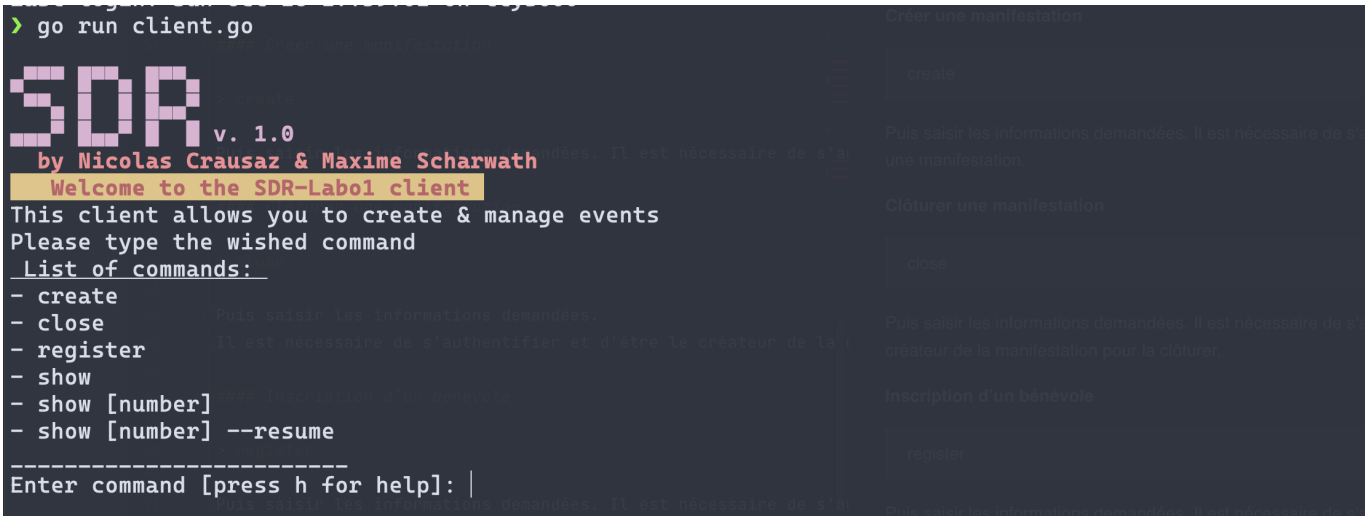
SDR 2022 / Labo 1 - Programmation répartie

Nicolas Crausaz & Maxime Scharwath



Configuration

La configuration du serveur et du client sont séparées dans deux fichiers différents.

La configuration du serveur se trouve dans `server.json`:

```
  "host": "localhost",    // IP / nom DNS du serveur
  "port": 9000,           // Port d'écoute du serveur
  "debug": false,         // Mode de debug de la concurence, ralenti les
entrées en section critique
  "showInfosLogs": false, // Active l'affichage des données brutes lors
des communications
  "users": [...],         // Utilisateurs enregistrés
  "events": [...]         // Evénements enregistrés
```

La configuration du client se trouve dans `client.json`:

```
{
  "srvHost": "localhost",  // IP / nom DNS du serveur
  "srvPort": 9000,         // Port d'écoute du serveur
  "showInfosLogs": false,  // Active l'affichage des données brutes lors
des communications
}
```

Utilisation

> **Attention** Pour une utilisabilité optimale, il est recommandé d'utiliser un terminal qui supporte les couleurs et les emojis. Fonctionne sur Windows Terminal, terminal de MacOS et Linux.

Lancer le serveur (directement, ou via un exécutable)

```
go run server.go
ou
go build server.go && ./server
```

Le serveur attendra des connexions sur le port TCP configuré dans `server.json`

Lancer un client (directement, ou via un exécutable)

```
go run client.go
ou
go build client.go && ./client
```

Le client se connectera au serveur configuré dans `client.json`

Liste de commandes disponible

Créer une manifestation

```
create
```

Puis saisir les informations demandées. Il est nécessaire de s'authentifier pour créer une manifestation.

Clôturer une manifestation

```
close
```

Puis saisir les informations demandées. Il est nécessaire de s'authentifier et d'être le créateur de la manifestation pour la clôturer.

Inscription d'un bénévole

```
register
```

Puis saisir les informations demandées. Il est nécessaire de s'authentifier. Il est possible d'être inscrit qu'à un seul poste par manifestation, l'inscription la plus récente sera conservée.

Liste des manifestations

```
show
```

Affiche l'état de toutes les manifestations.

```
-------------------------
Enter command [press h for help]: show
Number          Name                    Organizer name          open
1               Fête nationale          user1                   yes
2               Festival                nicolas                 yes
```

Informations d'une manifestation

> show <numéro manifesation>

Affiche les informations d'une manifestation

```
Enter command [press h for help]: show 2
Event #2: Festival
List of jobs:
Number          Name            Max capacity
1               Bar             20
2               Tickets         30
```

Répartition des postes pour une manifestation

> show <numéro manifesation> --resume

Affiche l'état des postes d'une manifestation.

```
Enter command [press h for help]: show 2 --resume
Event #2: Festival
Current board of registrations
                Bar#1 (1/20)    Tickets#2 (1/30)
user1                           x
nicolas         x
```

Protocole de communication

Le protocole de communication est basé sur le protocole TCP. Les messages sont sérialisés en JSON.

1. Le client envoie une chaîne de caractères au serveur appelée `Endpoint` qui indique la fonctionnalité à appeler.
2. Le serveur va répondre avec un message de type `Header` qui indique si le `Endpoint` existe et si l'utilisateur doit être authentifié.
3. Si la requête nécessite une authentification, le client envoie un message de type `Credentials` avec les identifiants de l'utilisateur.
4. Si l'authentification est réussie, le serveur envoie un message de type `AuthResponse` qui indique si les identifiants sont valides.
5. Le client envoie n'importe quel type de message qui correspond à la fonctionnalité demandée.
6. Le serveur envoie un message de type `Response` qui indique si la requête a été traitée avec succès ou non et contient le résultat de la requête.

Les données sont envoyées sur le réseau sous forme de chaînes de caractères finissant par un caractère de fin de ligne `\n`.

Nous utilisons des structures de type DTO (Data Transfer Object) pour sérialiser les données afin de faciliter la lecture et l'écriture des messages.

Tests

Intégration

> **Attention** Les tests vont créer des serveurs et des clients qui vont communiquer entre eux. Il est donc nécessaire de s'assurer que le port `9001` soit disponible.

Pour exécuter les tests, lancez la commande :

```
go test ./tests/integration_test.go
```

Concurrence

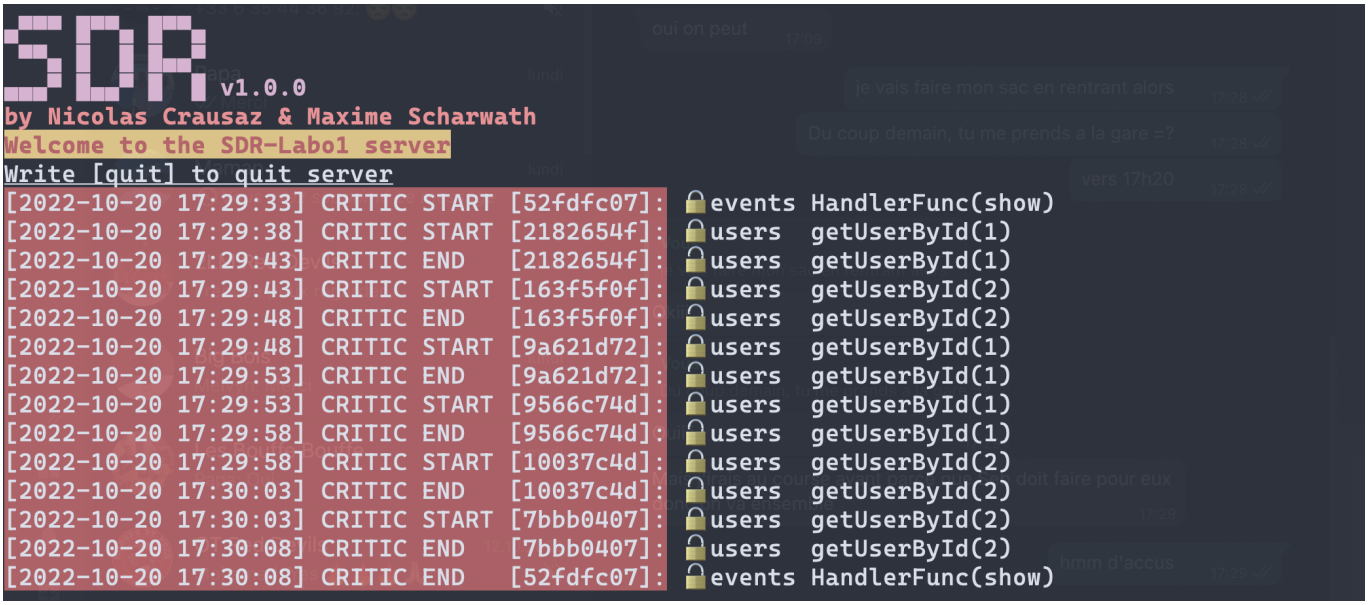
Pour effectuer des tests manuels sur la concurrence et sur le protocole, modifiez la configuration du serveur pour ralentir les entrées en zones critiques :

```
"debug": false
```

Il suffit ensuite de démarrer un serveur et plusieurs clients

```
go run server.go
go run client.go (selon le nombre de clients souhaités)
```

En exécutant des commandes depuis les clients, on peut observer les entrées / sorties en sections critiques sur le serveur:



Limitations

Il n'y a pas de persistance des données au-delà de l'exécution du serveur.