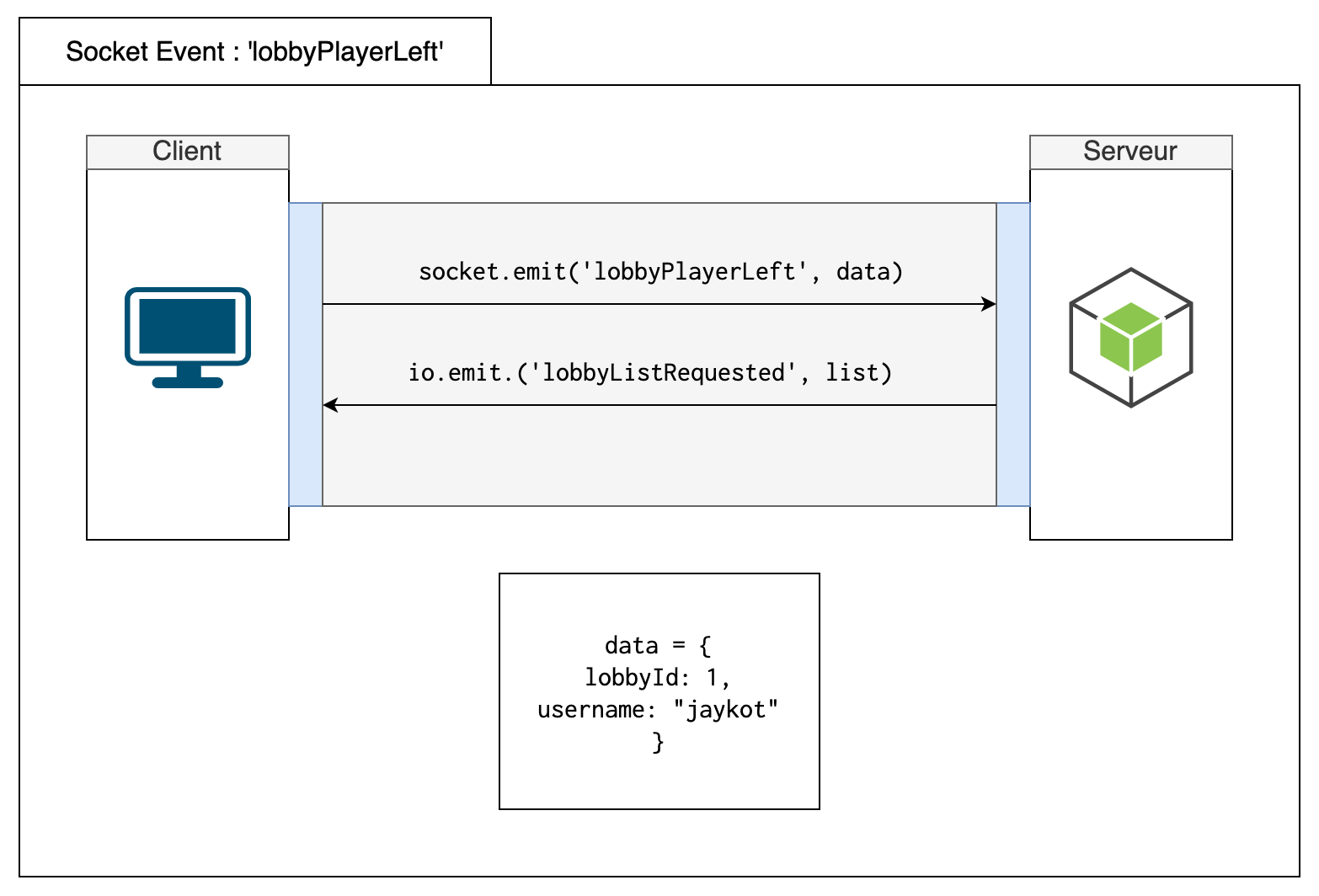


## 3.6.4 Joindre un lobby



## 

## 3.6.5 Quitter un lobby



## 3.7 États dans une partie

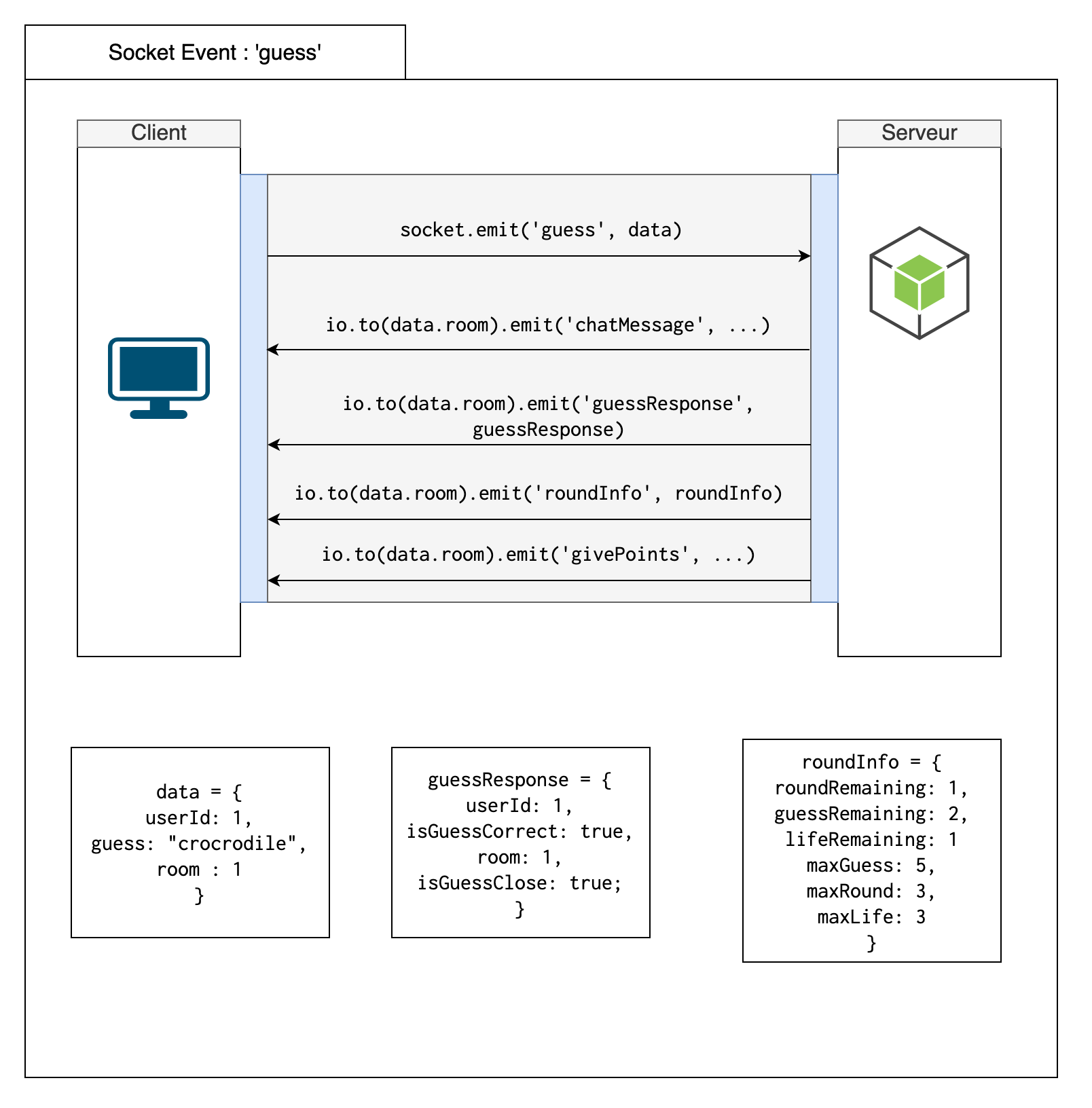
## 3.7.1 Début d’une partie

Cet événement envoie le signal pour débuter la partie en envoyant les informations pertinentes pour la construction du jeu. On change également le statut du lobby dans la liste des lobbies du serveur. C’est en recevant cet événement que le serveur s’assure d’ajouter les joueurs virtuels dans la partie si il y a lieu. Ces informations sont par la suite transmises à tous les clients de la partie avec l’événement gameSetup.

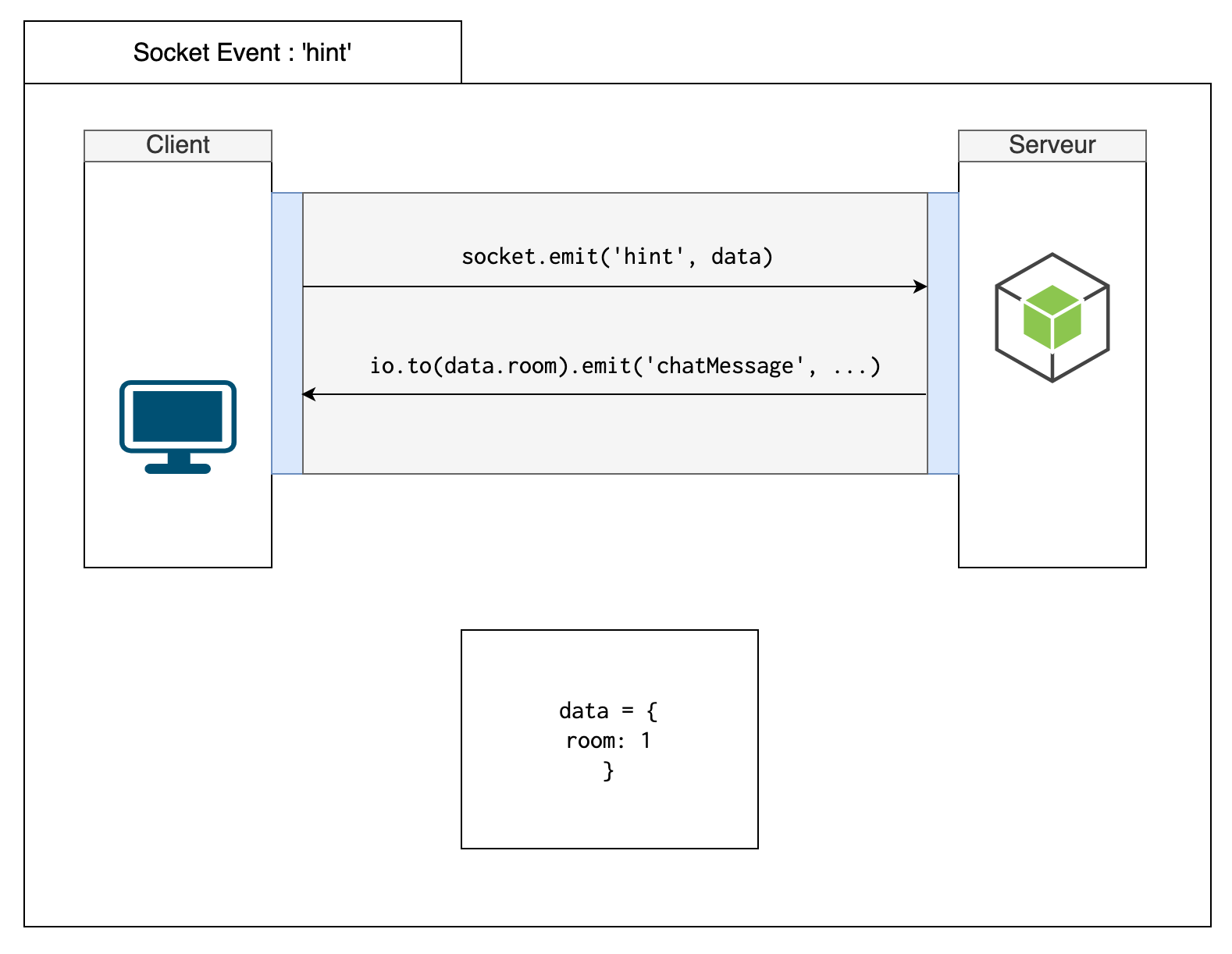
## 

## 

## 3.7.2 L’utilisateur devine le mot dessiné

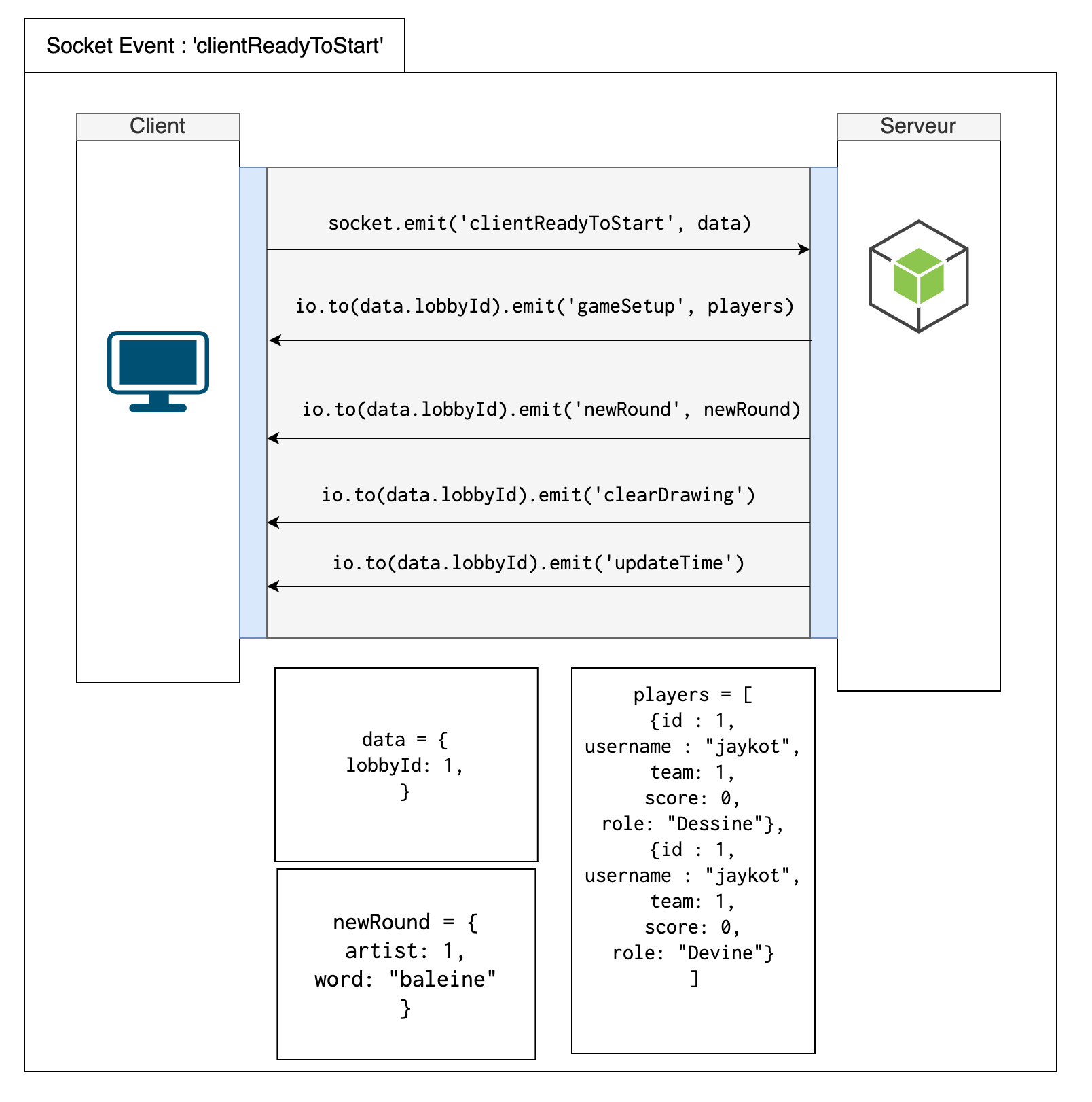


## 3.7.3 L’utilisateur demande un indice pour le mot dessiné

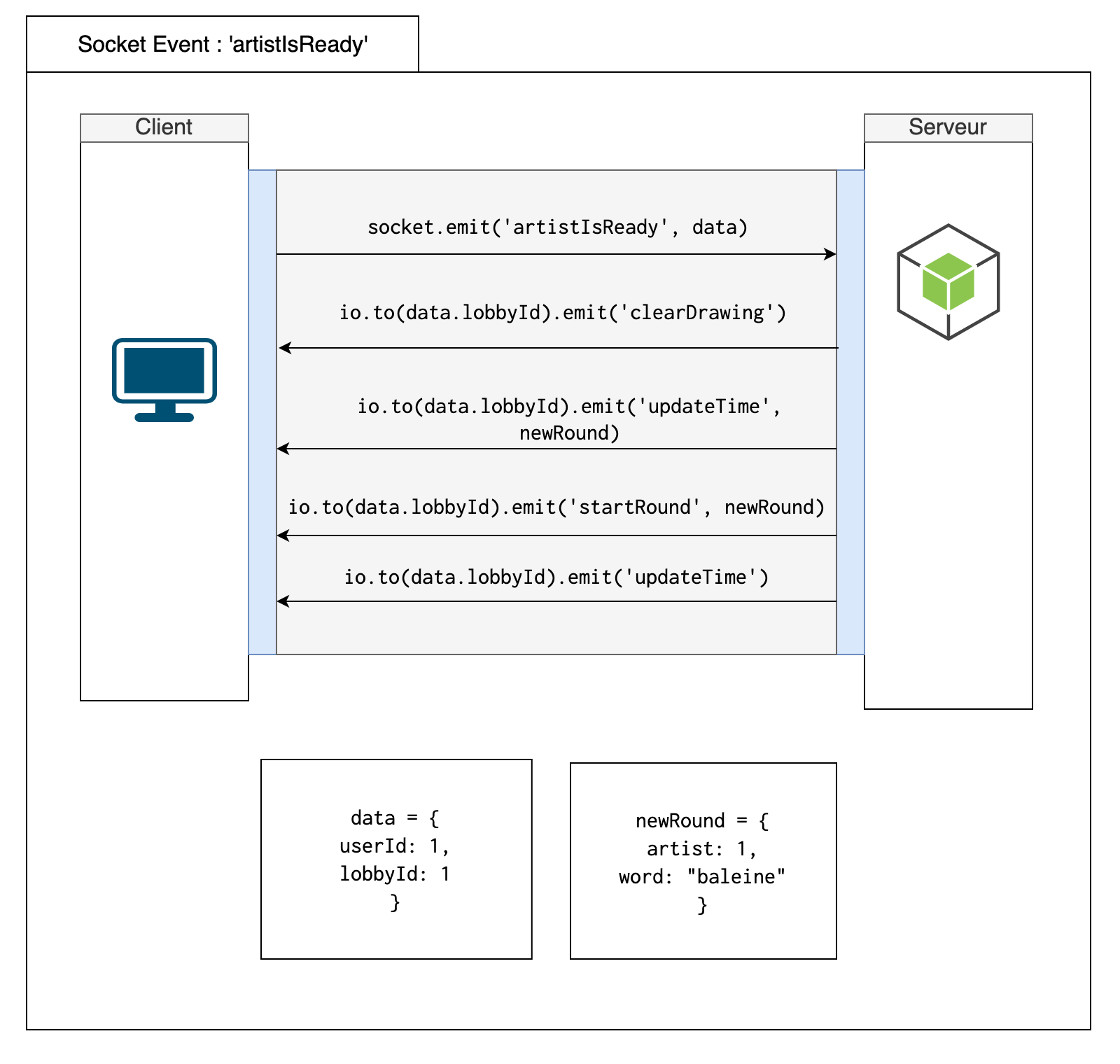


## 3.7.4 L’utilisateur envoie le signal qu’il est prêt à débuter la nouvelle round dans la partie

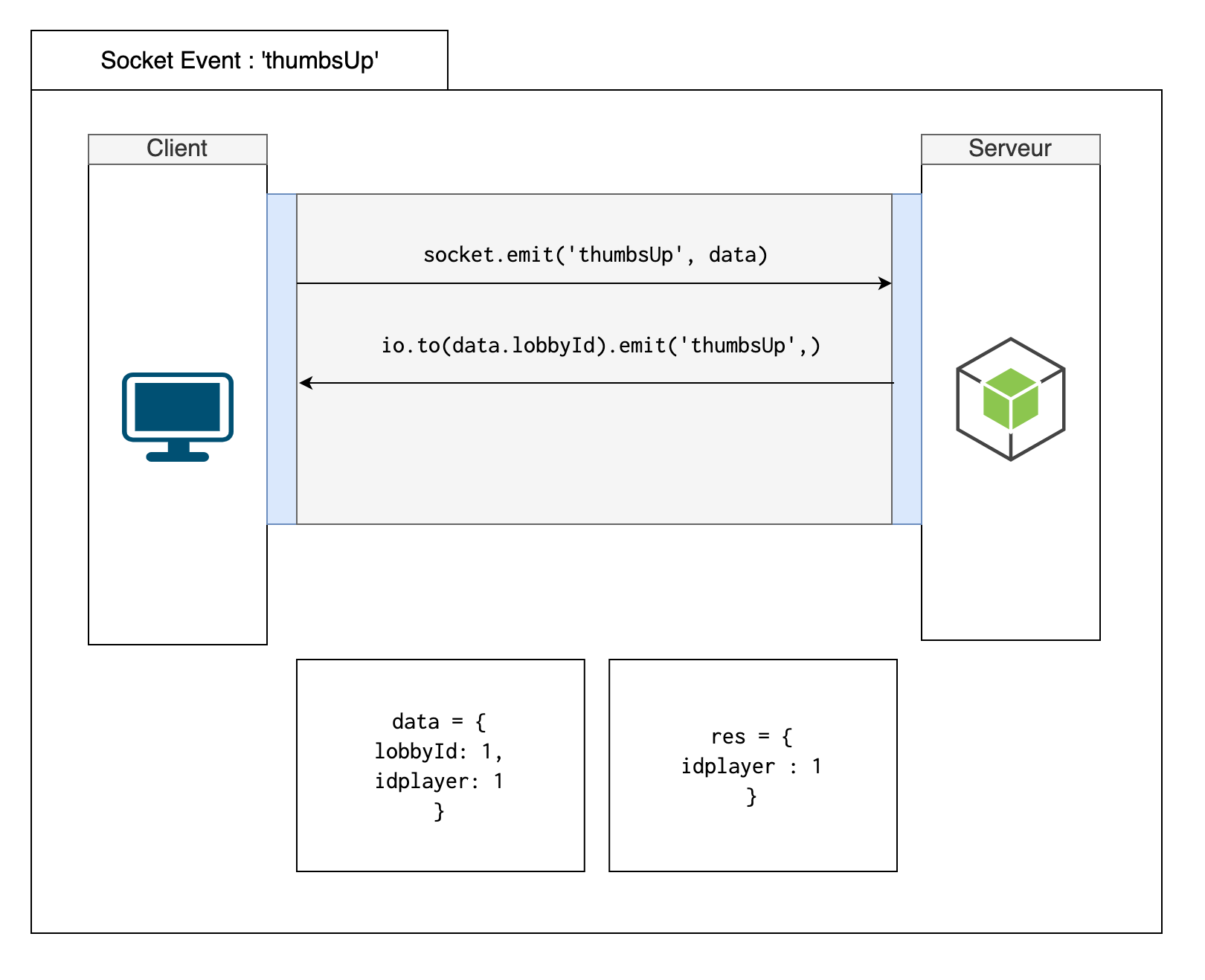
Le serveur attend que tous les clients soient prêts avant de débuter une nouvelle partie. Par la suite, il envoie le signal du début d’une nouvelle partie.



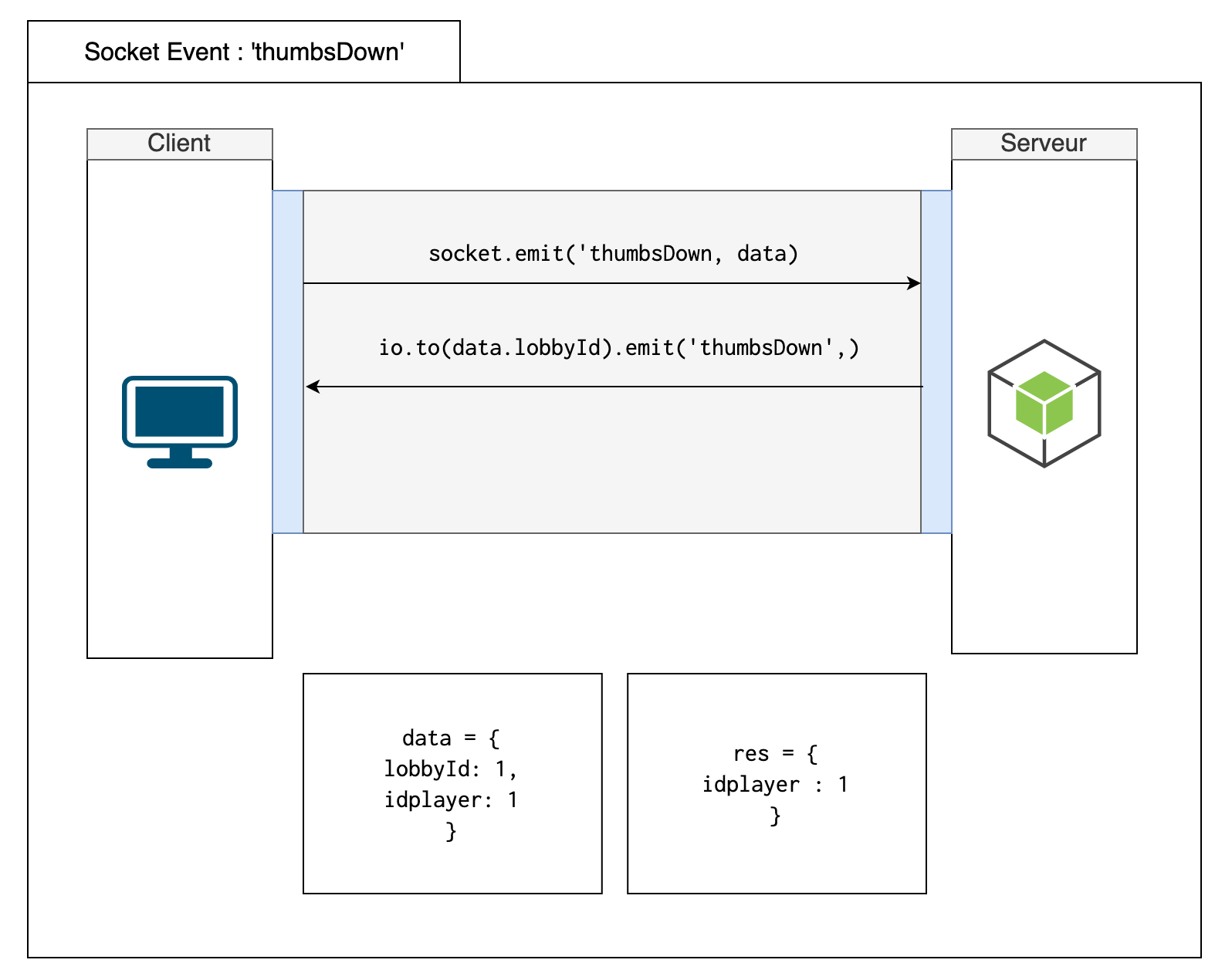
## 3.7.5 L’artiste envoie le signal qu’il est prêt



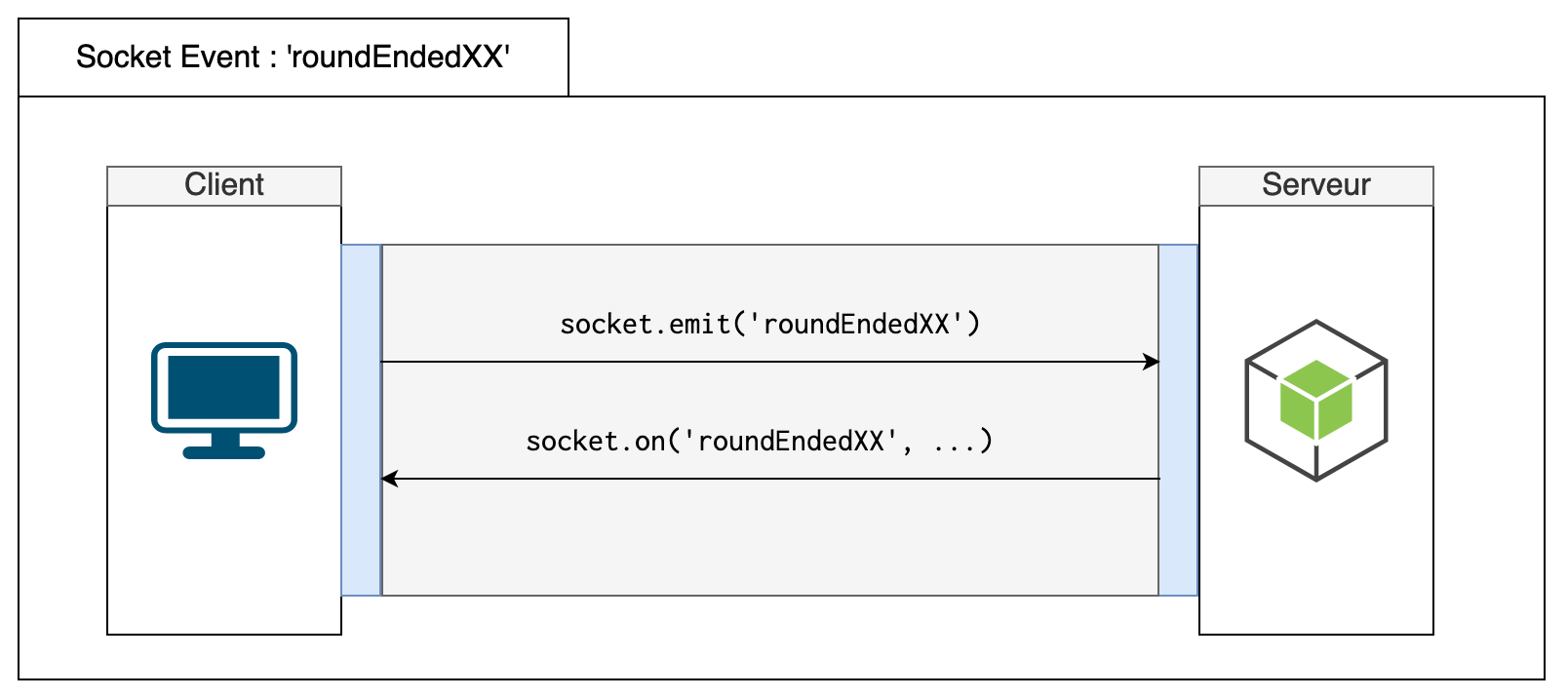
## 3.7.6 L’utilisateur envoie un J’aime au dessinateur



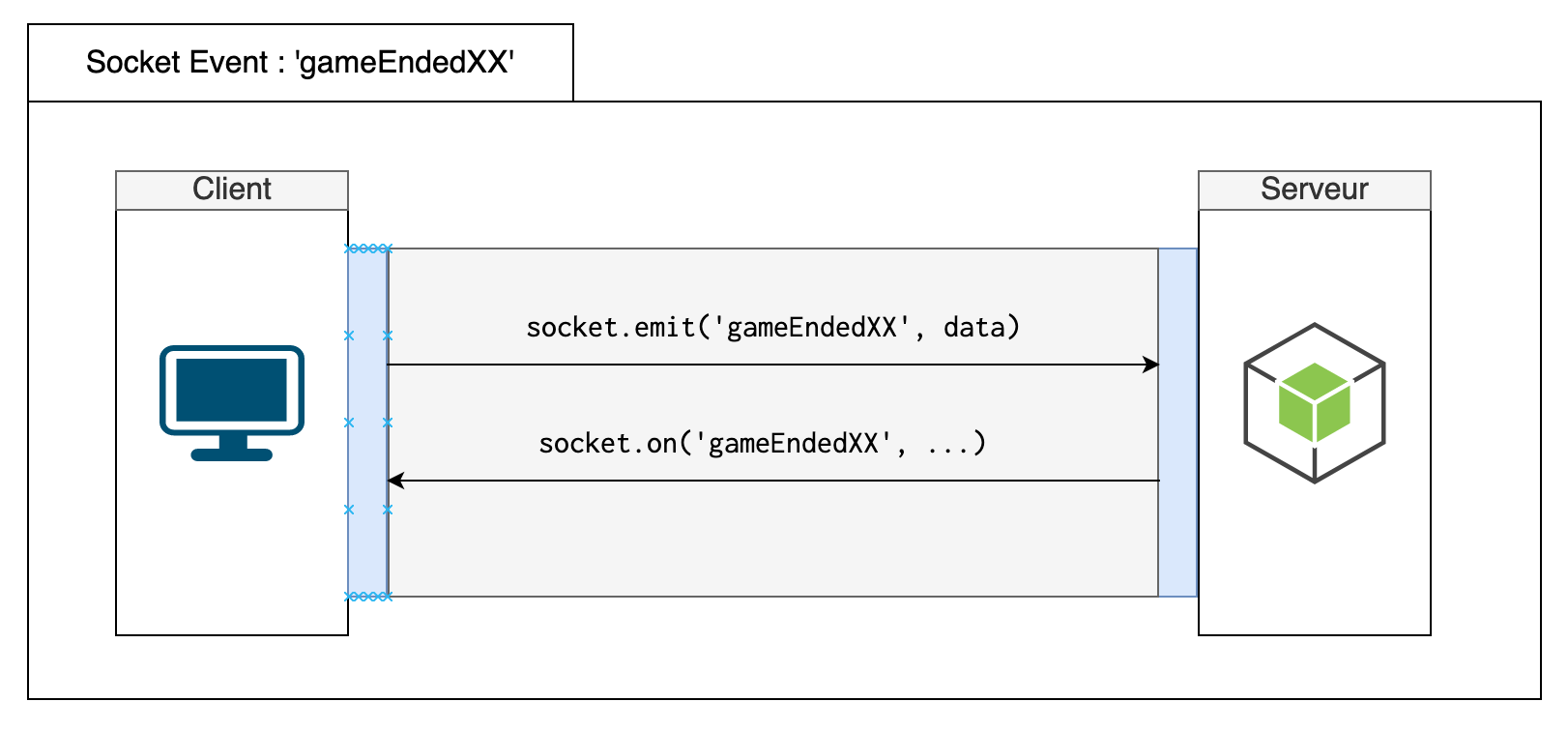
## 3.7.7 L’utilisateur envoie un Je n’aime pas au dessinateur



## 3.7.8 Fin d’une manche

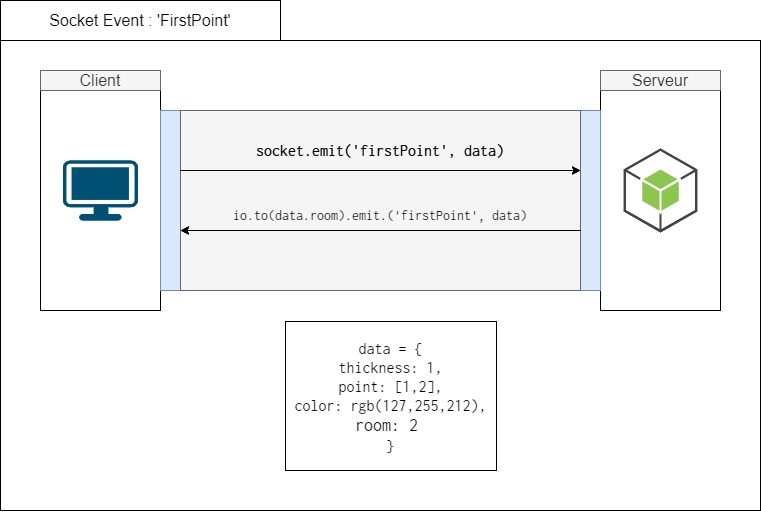


## 3.7.9 Fin d’une partie

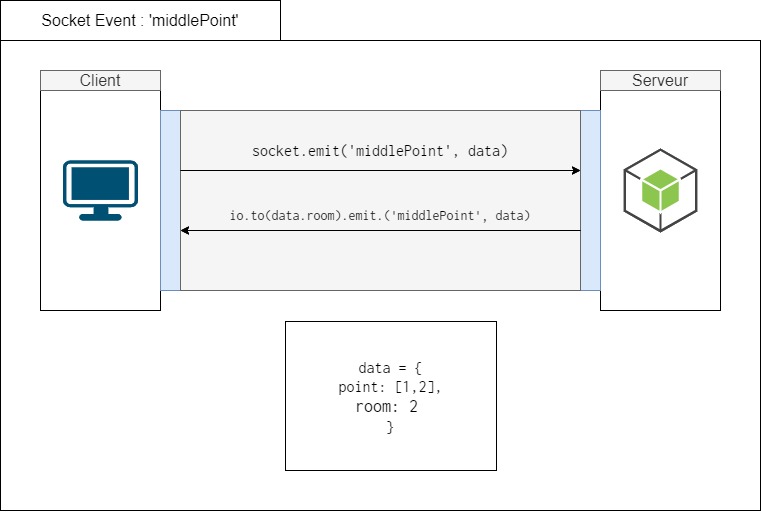


## 3.8 Envoie des dessins dans la partie

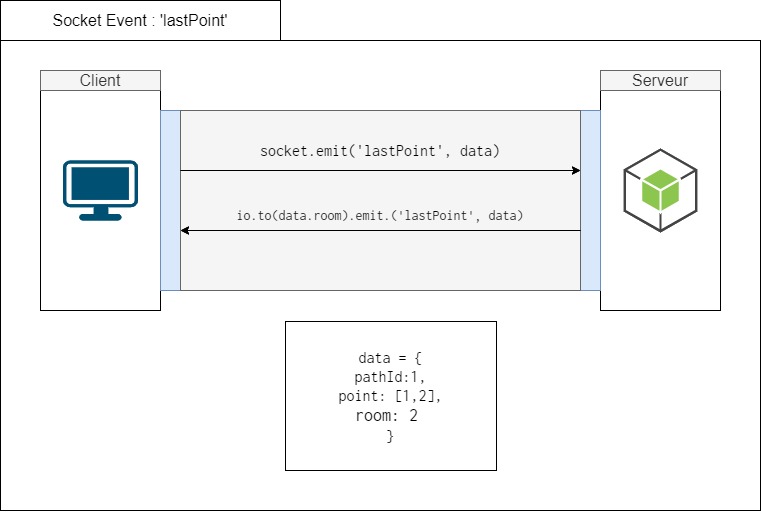
## 3.8.1 Premier point d’un trait



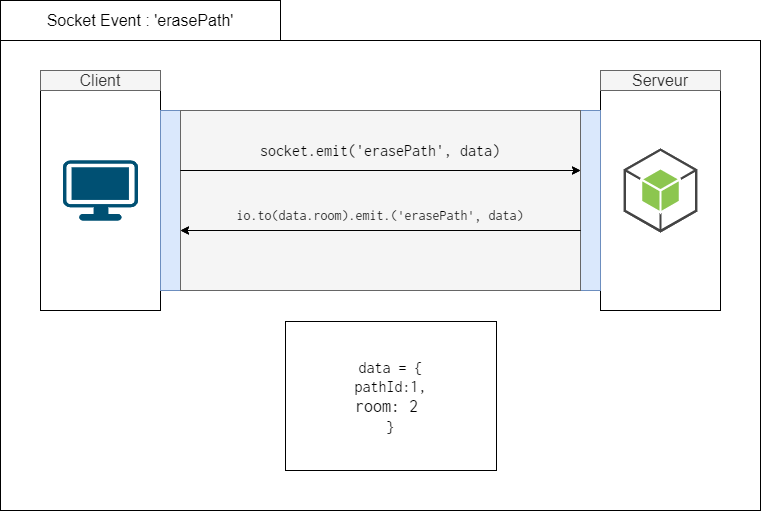
## 3.8.2 Point au milieu d’un trait



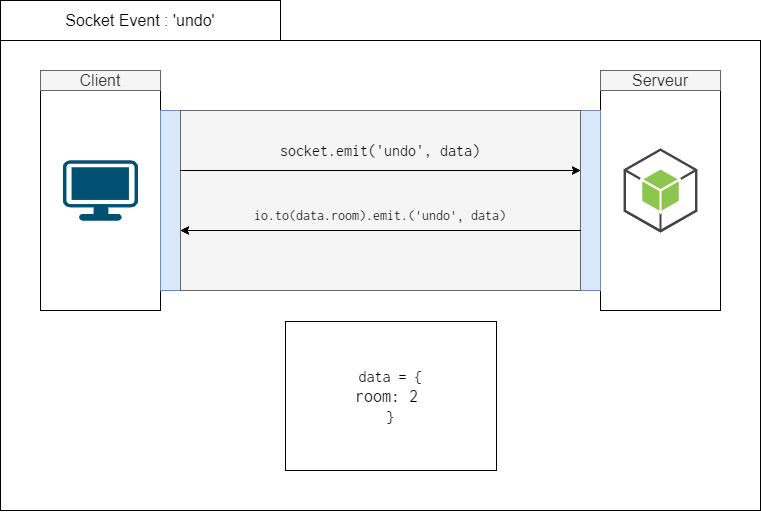
## 3.8.3 Dernier point d’un trait



## 3.8.4 Effacer un trait dans le dessin



## 3.8.5 Annuler le dernier trait



## 3.8.6 Refaire du dernier trait annulé

## 

## 

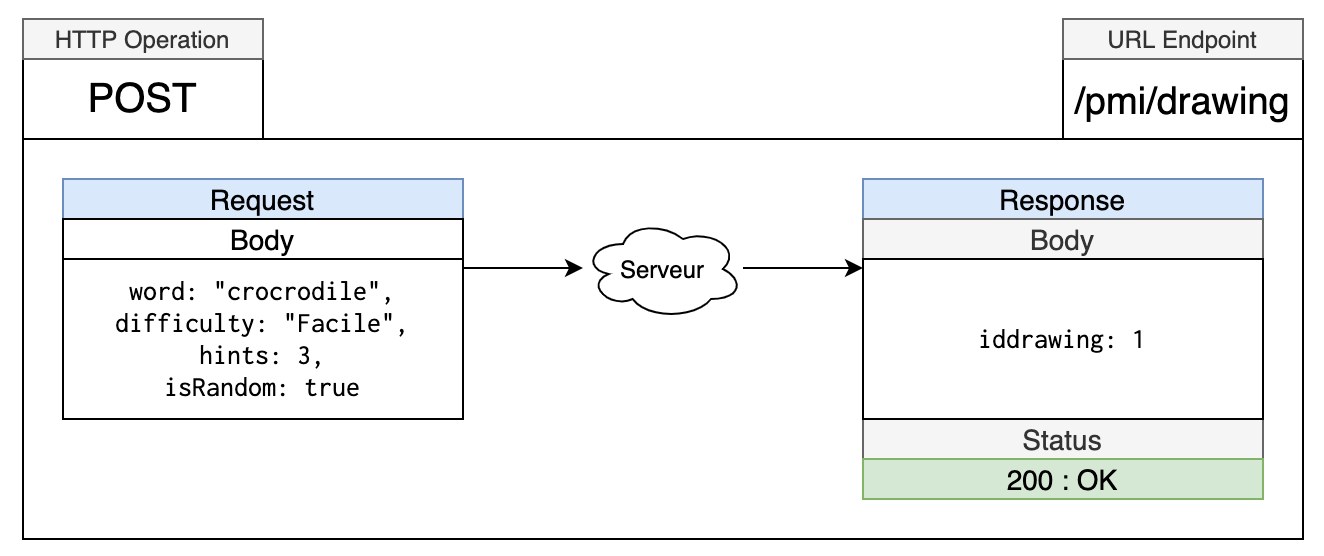
## 

## 

## 

## 3.9 Paire mot-image

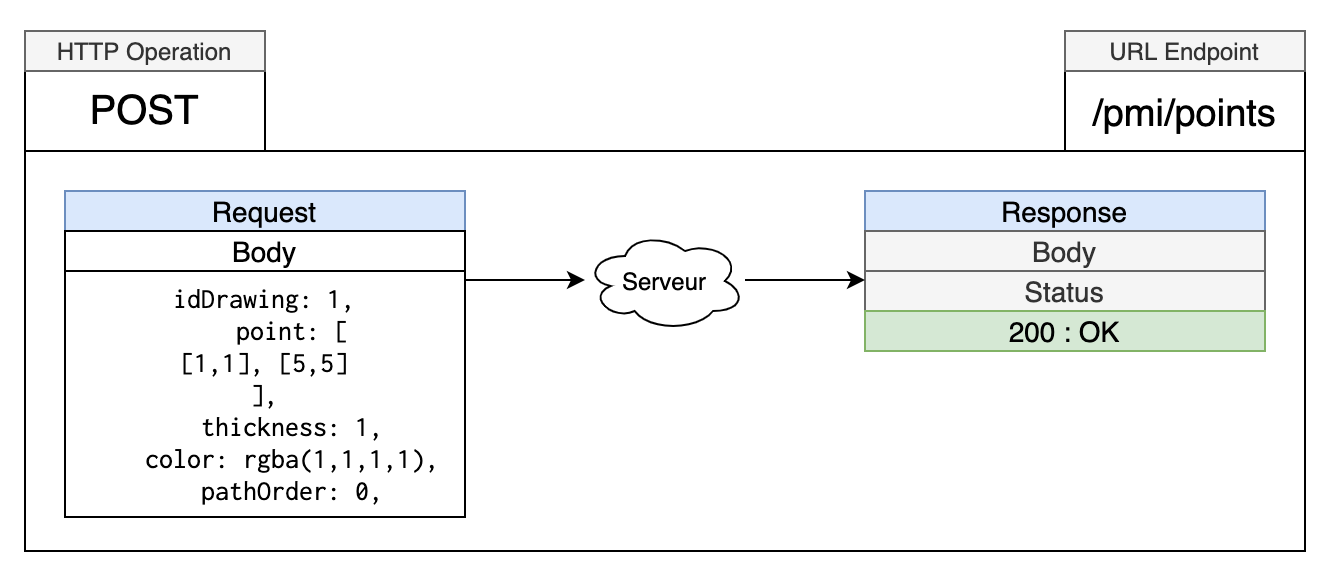
## 3.9.1 Envoie des infos de dessin paire-mot-image



Cette requête nous permet d’ajouter les informations nécessaires pour une paire mot-image. Nous ajouterons par la suite la path dans la requête suivante.

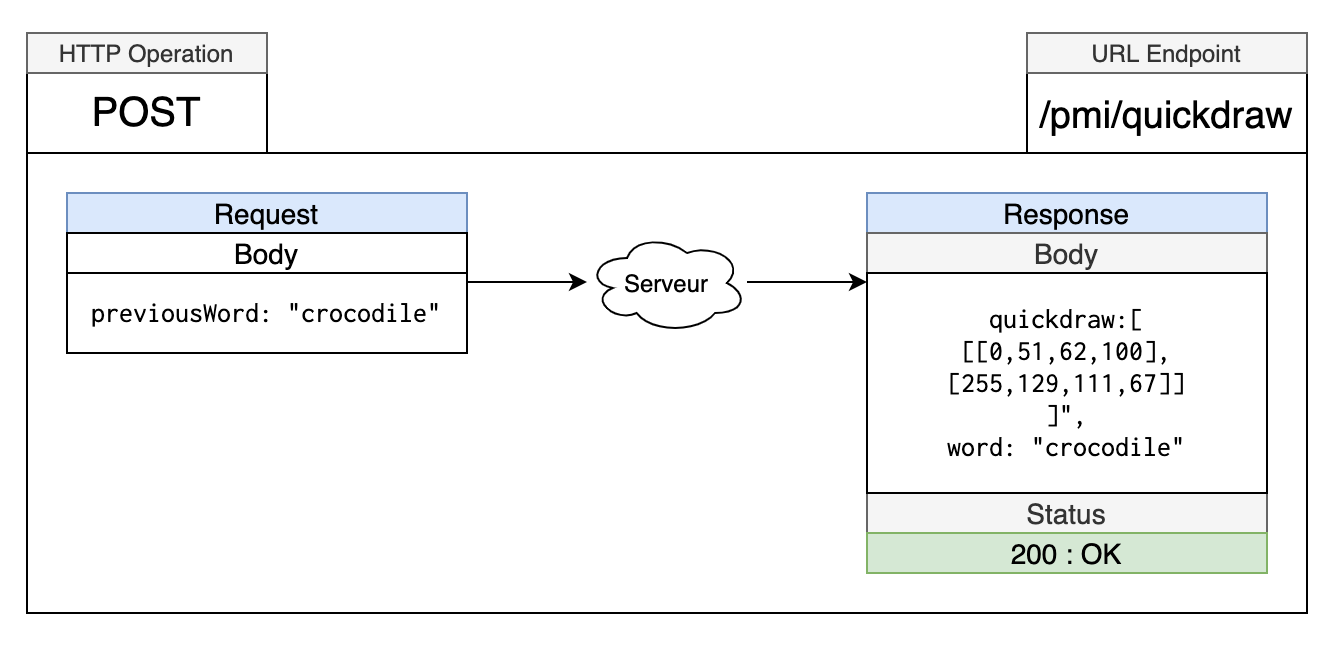
## 3.9.2 Envoie d’un path paire-mot-image

Avec cette requête, on ajoute le path de points au dessin dans le BD avec le idDrawing envoyé.



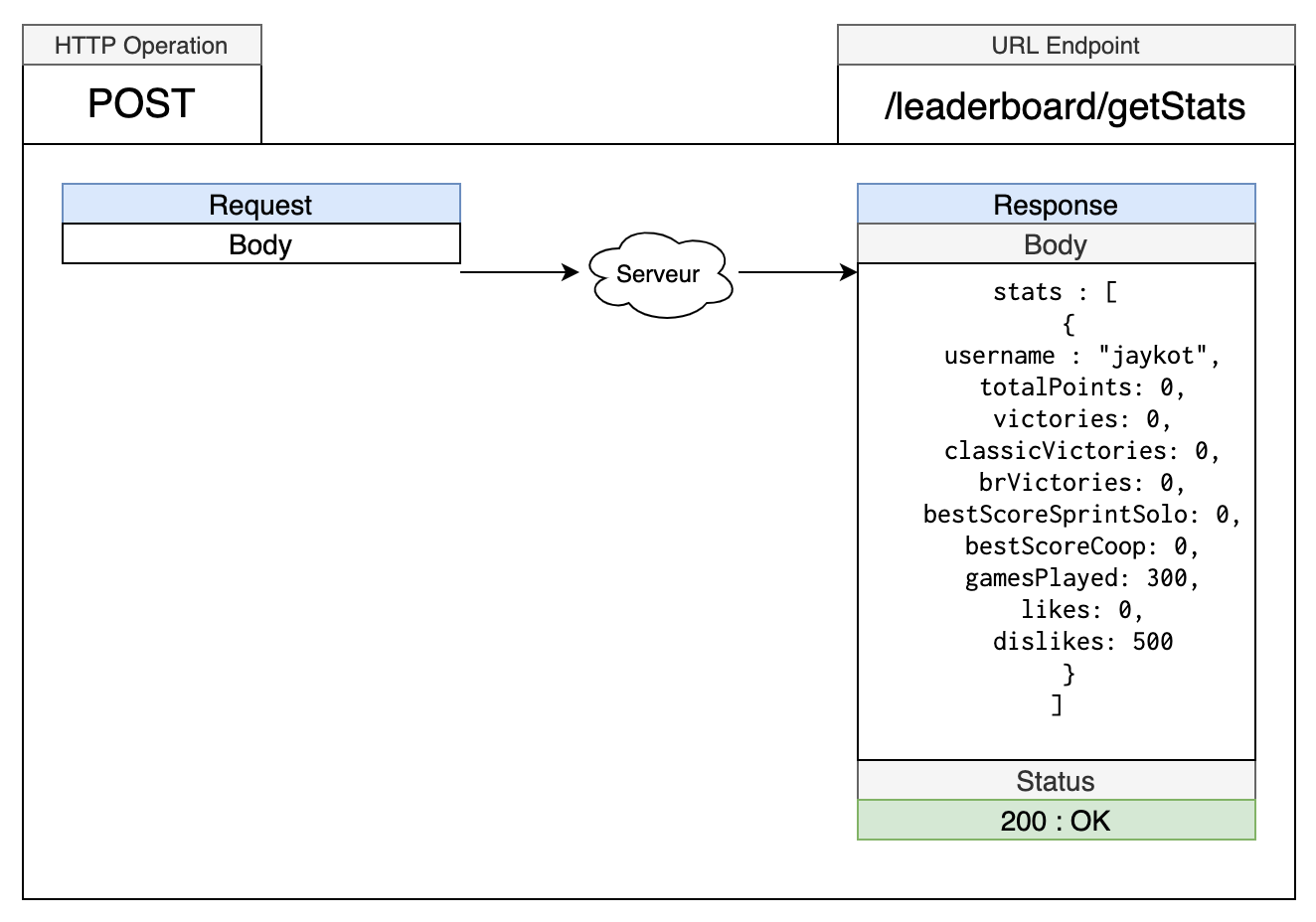
## 3.9.3 Mot quickdraw

Cette requête nous permet d’avoir accès à un mot ainsi que la path provenant de la librairie quickdraw.



## 3.10 Leaderboard

Cette requête nous permet d’obtenir l’information nécessaire pour construire le tableau de leaderboard sur le client lourd et le client léger. Le serveur retourne un array de statistiques pour chaque utilisateur inscrit à l’application.



## 

## 

## 

## 

## 

## 

## 

## 

## 

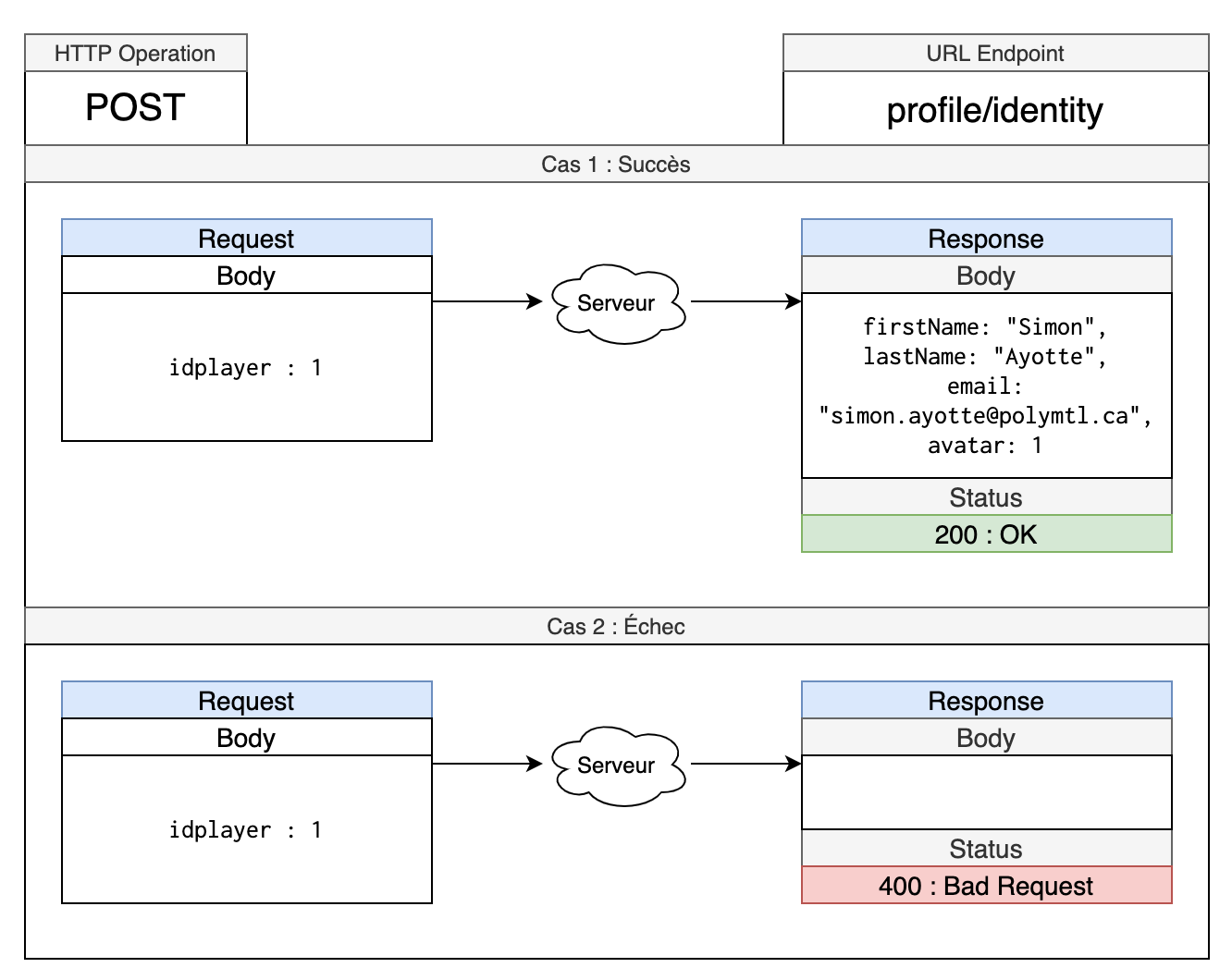
## 

## 

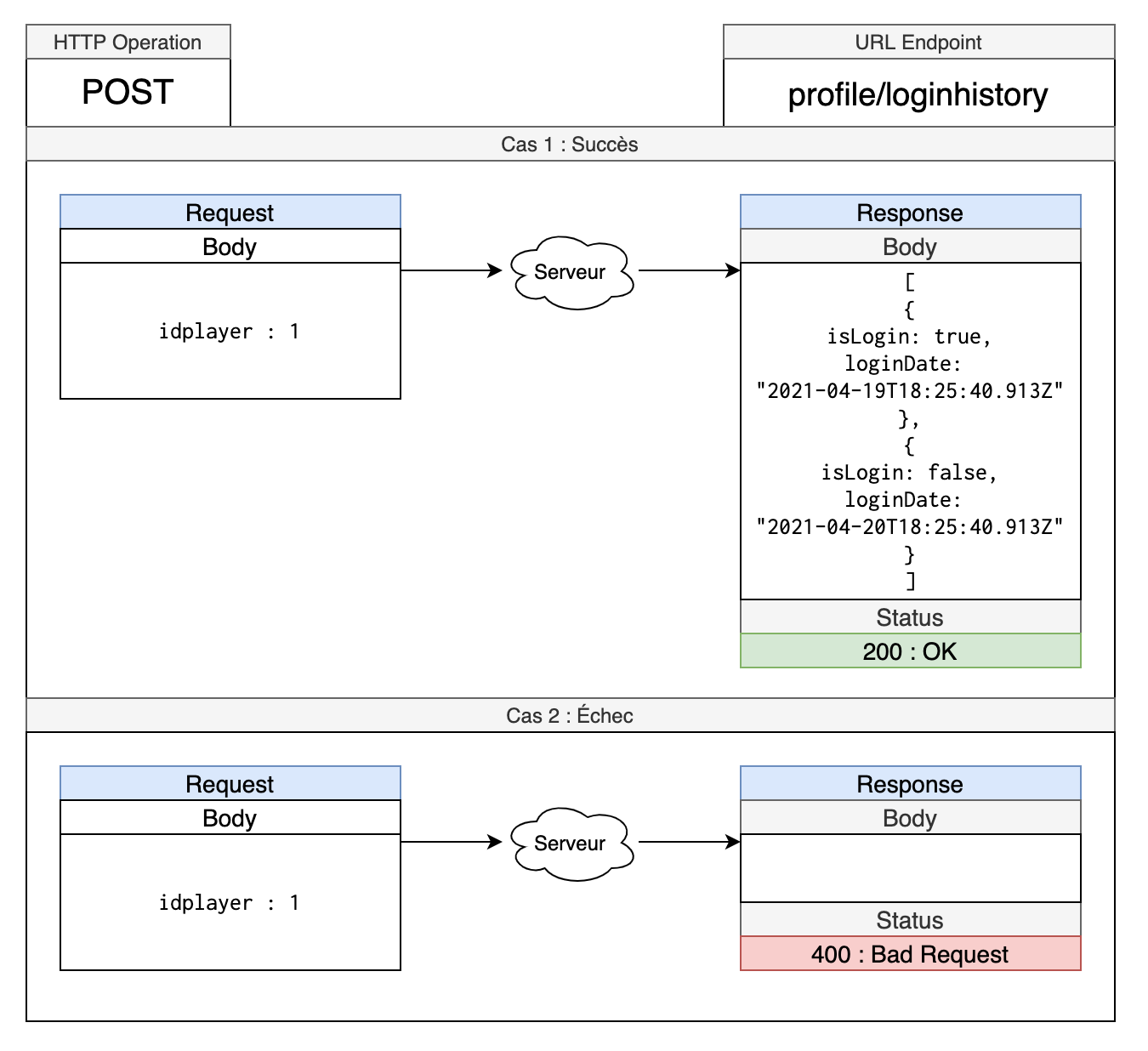
## 

## 3.11 Profile

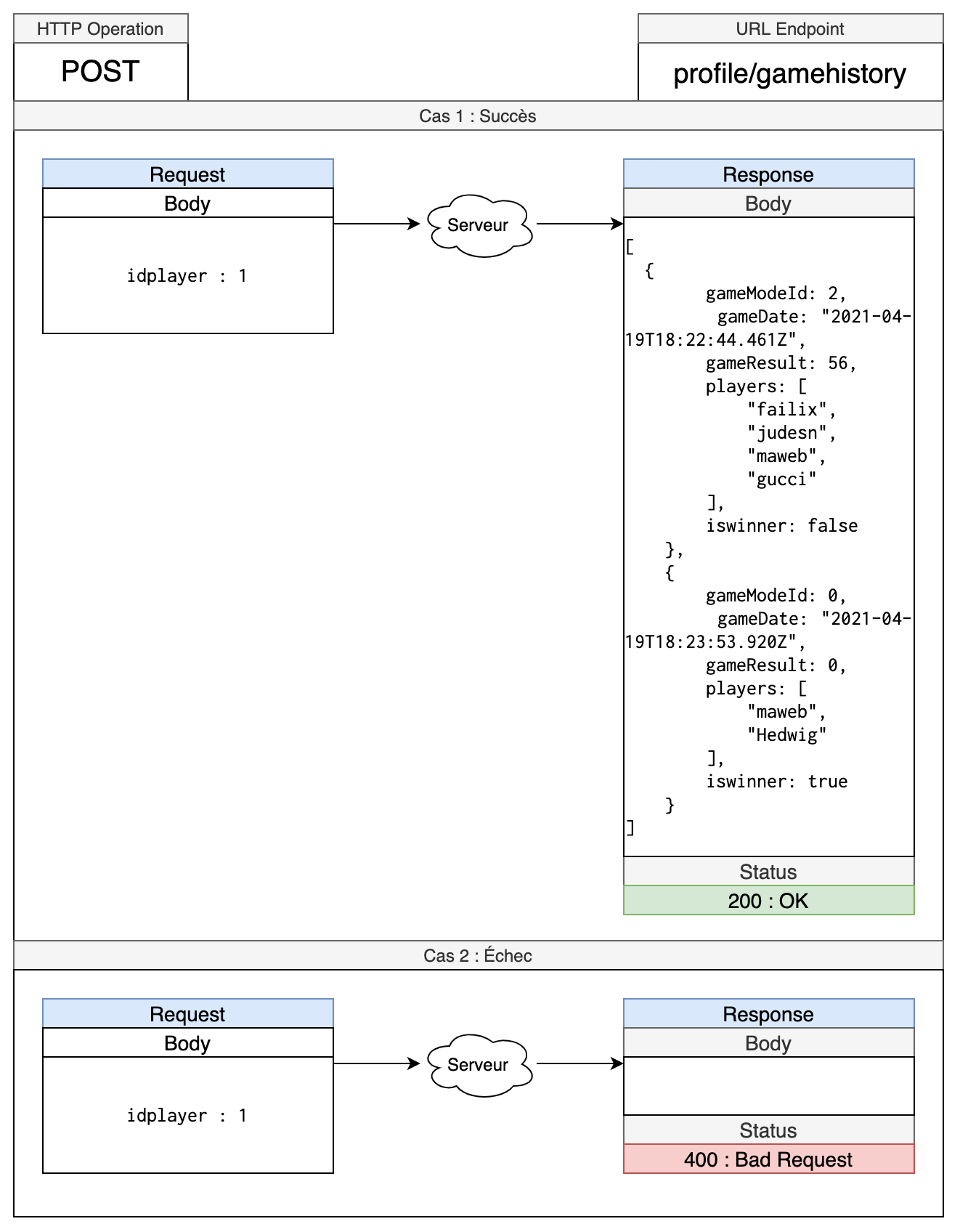
## 3.11.1 Identité de l’utilisateur



## 3.11.2 Historique de connexion



## 3.11.3 Historique de partie



## 3.11.4 Statistiques

