SR05 - Projet - README

Plateforme d'achat en ligne

Groupe 23 : HU Ruiqi, LI Chenxin, TIAN Linxiao, WANG Hongzhe

1. Introduction

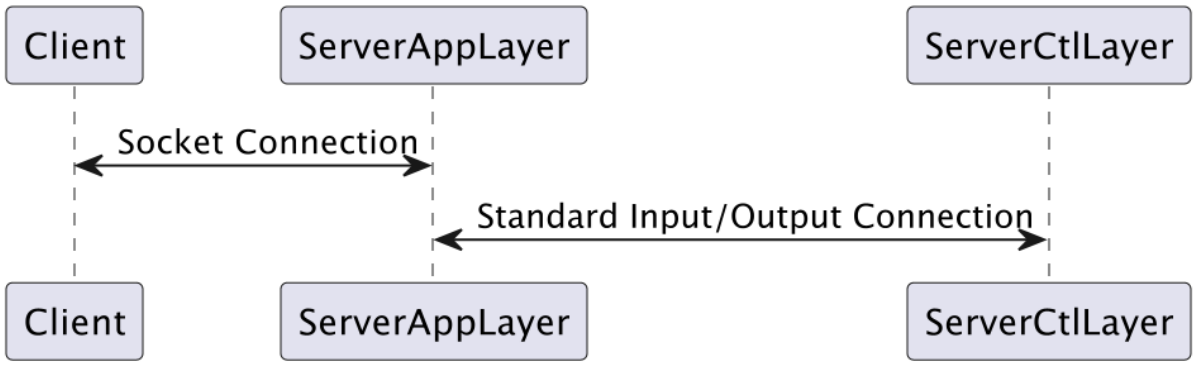
La scène de la conception de notre projet est une plateforme d'achat en ligne. Il s'agit d'une application distribuée,

La donnée partagée entre les différents sites est la quantité restante de marchandises. L'application a deux fonctions principales.Tout d'abord, la fonction de shopping implémente un algorithme de fichier d'attente distribuée qui organise l'exclusion mutuelle. Un seul site est autorisé à la fois et va modifier la copie des données (shopping) et la diffuser. L'algorithme utilise Estampilles. De plus, notre application comprend une fonctionnalité de sauvegarde distribuée obsolète (voir les enregistrés d'achat). Nous utilisons Algorithme avec marqueur pour créer des instantanés. À cette fin, nous avons également ajouté des horloges vectorielles pour les sites distribués.

En ce qui concerne l'exécution du programme, nous avons configuré la version Web du client pour se connecter afin de créer une Websocket et de se connecter à chaque site distribué. Chaque site utilise une architecture qui sépare les fonctions applicatives des fonctions de contrôle Nous avons construit un réseau unidirectionnel en forme d'anneau de trois sites pour la plate-forme de test.

2. Architecture du programme

Chacun de nos sites est constitué de clients et de serveurs, voici le schéma d'architecture



Client

Afin d'assurer la polyvalence et la portabilité du programme d'application, et de faciliter les tests d'application et l'utilisation par les utilisateurs, nous avons conçu un client Web avec une interface graphique pour le programme d'application. Il est principalement utilisé pour envoyer des requêtes formatées à la couche APP du serveur et afficher les résultats du serveur à l'utilisateur sous une forme facile à lire. Sa fonction principale est de se connecter au serveur, d'afficher l'horodatage du site, de vérifier l'inventaire, d'envoyer une demande d'instantané, d'afficher le résultat de l'instantané, d'envoyer une demande d'instantané et d'afficher le résultat de l'instantané. Afin de réaliser ces fonctions, nous avons un code HTML simple, un embellissement CSS et JS pour obtenir des effets dynamiques.

Connexion par websocket

La communication front-end et back-end est réalisée via Websocket.

La partie socket du frontal est implémentée à l'aide d'un objet de classe websocket et de ses méthodes en JavaScript.

```
var ws;

document.getElementById("connecter").onclick = function (evt) {

   if (ws) {

       return false;

   }

   var host = document.getElementById("host").value;

   var port = document.getElementById("port").value;

   addToLog("Tentative de connexion");

   addToLog("host = " + host + ", port = " + port);

   ws = new WebSocket("ws://" + host + ":" + port + "/ws");

   ws.onopen = function (evt) {

       addToLog("Websocket ouverte");

   }

   ws.onclose = function (evt) {

       addToLog("Websocket fermée");

       ws = null;

   }

   ws.onerror = function (evt) {
```

```
        addToLog("Erreur: " + evt.data);

    }
```

Les deux fonctions clés sont :

1.ws.onmessage écoute les données au format Json envoyées par le backend et les analyse et les traite.

```
ws.onmessage = function (evt) {

        const data = JSON.parse(evt.data);

        //...Traitement des données

        addToLog("Réception: " + evt.data);

    }
```

2. L'événement de liaison de bouton frontal lit la valeur d'entrée du formulaire, la convertit au format Json et l'envoie au backend.

```
document.getElementById('submit').onclick = function (evt) {

        const input = document.getElementById('count');

        const price = input.value;

        if (price !== ''){

        //Obtenir la quantité pour acheter et réinitialiser l'entrée

            addToLog("success achat");

            input.value = '';

            const data = {number: price};

            addToLog("Pret: " + data.number);

            const jsonData = JSON.stringify(data);

            // Envoyer des données directement

            ws.send(jsonData);

            addToLog("Emission: " + data.number);

        }

        return false;

    }

document.getElementById('snapshot').onclick = function (evt) {
```

```
        const data = {text:"demand snapshot"};
        const jsonData = JSON.stringify(data);
        // Envoyer une demande d'instantané
        ws.send(jsonData);
        addToLog("Emission: " + data.text);
        return false;
}
```

En back-end, toutes les fonctions liées à la communication socket dans la couche App sont écrites dans serveur.go.

La fonction do_websocket est utilisée pour répondre aux demandes de connexion socket et lancer deux go_routines, ws_receive et receive.

```
// Fonction pour gérer une connexion WebSocket
func do_websocket(w http.ResponseWriter, r *http.Request) {
  // Configurer l'upgrader pour la connexion WebSocket
  var upgrader = websocket.Upgrader{
     CheckOrigin: func(r *http.Request) bool { return true },
  }
  conn, err := upgrader.Upgrade(w, r, nil)
  if err != nil {
     fmt.Println("upgrade:", err)
     return
  }
  // Afficher un message de connexion réussie
  display_d("ws_create", "Client connecté")
  ws = conn
  go receive()
  go ws_receive()
}
```

La fonction ws_send permet d'envoyer des données JSON au format fixe au client.

```
// Fonction pour envoyer des messages WebSocket
func ws_send(text, mylock string) {
  msg := &myData{
     Number:       strconv.Itoa(stock),
     Text:         text,
     MyLock:       mylock,
     Horloge:      strconv.Itoa(horloge),
     Snapshot:     "",
     Snapshottime: "",
  }
  err := ws.WriteJSON(msg)
  if err != nil {
     display_d("write:", string(err.Error()))
     return
  }
}
```

La fonction ws_receive est utilisée pour recevoir des données JSON, analyser le type de requête et envoyer la requête correspondante à la couche ctl.

```
// Fonction pour recevoir des messages WebSocket
func ws_receive() {
  // Fermer la connexion WebSocket à la fin de la réception
  defer ws_close()
  // Inverser le signe du nom pour désigner le récepteur
  nom *= -1
  // Envoyer un message de démarrage au client
  ws_send("start", status)
  l := log.New(os.Stderr, "", 0)
  for {
     _, message, err := ws.ReadMessage()
```

```
    if err != nil {
        l.Println("read:", err)
        return
    }
    mutex.Lock()
    var data myData
    err = json.Unmarshal(message, &data)
    if err != nil {
        l.Println("unmarshal:", err)
        return
     } // Si le message est une demande de snapshot, envoyer une demande au
serveur
    if data.Text == "demand snapshot" {
                msg_send(msg_format("receiver",  strconv.Itoa(nom*(-1)))  +
msg_format("type", "demandeSnap") + msg_format("sender", strconv.Itoa(nom)) +
msg_format("hlg", strconv.Itoa(horloge)))
        status = "locked"
              ws_send("L'instantané est en cours de génération, veuillez
patienter", status)
        mutex.Unlock()
        continue
    }
    count, _ = strconv.Atoi(data.Number)
    l.Printf("Received message: %d\n", count)
    // Envoyer une demande d'exclusion mutuelle au serve
                msg_send(msg_format("receiver",   strconv.Itoa(nom*(-1)))   +
msg_format("type", "demandeSC") + msg_format("sender", strconv.Itoa(nom)) +
msg_format("hlg", strconv.Itoa(horloge)))
    status = "locked"
    ws_send("Attendre en ligne s'il vous plaît soyez patient", status)
    mutex.Unlock()
  }
}
```

Serveur

Notre serveur est divisé en couche d'application et couche de contrôle

Couche d’application

Cette couche est principalement responsable de la communication avec le web et de la mise en œuvre des fonctions liées à la scène : instantanés, shopping. Afin d'assurer l'exécution séquentielle du site, nous nous référons à la structure de AC4 et ajoutons des verrous mutex aux deux goroutines, ws_receive et receive.

Couche de contrôle

Cette couche implémente principalement deux algorithmes distribués et deux horloges. Algorithme de la file d'attente répartie, estampilles, Algorithme de calcul d'instantanés, horloges vectorielles.

En fait, les fonctions de la couche de contrôle sont universelles.Pour différents scénarios d'application, il suffit de remplacer le contenu de la couche APP pour atteindre la portabilité.

Connexion par I/O standard

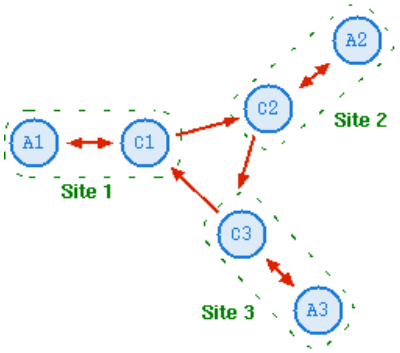
La communication entre les deux couches est mise en œuvre par la redirection des entrées et sorties standards. Nous utilisons les commandes pipes, cat et tee pour copier les messages sur plusieurs pipes et on précise le format du message. On divise chaque paire clé-valeur avec /, puis divise la clé et la valeur avec =.

Les horloges (estampilles) de la couche app et de la couche ctl sont mêmes, et sont réalisées la synchronisation en fonction du message de type *updateHorloge* et autres messages envoyés par la couche ctl.

Reaseau du systeme repartie

Nous utilisons des scripts shell pour nous occuper de la construction des sites individuels et des pipelines de communication inter-sites.

La structure que nous prenons est un réseau en anneau unidirectionnel.



```
mkfifo /tmp/in_A1 /tmp/out_A1

mkfifo /tmp/in_C1 /tmp/out_C1

mkfifo /tmp/in_A2 /tmp/out_A2

mkfifo /tmp/in_C2 /tmp/out_C2

mkfifo /tmp/in_A3 /tmp/out_A3

mkfifo /tmp/in_C3 /tmp/out_C3


./app -n 1 -p 4444 < /tmp/in_A1 > /tmp/out_A1 &

./ctl -n 1 < /tmp/in_C1 > /tmp/out_C1 &

./app -n 2 -p 5555 < /tmp/in_A2 > /tmp/out_A2 &

./ctl -n 2 < /tmp/in_C2 > /tmp/out_C2 &

./app -n 3 -p 7777 < /tmp/in_A3 > /tmp/out_A3 &

./ctl -n 3 < /tmp/in_C3 > /tmp/out_C3 &


cat /tmp/out_A1 > /tmp/in_C1 &

cat /tmp/out_C1 | tee /tmp/in_A1 > /tmp/in_C2 &

cat /tmp/out_A2 > /tmp/in_C2 &

cat /tmp/out_C2 | tee /tmp/in_A2 > /tmp/in_C3 &

cat /tmp/out_A3 > /tmp/in_C3 &

cat /tmp/out_C3 | tee /tmp/in_A3 > /tmp/in_C1 &
```

Afin de bien arrêter tout le système, nous appelons la commande destruct, nous pouvons appuyer sur Ctrl+C pour terminer le programme et supprimer le pipeline.

```
# Fonction de nettoyage

nettoyer () {

 # Suppression des processus de l'application app

 killall app 2> /dev/null

 # Suppression des processus de l'application ctl

 killall ctl 2> /dev/null

 # Suppression des processus tee et cat
```

```
 killall tee 2> /dev/null
 killall cat 2> /dev/null
 # Suppression des tubes nommés
 \rm -f /tmp/in* /tmp/out*
 \rm -f error.log
 exit 0
}
# Appel de la fonction nettoyer à la réception d'un signal
```

3. Fonctionnment

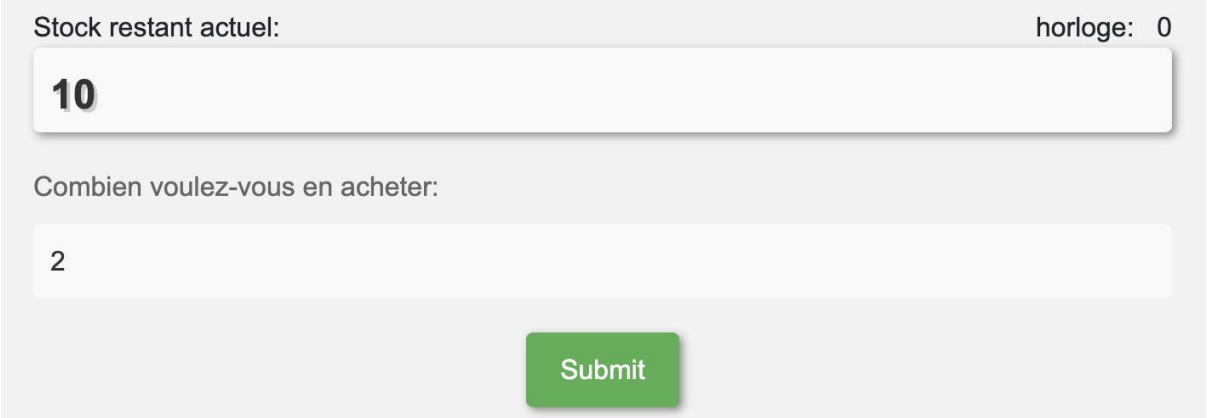
Interface visuelle utilisateur

Gestion de connexion:

C’est un formulaire permettant de saisir le nom de la machine (host) et le numéro de port.

Ces deux boutons, “Connecter” et “Fermer”, permettent d'ouvrir ou de fermer la websocket avec la couche d’application.

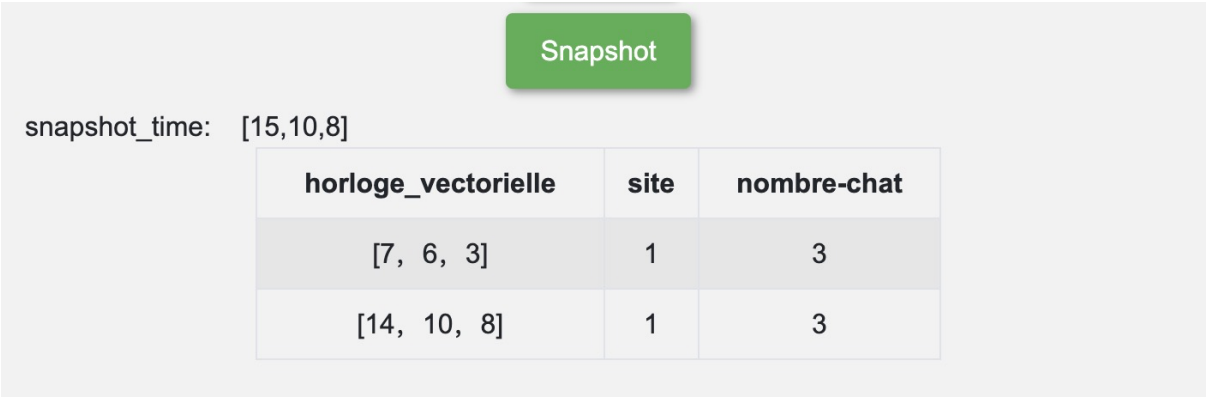
Gestion de donnée partagée



Cette section est chargée de rafraîchir et d'afficher le valeur de la donnée partagée et le valeur de horloge du site, ainsi que fournir la fonctionnalité d'envoyer des demandes de modification de donnée à la couche de présentation.

Il convient de noter que lorsque le stock est égal à 0 ou que la demande de modification de donnée n'a pas encore été exécutée, la zone de saisie est verrouillée afin d'éviter la soumission de données en double.

Snapshot



Ce bouton "Snapshot" permet au côté client d'envoyer une demande de “snapshot” à la couche d'application. Après, La page affiche des informations sur les instantanés et sur l'horloge vectorielle du dernier instantané.

Impression des journaux

Logs
Tentative de connexion
host = localhost, port = 4444
Websocket ouverte
Réception:
{"number":"10","text":"start","mylock":"unlocked","horloge":"0","snapshot":"","snapshot_time":""}
Réception: {"number":"10","text":"mettre à jour
l'horloge","mylock":"unlocked","horloge":"2","snapshot":"","snapshot_time":""}
success achat
Pret: 1
Emission: 1
Réception: {"number":"10","text":"Attendre en ligne s'il vous plaît soyez
patient","mylock":"locked","horloge":"2","snapshot":"","snapshot_time":""}
Réception: {"number":"10","text":"mettre à jour

Cette section est chargée d'afficher les informations de communication entre le côté client et la couche d'application, y compris la connexion de websocket, la réception de données et l'émission de données, etc.

Donnée partagée entre les sites

Nous avons implémenté "Algorithme de la file d'attente répartie Fichier" pour garantir qu'un seul site à la fois obtient le droit d'achat.

Dans la couche de contrôle

Nous avons une structure de données appelée "tab" pour chaque site, qui stocke l'état et le timestamp de tous les sites. Lorsque nous recevons un signal, l'état et le timestamp de chaque site dans "tab" sont mis à jour en fonction de l'algorithme correspondant.

```
type site struct {

   id          int

   logicalTime int

   tab         [N + 1][2]int

}
```

Tout d'abord, dans la fonction run(), la couche de contrôle du site lit un nouveau message à partir de l'entrée standard. Alors les caractères non imprimables sont supprimés de la chaîne, puis le message reçu est divisé en paires clé-valeur multiples. Ensuite, selon la fonction findval pour analyser le contenu du message et assigner des valeurs aux variables correspondantes.

Ensuite, handleMessage est appelé pour traiter le message reçu. handleMessage recevra 5 types de messages (peut être envoyé par une couche d'application ou par une autre couche de contrôle sur un autre site), donc nous avons déclaré une variable énumérée messageType: demande, libération, ack,demandeSC et finSC. Ainsi que demandeSnap et finSnap utilisés pour générer des instantanés.

```
const (
    request messageType = iota
    release
    ack
    demandeSC
    finSC
    demandeSnap
    finSnap
)
```

Dans la transmission de messages, nous avons défini une structure "message" pour stocker toutes les informations du message dans une variable, afin de faciliter l'appel des fonctions. msgType stocke le type de message, logicalTime est le timestamp inclus dans le message, sender est le site émetteur (un nombre positif pour un message émis par la couche control et négatif pour un message correspondant à la couche application), receiver est le site récepteur, count représente le nombre restant d'articles et h1,h2,h3 sont des vecteurs temps utilisés pour générer une capture instantanée.

```
type message struct {
    msgType     messageType
    logicalTime int
    sender      int
    receiver    int
    count       int
    h1          int
    h2          int
    h3          int
}
```

Dans la couche d’application

Similaire à la couche de contrôle, nous lisons les messages depuis l'entrée standard dans la fonction receive. Ensuite, nous appelons la fonction handleMessage pour traiter le message reçu.

Comme pour la couche de contrôle, nous énumérons les types de messages reçus : updateSC, permetSC, updateHorloge et donneSnap.

```
const (
   updateSC messageType = iota
   permetSC
   updateHorloge
   donneSnap
)
```

Et la structure de message utilisée pour la transmission entre les fonctions, où snapshot contient des informations instantanées et snapshot_time est le vecteur d'horloge au moment de l'instantané.

```
type message struct {
   msgType       messageType
   count         int
   snapshot      string
   snapshot_time string
}
```

Nous avons écrit une fonction standard pour envoyer des messages afin d'améliorer la lisibilité du code.

```
var fieldsep = "/"
var keyvalsep = "="

func msg_format(key string, val string) string {
   return fieldsep + keyvalsep + key + keyvalsep + val
}
```

```
func msg_send(msg string) {

   fmt.Print(msg + "\n")

}
```

Et a écrit la fonction findval pour trouver la valeur correspondante de la clé dans le message.

```
func findval(msg string, key string) string {

   if len(msg) < 4 {

      return ""

   }


   sep := msg[0:1]

   tab_allkeyvals := strings.Split(msg[1:], sep)


   for _, keyval := range tab_allkeyvals {

      //l := log.New(os.Stderr, "", 0)

      //l.Printf(keyval)

      if len(keyval) >= 4 {

         equ := keyval[0:1]

         tabkeyval := strings.Split(keyval[1:], equ)

         if tabkeyval[0] == key {

            return tabkeyval[1]

         }

      }

   }


   return ""

}
```

Dans la communication de web-socket

Pour l'interaction de données entre la couche d'application et les pages web, nous avons utilisé une communication par web-socket.

Pour faciliter la communication via socket, nous avons défini une structure myData pour stocker les informations des messages.

```
type myData struct {
   Number       string `json:"number"`
   Text         string `json:"text"`
   MyLock       string `json:"mylock"`
   Horloge      string `json:"horloge"`
   Snapshot     string `json:"snapshot"`
   Snapshottime string `json:"snapshot_time"`
}
```

Au niveau de l'application, nous exécutons la fonction do_websocket pour démarrer la communication par socket et lançons le processus ws_receive dans cette fonction pour traiter les informations envoyées par le frontend web.

Nous avons écrit la fonction ws_send pour normaliser les messages utilisés dans la communication par socket, où le contenu de la communication est envoyé et reçu au format json.

```
func ws_send(text, mylock string) {
   msg := &myData{
      Number:       strconv.Itoa(stock),
      Text:         text,
      MyLock:       mylock,
      Horloge:      strconv.Itoa(horloge),
      Snapshot:     "",
      Snapshottime: "",
   }
   err := ws.WriteJSON(msg)
   if err != nil {
      display_d("write:", string(err.Error()))
      return
```

```
    }
}
```

Dans ws_receive, un processus similaire est utilisé pour extraire le message du format JSON.

```
func ws_receive() {
     for {
      _, message, err := ws.ReadMessage()
      if err != nil {
         l.Println("read:", err)
         return
      }
      mutex.Lock()
      var data myData
      err = json.Unmarshal(message, &data)
      if err != nil {
         l.Println("unmarshal:", err)
         return
      }
```

Sauvegarde répartie datée

Pour réaliser une capture d'écran, nous avons ajouté la variable "horloge_vec" dans la couche de contrôle du site pour stocker le vecteur d'horloge. Nous avons également ajouté les variables h1, h2 et h3 dans la structure de message pour transmettre le vecteur temps.

Chaque fois que nous recevons un message de type demandeSnap, finSnap, request, release ou ack, nous mettons à jour notre vecteur horaire.

```
// Mettre à jour l'horloge vectorielle
arr := []int{0, msg.h1, msg.h2, msg.h3}
horloge_vec = calVec(horloge_vec, arr)
horloge_vec[s.id] += 1
```

Nous avons écrit la fonction calVec pour mettre à jour le vecteur d'horloge. La valeur de l'indice 0 dans ce tableau n'a pas de sens, les valeurs des horloges vectorielles des trois sites 1, 2 et 3 correspondent respectivement à horlog_vec[1], horlog_vec[2] et horlog_vec[3].

```
// La fonction calVec calcule la nouvelle horloge vectorielle en comparant
deux horloges vectorielles

func calVec(x, y []int) []int {

   res := make([]int, 4)

   res[0] = 0

   res[1] = max(x[1], y[1])

   res[2] = max(x[2], y[2])

   res[3] = max(x[3], y[3])

   return res

}
```

Nous avons ajouté une variable de tableau de chaînes d'instantanés à chaque site pour stocker les informations d'achat. Chaque fois que la couche de contrôle reçoit un message finSC, les informations sur l'achat sont ajoutées à la fin du tableau instantané, y compris l'horloge vectorielle actuelle, le site et la quantité achetée.

```
snapshot=append(snapshot,
"*horloge_vectorielle:["+strconv.Itoa(horloge_vec[1])+",
"+strconv.Itoa(horloge_vec[2])+",
"+strconv.Itoa(horloge_vec[3])+"]*site:"+strconv.Itoa(s.id)+"*nombre_achat:"+
strconv.Itoa(count))
```

Réception: {"number":"2","text":"sauvegarde","mylock":"unlocked","horloge":"16","snapshot":"*horloge_vectorielle:[5, 2, 2]*site:1*nombre_achat:1@*horloge_vectorielle:[9, 4, 4]*site:1*nombre_achat:3@*horloge_vectorielle:[13, 6, 6]*site:1*nombre_achat:4","snapshot_time":"[14,6,6]"}

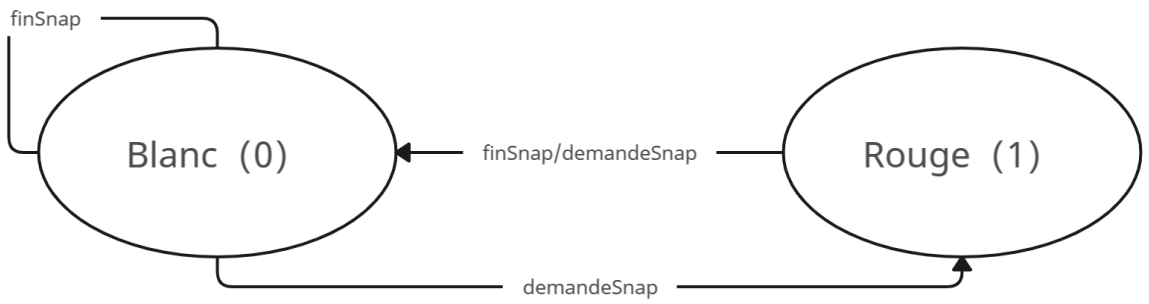
Il convient de mentionner que la couche de contrôle de chaque site ne sauvegardera que les enregistrements d'achat de son propre site. Cependant, si vous souhaitez que chaque site sauvegarde tous les enregistrements d'achat de tous les sites, il suffit simplement de diffuser vos propres enregistrements d'achat à tous les sites lors du finSC. Cela rendrait le système distribué moins "distribué", c'est pourquoi nous n'avons pas choisi cette option.

Pour réaliser l'algorithme de capture d'écran, nous avons ajouté une variable couleur à chaque site, qui est blanche (0) par défaut. Lorsque le message demandeSnap est reçu :

1. Si le site actuel est blanc (0), il changera la couleur de son propre site en rouge (1), enverra un message demandeSnap au prochain site. Il enverra également les instantanés précédemment stockés avec celui-ci à la couche application et les affichera sur la page Web.
2. Si le site actuel est rouge (1), étant donné que notre configuration de site est une boucle unidirectionnelle, cela signifie que c'est ce site qui a envoyé la demande de capture d'écran. Il enverra alors un message finSnap au prochain site et réinitialisera sa propre couleur à blanc (0).

Lorsqu'un message finSnap est reçu, seul si la couleur du propre site est rouge (1), elle sera modifiée en blanc (0) et un message finSnap sera envoyé au prochain site.

Avec cet algorithme, lors du processus de capture d'écran, l'interface utilisateur Web affiche les propres achats du client pour chaque station. Après avoir généré une capture d'écran, tous les sites reviennent à leur état initial blanc afin qu'il soit possible de capturer plusieurs fois des instantanés.



4. Utilisation

Utilisation du front-end

Connexion / Déconnexion

Veuillez entrer le numéro de port et l'adresse du serveur dans le formulaire sur la page, puis cliquez sur le bouton Connecter pour envoyer une connexion Websocket au serveur. Pour fermer la connexion, cliquez sur le bouton Fermer.

Acheter un produit

Pour acheter un produit, veuillez entrer la quantité du produit que vous souhaitez acheter dans le formulaire, puis cliquez sur le bouton Soumettre. Le client enverra une demande d'achat au serveur. Pendant cette période, veuillez patienter, puis vous pourrez afficher le résultat de l'achat dans la colonne du journal.

Afficher les enregistrements d'achat

Pour afficher les enregistrements d'achat, veuillez cliquer sur le bouton Instantané, le client enverra une demande au serveur pour obtenir des instantanés, et vous pourrez voir vos enregistrements d'achat fournis par le backend sur la page, l'horloge vectorielle correspondant à chaque enregistrement, et le heure à laquelle l'instantané a été obtenu Horloge vectorielle.

Utilisation du back-end

Dans les fichiers correspondant à la couche app et à la couche ctl du serveur, il y a des paramètres qui peuvent modifier la configuration.

Couche App

-p, numéro de port, valeur par défaut 4444.

-addr, adresse IP du serveur, valeur par défaut localhost.

-n, nom du site, un entier positif, Il doit être le même que le nom du site de la couche ctl.

Couche Ctl

-nom nom du site, un entier positif, Il doit être le même que le nom du site de la couche app.

ringapp.sh

Nous avons construit un réseau repartie par défaut de trois sites dans ringapp.sh pour vous aider à tester rapidement, vous pouvez le modifier.

Ensuite, vous pouvez modifier les paramètres de ligne de commande (adresse IP, numéro de port, nom du site) de la déclaration sur le démarrage du site dans ringapp.sh.

```
./app -n 1 -p 4444 < /tmp/in_A1 > /tmp/out_A1 &

./ctl -n 1 < /tmp/in_C1 > /tmp/out_C1 &

./app -n 2 -p 5555 < /tmp/in_A2 > /tmp/out_A2 &

./ctl -n 2 < /tmp/in_C2 > /tmp/out_C2 &

./app -n 3 -p 7777 < /tmp/in_A3 > /tmp/out_A3 &

./ctl -n 3 < /tmp/in_C3 > /tmp/out_C3 &
```

Déploiement en un clic

Nous avons inclus la compilation de chaque fichier go dans ringapp.sh

```
go build -o app app.go serveur.go

go build ctl.go
```

Tout ce que vous avez à faire est de télécharger notre programme, puis d'entrer dans le dossier, d'accorder les autorisations appropriées à ringapp.sh, puis d'exécuter la commande :

```
./ringapp.sh
```

Vous pouvez démarrer les serveurs des trois sites distribués en un clic pour suivre les demandes des clients.