Protocoles Internet
TP 4 : WebSocket

Juliusz Chroboczek

23 octobre 2023

Exercice 1. Écrivez un programme *Go* qui se connecte à la *WebSocket*

`wss://jch.irif.fr:8082/chat/ws`

puis envoie un message texte contenant la chaîne « `J'adore ce TP` ». Il lira ensuite les messages reçus jusqu'à ce que le serveur ferme la *WebSocket*.

Il faudra importer le paquet

```
github.com/gorilla/websocket
```

puis construire un `Dialer` à l'aide de

```
d := websocket.Dialer{}
```

Si le serveur utilise un certificat anonyme, il faudra faire :

```
d := websocket.Dialer{
    TLSClientConfig: &tls.Config{
        InsecureSkipVerify: true,
    },
}
```

Exercice 2. Modifiez votre programme pour qu'il envoie un message de type texte contenant la chaîne

```
{"type: "get", "count": 20}
```

puis affiche les messages reçus, toujours sans les décoder.

Exercice 3. Le protocole utilisé par le serveur manipule des messages JSON qui peuvent être décrits en Go par la structure suivante :

```
type jsonMessage struct {
    Type     string        `json:"type"`
    Message  string        `json:"message,omitempty"`
    Messages []chatMessage `json:"messages,omitempty"`
    Count    int           `json:"count,omitempty"`
    Error    string        `json:"error,omitempty"`
}

type chatMessage struct {
        Id   string `json:"id,omitempty"`
        Time int64  `json:"time,omitempty"`
        Body string `json:"body"`
}
```

Modifiez votre programme pour qu'il utilise les méthodes `WriteJSON` et `ReadJSON` pour afficher les messages sous forme lisible par un être humain. Assurez-vous de gérer les messages de type `error` ainsi que les erreurs retournées par les fonctions que vous utilisez.

Exercice 4. Le protocole implémenté par le serveur consiste des requêtes suivantes :

```
{"type": "get", "count": n}
{"type": "subscribe", "count": n}
{"type": "unsubscribe"}
{"type": "post", "message": message}
```

La requête `get` retourne les *n* derniers messages. La requête `subscribe` demande au serveur d'envoyer les messages au serveur au fur et à mesure qu'ils sont postés ; si le champ `count` est présent, elle retourne aussi les *n* derniers messages. Enfin, la requête `post` ajoute un message au *chat*.

1. Modifiez votre programme pour qu'il affiche les 20 derniers messages puis affiche les messages au fur et à mesure qu'ils arrivent.
2. Pourquoi la requête `subscribe` retourne-t-elle les derniers messages ? Quel problème y aurait-il si on utilisait la requête `get` avant de faire un `subscribe` ?
3. Que se passe-t-il si on supprime un message à l'aide d'une requête `DELETE` pendant qu'un client est abonné ?
4. Écrivez un programme qui lit des messages au clavier et les ajoute au *chat*.
5. Quel problème faut-il résoudre pour avoir un seul programme qui permet d'ajouter les messages au *chat* et aussi d'afficher les messages qui arrivent ? Proposez une solution, et, s'il vous reste du temps, implémentez la.