🚀 Serveur WebRTC + Social + Marketplace

Serveur complet en **pure Go** (sans CGO) intégrant :

- ✅ WebRTC (audio/video) avec **Pion**
- ✅ Rooms audio, vidéo et data
- ✅ Système social (posts, likes, commentaires)
- ✅ Marketplace (articles, ventes, paiements)
- ✅ Pub/Sub temps réel avec **MsgPack**
- ✅ Backend **PocketBase 0.33.0**
- ✅ API REST + OpenAPI/Swagger

📦 Installation

Prérequis

- Go 1.21+
- Aucune dépendance CGO requise

Installation

```bash
# Cloner ou créer le projet
mkdir webrtc-social-server
cd webrtc-social-server

# Créer les fichiers
# main.go, go.mod (voir artifacts)

# Installer les dépendances
go mod download

# Lancer le serveur
go run main.go serve
```

Le serveur démarre sur `http://localhost:8090`

🏗️ Architecture

Collections PocketBase

1. posts

```
go

- user (relation users)
- categories (select multiple)
- isPublic (bool)
- type (select: html, reel, images, url)
- content (text)
- images (file, max 3)
- video (file mp4)
- article (relation articles, optional)
- action (select: none, buy, join, subscribe, read, listen)
- actionText (text)
- dataAction (json)
- likesCount (number)
- commentsCount (number)
```

2. articles

```
go

- title (text, required)
- desc (text)
- prixOriginal (number)
- prix (number, required)
- quantite (number, required)
- dueDate (date)
- images (file, max 3)
- user (relation users)
```

3. ventesArticle

```
go

- article (relation articles)
- montant (number)
- status (select: paye, encours, echec, annule)
- user (relation users)
- paiementDate (date)
- cancelDate (date)
- failDate (date)
- fromPost (relation posts)
```

4. operations

```
go

- user (relation users)
- vente (relation ventesArticle, optional)
- montant (number)
- operation (select: cashin, cashout)
- desc (text)
- status (select: paye, en_attente, encours, echec, annule)
```

5. likes

```
go

- user (relation users)
- post (relation posts)
- reaction (select: like, love, fire, wow, sad, angry)
- UNIQUE INDEX (user, post)
```

6. comments

```
go

- user (relation users)
- post (relation posts)
- content (text, required)
- parentComment (relation comments, optional)
```

7. rooms

```
go

- roomType (select: audio, video, data)
- name (text)
- creator (relation users)
- isPublic (bool)
- maxParticipants (number)
```

🌐 API REST

Authentification

Toutes les routes nécessitant une authentification utilisent PocketBase Auth:

```
http

Authorization: Bearer YOUR_TOKEN_HERE
```

Routes WebRTC

Créer une room

```http
POST /api/rooms
Content-Type: application/json

{
 "room_type": "audio",
 "name": "Ma Room Audio"
}

Response:
{
 "room_id": "20241115120000-1",
 "room_type": "audio",
 "name": "Ma Room Audio"
}
```

Rejoindre une room

```http
POST /api/rooms/:roomId/join
Authorization: Bearer TOKEN

Response:
{
 "participant_id": "20241115120001-2",
 "sdp": {
  "type": "offer",
  "sdp": "v=0..."
 }
}
```

Envoyer une réponse SDP

```http
POST /api/rooms/:roomId/participants/:participantId/answer
Authorization: Bearer TOKEN
Content-Type: application/json

{
 "type": "answer",
 "sdp": "v=0..."
}
```

Routes Social

Liker un post

```http
POST /api/posts/:postId/like?reaction=fire
Authorization: Bearer TOKEN

Response:
{
  "success": true,
  "like_id": "abc123"
}
```

Commenter un post

```http
POST /api/posts/:postId/comment
Authorization: Bearer TOKEN
Content-Type: application/json

{
  "content": "Super post! 🔥",
  "parent_comment": null
}

Response:
{
  "success": true,
  "comment_id": "def456"
}
```

Acheter un article

```http
POST /api/articles/:articleId/buy
Authorization: Bearer TOKEN

Response:
{
  "success": true,
  "vente_id": "ghi789"
}
```

Routes PocketBase Standards

Toutes les routes CRUD PocketBase sont disponibles :

```http
# Créer un post
POST /api/collections/posts/records
Content-Type: multipart/form-data

# Récupérer les posts
GET /api/collections/posts/records?page=1&perPage=20&sort=-created&expand=user,article

# Mettre à jour un post
PATCH /api/collections/posts/records/:id

# Supprimer un post
DELETE /api/collections/posts/records/:id
```

Events Temps Réel (SSE)

```http
GET /api/events/post_events
GET /api/events/sales
GET /api/events/reactions

Response (Server-Sent Events):
data: {"type":"new_post","post_id":"abc","user_id":"xyz"}

data: {"type":"like","post_id":"abc","reaction":"fire"}

data: {"type":"comment","post_id":"abc","comment":"Cool!"}
```

🔌 WebRTC DataChannel

Format des messages (MsgPack)

```go
type DataEvent struct {
    Type      string                 `msgpack:"type"`
    RoomID    string                 `msgpack:"room_id"`
    Data      map[string]interface{} `msgpack:"data"`
    Timestamp int64                  `msgpack:"timestamp"`
}
```

Types d'événements

Chat

javascript

```
{
  type: "chat",
  data: {
    message: "Hello!"
  }
}

// Broadcast reçu:
{
  type: "chat",
  room_id: "room_123",
  data: {
    from: "user_xyz",
    message: "Hello!"
  },
  timestamp: 1699999999
}
```

Réaction

javascript

```
{
  type: "reaction",
  data: {
    type: "👋"
  }
}
```

Événements système

javascript

```javascript
// Participant rejoint
{
  type: "participant_joined",
  data: {
    participant_id: "part_123",
    user_id: "user_xyz"
  }
}

// Participant quitte
{
  type: "participant_left",
  data: {
    participant_id: "part_123"
  }
}
```

💻 Utilisation Client

Client JavaScript (Vanilla)

javascript

```javascript
import { SocialAPIClient, WebRTCClient } from './client.js';

const social = new SocialAPIClient('http://localhost:8090', token);
const webrtc = new WebRTCClient('http://localhost:8090', token);

// Créer un post
const post = await social.createPost({
  user: userId,
  type: 'images',
  content: 'Mon nouveau post!',
  isPublic: true,
  images: [file1, file2]
});

// S'abonner aux événements
social.subscribeToEvents('post_events', (event) => {
  console.log('Nouvel événement:', event);
});

// Rejoindre une room audio
await webrtc.joinRoom('room_123', 'audio');
webrtc.sendChatMessage('Hello!');
```

Client React

```jsx
function PostFeed() {
  const [posts, setPosts] = useState([]);
  const client = new SocialAPIClient(SERVER_URL, token);

  useEffect(() => {
    client.getPosts().then(data => setPosts(data.items));

    client.subscribeToEvents('post_events', (event) => {
      if (event.type === 'new_post') {
        // Actualiser le feed
      }
    });
  }, []);

  const handleLike = (postId) => {
    client.likePost(postId, 'fire');
  };

  return (
    <div>
      {posts.map(post => (
        <Post key={post.id} post={post} onLike={handleLike} />
      ))}
    </div>
  );
}
```

🔥 Workflows Complets

1. Créer et vendre un produit

javascript

```javascript
// 1. Créer l'article
const article = await social.createArticle({
  title: 'iPhone 15 Pro',
  desc: 'Neuf, sous garantie',
  prixOriginal: 1200,
  prix: 999,
  quantite: 1,
  user: userId,
  images: [photo1, photo2]
});

// 2. Créer un post lié
const post = await social.createPost({
  user: userId,
  type: 'images',
  content: 'iPhone 15 Pro à vendre! État neuf 📱',
  article: article.id,
  action: 'buy',
  actionText: 'Acheter maintenant',
  isPublic: true,
  categories: ['tech'],
  images: [photo1, photo2]
});

// 3. Les utilisateurs peuvent acheter
await social.buyArticle(article.id);

// Le système crée automatiquement:
// - Une ventesArticle (status: encours)
// - Une operation (cashout pour l'acheteur)
// - Décrémente la quantité
```

2. Room audio avec chat

javascript

```javascript
const webrtc = new WebRTCClient(SERVER_URL, token);

// Callbacks
webrtc.onRemoteTrack = (stream) => {
  audioElement.srcObject = stream;
};

webrtc.onDataEvent = (event) => {
  if (event.type === 'chat') {
    addMessageToUI(event.data.from, event.data.message);
  }
};

// Rejoindre
await webrtc.joinRoom('room_123', 'audio');

// Envoyer un message
webrtc.sendChatMessage('Salut!');
```

3. Marketplace complet

javascript

```javascript
// Récupérer les articles disponibles
const posts = await social.getPosts(1, 20, 'article!=null && isPublic=true');

// Afficher avec filtres
posts.items.forEach(post => {
  const article = post.expand.article;
  if (article.quantite > 0) {
    displayProduct(post, article);
  }
});

// Acheter
await social.buyArticle(articleId);

// Vérifier mes achats
const operations = await social.getMyOperations();
const achats = operations.items.filter(op => op.operation === 'cashout');

// Vérifier mes ventes (en tant que vendeur)
const ventes = await social.getMySales();
```

📊 Pub/Sub Interne

Le serveur utilise un système pub/sub en mémoire pour la communication temps réel:

```go
// Topics disponibles
- "post_events"    // Nouveaux posts, likes, commentaires
- "sales"          // Achats, ventes
- "reactions"      // Réactions dans les rooms
- "notifications"  // Notifications générales
```

Publier un événement

```go
pubsub.Publish("post_events", PubSubMessage{
    Topic: "post_events",
    Payload: map[string]interface{}{
        "type": "new_post",
        "post_id": postID,
        "user_id": userID,
    },
})
```

S'abonner (côté serveur)

```go
ch := pubsub.Subscribe("post_events")
for msg := range ch {
    // Traiter l'événement
    log.Println(msg.Payload)
}
```

🔐 Sécurité

Authentification

- PocketBase gère automatiquement l'authentification JWT
- Routes protégées avec `apis.RequireRecordAuth()`

Permissions

Configurer les rules PocketBase pour chaque collection:

javascript

```
// Exemple: posts
listRule: "@request.auth.id != '' && (isPublic = true || user = @request.auth.id)"
viewRule: "@request.auth.id != '' && (isPublic = true || user = @request.auth.id)"
createRule: "@request.auth.id != ''"
updateRule: "user = @request.auth.id"
deleteRule: "user = @request.auth.id"
```

🚀 Déploiement

Production

bash

```
# Build
go build -o server main.go

# Lancer
./server serve --http=0.0.0.0:8090

# Avec variables d'environnement
export PB_ENCRYPTION_KEY="your-32-char-key-here"
./server serve
```

Docker

dockerfile

```
FROM golang:1.21-alpine AS builder
WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download
COPY . .
RUN go build -o server main.go

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /app/server .
EXPOSE 8090
CMD ["./server", "serve", "--http=0.0.0.0:8090"]
```

Nginx Reverse Proxy

```nginx
upstream webrtc_backend {
    server 127.0.0.1:8090;
}

server {
    listen 443 ssl http2;
    server_name your-domain.com;

    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;

    location / {
        proxy_pass http://webrtc_backend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api/events/ {
        proxy_pass http://webrtc_backend;
        proxy_buffering off;
        proxy_cache off;
        proxy_set_header Connection '';
        chunked_transfer_encoding off;
    }
}
```

📈 Performance

Optimisations

1. **Indexes** : Créés automatiquement sur `(user, post)` pour les likes
2. **Pagination** : Toutes les listes supportent la pagination
3. **Expand** : Utiliser l'expand pour charger les relations en une requête
4. **Caching** : PocketBase cache automatiquement les requêtes fréquentes

Scalabilité

- **WebRTC** : Architecture SFU pour broadcaster les streams
- **Pub/Sub** : En mémoire (pour production, utiliser Redis)
- **Base de données** : SQLite par défaut (pour production, utiliser PostgreSQL)

🧪 Tests

```bash
# Test d'intégration
curl -X POST http://localhost:8090/api/rooms \
  -H "Content-Type: application/json" \
  -d '{"room_type":"audio","name":"Test Room"}'

# Test SSE
curl -N http://localhost:8090/api/events/post_events

# Test PocketBase API
curl http://localhost:8090/api/collections/posts/records
```

📚 Documentation API

OpenAPI/Swagger disponible sur :

```
GET /api/openapi
```

Import dans Postman ou Swagger UI pour tester l'API.

🤝 Contribution

- Pure Go, aucun CGO
- Respecter la structure PocketBase 0.33.0
- Utiliser MsgPack pour les DataChannels
- Documenter les nouveaux endpoints

📄 Licence

MIT License

🎯 Roadmap

- [ ] Support Redis pour Pub/Sub distribué
- [ ] Support PostgreSQL
- [ ] Transcription audio avec Whisper
- [ ] Modération IA des posts
- [ ] Analytics temps réel
- [ ] Mobile SDK (iOS/Android)
- [x] **Interpréteur TypeScript/JavaScript avec Goja**

🔧 TypeScript Interpreter

Le serveur inclut un **interpréteur TypeScript/JavaScript complet** avec Goja qui permet d'exécuter du code côté serveur avec accès à toutes les APIs:

📁 Structure des dossiers

```
project/
├── pb_hooks/        <- Scripts hooks (contextes indépendants)
│   ├── on-post-create.js
│   ├── moderation.js
│   └── rewards.js
│
├── pb_modules/      <- Modules partagés (espace commun)
│   ├── counter.js   <- État partagé entre tous les scripts
│   ├── logger.js
│   └── cache.js
│
└── main.go
```

⚡ Fonctionnement

1. **Scripts Hooks** (`pb_hooks/`)
   - Chaque script a son **propre contexte isolé**
   - S'exécute automatiquement en arrière-plan au démarrage
   - Peut importer des modules partagés avec `require()`
   - Hot-reload automatique quand le fichier change
2. **Modules Partagés** (`pb_modules/`)
   - **État partagé** entre TOUS les scripts
   - Une seule instance par module pour toute l'application
   - Exemple: `counter.inc()` incrémente la même variable partout

📦 Exemple Module Partagé

javascript

```javascript
// pb_modules/counter.js
let i = 0;

function inc() {
  i++;
  return i;
}

function get() {
  return i;
}

exports.inc = inc;
exports.get = get;
```

🎯 Exemple Hook

javascript

```javascript
// pb_hooks/on-post-create.js
const counter = require("counter"); // Module partagé
const logger = require("logger");

function main() {
  logger.info("Hook initialized");

  pubsub.subscribe("post_events", function(data) {
    const event = utils.jsonDecode(data);

    if (event.type === "new_post") {
      const count = counter.inc(); // Incrémente le compteur global
      logger.info(`Total posts: ${count}`);
    }
  });
}

main(); // Auto-démarrage
```

APIs Exposées

- ✅ **db** - CRUD complet
- ✅ **webrtc** - Gestion rooms
- ✅ **pubsub** - Pub/Sub temps réel
- ✅ **social** - Likes, comments, trending
- ✅ **utils** - JSON, MsgPack, UUID
- ✅ **cron** - Tâches planifiées
- ✅ **require()** - Import modules partagés

🔄 Hot Reload

Les fichiers sont surveillés avec `fsnotify`:

- Modification d'un hook → rechargement automatique
- Modification d'un module → rechargement de tous les scripts
- Nouveau fichier détecté → chargement automatique

📊 Monitoring

```bash
# Voir le statut des scripts
GET /api/scripts/status

# Recharger un script (admin)
POST /api/scripts/:scriptId/reload
```

📍 Système de Géolocalisation & Présence

Le serveur inclut un système complet de géolocalisation temps réel avec **WebRTC** et **REST API**.

🎯 Fonctionnalités

1. Localisation en Temps Réel

- ✅ Partage de position GPS (lat/lng, accuracy, altitude, speed, heading)
- ✅ Mise à jour temps réel via WebRTC DataChannel ou REST
- ✅ Historique des positions
- ✅ Calcul de distance (Haversine)

2. Présence Utilisateur

- ✅ États: `online`, `away`, `busy`, `offline`
- ✅ Détection automatique de présence (lastSeen)
- ✅ Notifications de changement de présence
- ✅ Statistiques de présence

3. Recherche Géographique

- ✅ **Nearby** - Utilisateurs dans un rayon (mètres)
- ✅ **Polygon** - Utilisateurs dans un polygone GeoJSON
- ✅ **Circle** - Utilisateurs dans un cercle
- ✅ Filtrage par présence

4. Geofencing

- ✅ Créer des zones géographiques (Point, Circle, Polygon)
- ✅ Détecter entrée/sortie de zone
- ✅ Actions automatiques: notification, chat, ads, call
- ✅ Métadonnées personnalisées

📡 API REST

bash

```
# Mettre à jour sa localisation
POST /api/location/update
Body: {
  "location": {
    "point": { "lat": 48.8566, "lng": 2.3522 },
    "accuracy": 10,
    "altitude": 35,
    "speed": 0,
    "heading": 0
  },
  "presence": "online"
}

# Récupérer la position d'un utilisateur
GET /api/location/user/:userId

# Trouver utilisateurs à proximité
POST /api/location/nearby
Body: {
  "point": { "lat": 48.8566, "lng": 2.3522 },
  "radius": 1000
}

# Trouver utilisateurs dans polygone
POST /api/location/polygon
Body: {
  "polygon": [
    { "lat": 48.8, "lng": 2.3 },
    { "lat": 48.9, "lng": 2.3 },
    { "lat": 48.9, "lng": 2.4 },
    { "lat": 48.8, "lng": 2.4 }
  ]
}

# Utilisateurs par présence
GET /api/location/presence?presence=online

# Créer geofence
POST /api/geofences
Body: {
  "name": "Zone commerciale",
  "geometry": {
    "type": "Circle",
    "coordinates": [2.3522, 48.8566],
    "radius": 500
  },
  "actions": ["notification", "ads"],
```

```
  "trigger_type": "enter",
  "metadata": {
    "notification_message": "Bienvenue!",
    "ad_data": { ... }
  }
}

# Notifier utilisateurs dans zone
POST /api/location/notify-zone
Body: {
  "point": { "lat": 48.8566, "lng": 2.3522 },
  "radius": 2000,
  "title": "Événement",
  "message": "Concert ce soir!"
}
```

🌐 WebRTC Integration

```
bash

# Broadcaster position dans room
POST /api/rooms/:roomId/broadcast-location
Body: {
  "location": {
    "point": { "lat": 48.8566, "lng": 2.3522 },
    "accuracy": 10
  }
}
```

🎯 Cas d'Usage

1. Chat de Proximité

Détecter automatiquement les utilisateurs proches et proposer un chat

2. Publicité Ciblée

Afficher des pubs quand l'utilisateur entre dans une zone commerciale

3. Alertes d'Urgence

Notifier tous les utilisateurs dans un rayon lors d'une urgence

4. Suivi en Temps Réel

Suivre la position d'utilisateurs dans une room WebRTC

5. Événements Locaux

Notifier les utilisateurs proches d'événements

📦 API TypeScript (Hooks)

```javascript
// Dans vos scripts hooks
const nearby = location.findNearby(lat, lng, radius);
const onlineUsers = location.getUsersByPresence("online");
const distance = location.distance(lat1, lng1, lat2, lng2);
location.updateLocation(userId, lat, lng, accuracy, "online");
```

📊 Collections

- **users** - `location` (JSON), `presence`, `lastSeen`
- **geofences** - Zones géographiques configurables
- **locationHistory** - Historique des positions (optionnel)

🔔 Événements Pub/Sub

- `location_updates` - Mise à jour position
- `geo_events` - Événements geofence (enter/exit)
- `presence_changes` - Changement présence

🔗 Canaux Dédiés Utilisateur

Chaque utilisateur possède **2 canaux de communication dédiés** :

1. 📡 SSE Channel (Half-Duplex)

Canal Server-Sent Events pour recevoir :

- ✅ Réponses aux requêtes REST (si `respond_to=sse`)
- ✅ Notifications push
- ✅ Événements temps réel
- ✅ Mises à jour de localisation
- ✅ Événements sociaux (likes, comments)

```javascript
// Connexion SSE
GET /api/user/sse
Authorization: Bearer TOKEN

// Recevoir les événements
{
  "type": "notification",
  "request_id": "req_123",
  "data": { ... },
  "timestamp": 1234567890
}
```

2. 🎯 User Room WebRTC (Full-Duplex)

Room WebRTC dédiée avec DataChannel pour :

- ✅ **Envoyer** des requêtes API (comme REST)
- ✅ **Recevoir** des réponses instantanées
- ✅ Gérer la présence automatiquement
- ✅ Notifications bidirectionnelles
- ✅ Latence ultra-faible

```javascript
// Connexion à la room utilisateur
POST /api/user/room/connect
→ Retourne SDP offer

POST /api/user/room/answer
Body: { type: "answer", sdp: "..." }
```

📨 Requêtes API via WebRTC

Toutes les REST API sont disponibles via DataChannel !

Format de requête (MsgPack) :

```javascript
{
  request_id: "req_123",
  method: "POST",
  endpoint: "/posts",
  body: {
    content: "Hello",
    type: "html",
    isPublic: true
  },
  query: null
}
```

Format de réponse :

```javascript
{
  request_id: "req_123",
  status_code: 200,
  data: {
    success: true,
    post_id: "abc123",
    post: { ... }
  },
  error: null,
  timestamp: 1234567890
}
```

🎮 Endpoints Disponibles via WebRTC

Location

- `POST /location/update` - Mettre à jour position
- `POST /location/nearby` - Trouver utilisateurs proches
- `POST /location/polygon` - Utilisateurs dans polygone

Social

- `GET /posts` - Récupérer posts
- `POST /posts` - Créer post
- `POST /posts/like` - Liker post
- `POST /posts/comment` - Commenter

Marketplace

- `GET /articles` - Récupérer articles
- `POST /articles/buy` - Acheter article

Présence

- `POST /presence/update` - Changer présence

Rooms

- `POST /rooms/join` - Rejoindre room
- `POST /rooms/leave` - Quitter room

🔄 REST avec respond_to=sse

Les requêtes REST peuvent retourner la réponse via SSE :

```bash
POST /api/rooms?respond_to=sse
X-Request-ID: req_123
Authorization: Bearer TOKEN

# Réponse immédiate:
{ "status": "response_sent_via_sse" }

# Réponse détaillée arrive via SSE:
data: {
  "type": "room_created",
  "request_id": "req_123",
  "data": {
    "room_id": "room_abc",
    "room_type": "audio"
  }
}
```

📊 Client JavaScript

```javascript
const client = new UserChannelsClient(serverUrl, authToken);

// Connexion SSE
client.connectSSE();
client.onSSE('notification', (data) => {
  console.log('Notification:', data);
});

// Connexion User Room
await client.connectUserRoom();

// Utiliser l'API via WebRTC
const post = await client.createPost('Hello!');
const nearby = await client.findNearby(48.8566, 2.3522, 1000);
await client.likePost(postId, 'fire');
await client.updatePresence('away');
```

🎯 Avantages

SSE (Half-Duplex)

- ✅ Simple à implémenter
- ✅ Fonctionne partout (HTTP/HTTPS)
- ✅ Reconnexion automatique
- ✅ Parfait pour notifications

User Room WebRTC (Full-Duplex)

- ✅ **Latence ultra-faible** (<50ms)
- ✅ Communication bidirectionnelle
- ✅ Pas de polling
- ✅ Détection présence automatique
- ✅ Fonctionne hors-ligne (queue locale)
- ✅ **Toutes les API REST disponibles**

📡 Notifications Automatiques

Le système envoie automatiquement via **SSE** + **User Room** :

javascript

```javascript
// Quelqu'un like votre post → notification via SSE + Room
{
  "type": "post_liked",
  "data": {
    "post_id": "abc",
    "user_id": "xyz",
    "reaction": "fire"
  }
}

// Nouveau commentaire → notification via SSE + Room
{
  "type": "post_commented",
  "data": {
    "post_id": "abc",
    "user_id": "xyz",
    "content": "Great post!"
  }
}

// Événement géofence → notification
{
  "type": "geo_event",
  "data": {
    "type": "user_entered",
    "fence_id": "fence_123"
  }
}
```

🔌 Gestion Présence Automatique

La connexion à la User Room met automatiquement `presence = online` et la déconnexion met `offline`.

```javascript
// Détection automatique d'activité
client.connectUserRoom(); // → presence = online
client.disconnectUserRoom(); // → presence = offline

// Tab cachée
document.addEventListener('visibilitychange', () => {
  if (document.hidden) {
    client.updatePresence('away');
  } else {
    client.updatePresence('online');
  }
});
```

👥 Système Follow/Followers

Système complet de follow avec **4 modes** et gestion des **admins**.

🎯 Types de Follow

1. Gratuit (`free`)

```javascript
{
  follow_type: 'free',
  is_accepting_followers: true
}
// → Follow instantané, status = 'active'
```

2. Avec Approbation (`require_approval`)

```javascript
{
  follow_type: 'require_approval',
  is_accepting_followers: true
}
// → Status = 'pending', nécessite approbation
```

3. Payant Périodique (`paid_period`)

```javascript
{
  follow_type: 'paid_period',
  price: 9.99,
  period_days: 30
}
// → Abonnement mensuel, expire après 30 jours
```

4. Payant à Vie (`paid_lifetime`)

```javascript
{
  follow_type: 'paid_lifetime',
  price: 99.99
}
// → Paiement unique, accès permanent
```

📋 Roles des Followers

- **Follower** (`follower`) - Suiveur standard
- **Admin** (`admin`) - Peut approuver/rejeter des follows, modérer

🔧 API Follow

```bash
# Configurer ses paramètres
PUT /api/user/follow-settings
Body: { follow_type, price, period_days, description }

# Follow un utilisateur
POST /api/users/:userId/follow

# Unfollow
DELETE /api/users/:userId/follow

# Approuver/Rejeter demande
POST /api/follows/:followId/approve
POST /api/follows/:followId/reject

# Promouvoir en admin
POST /api/follows/:followId/promote

# Liste followers/following
GET /api/users/:userId/followers?status=active
GET /api/users/:userId/following
```

🏠 Gestion des Rooms

Système de rooms avec **owner**, **admins**, **participants** et **4 modes d'accès**.

🎯 Types d'Accès Room

1. Gratuit (`free`)

```javascript
{
  join_type: 'free',
  max_participants: 50
}
// → Accès instantané
```

2. Avec Approbation (`require_approval`)

```javascript
{
  join_type: 'require_approval'
}
// → Owner/admins doivent approuver
```

3. Payant Périodique (`paid_period`)

```javascript
{
  join_type: 'paid_period',
  price: 19.99,
  period_days: 30
}
// → Abonnement mensuel
```

4. Payant à Vie (`paid_lifetime`)

```javascript
{
  join_type: 'paid_lifetime',
  price: 299.99
}
// → Paiement unique
```

👑 Hiérarchie des Rôles

Owner (Propriétaire)

- ✅ Tous les droits
- ✅ Promouvoir/rétrograder admins
- ✅ Transférer propriété
- ✅ Modifier paramètres room
- ✅ Bannir membres
- ❌ Ne peut pas quitter (doit transférer)

Admin (Administrateur)

- ✅ Approuver/rejeter membres
- ✅ Bannir membres
- ✅ Modérer la room
- ❌ Ne peut pas promouvoir d'autres admins
- ❌ Ne peut pas modifier paramètres

Participant (Membre)

- ✅ Participer à la room
- ✅ Quitter librement
- ❌ Pas de droits de modération

🔧 **API Rooms**

```bash
# Créer room
POST /api/rooms/create
Body: {
  room_type, name, description, is_public,
  max_participants, join_type, price, period_days
}

# Rejoindre
POST /api/rooms/:roomId/join-request

# Gérer membres
POST /api/room-members/:memberId/approve
POST /api/room-members/:memberId/reject
POST /api/room-members/:memberId/ban

# Gérer rôles
POST /api/room-members/:memberId/promote  # → admin
POST /api/room-members/:memberId/demote   # → participant

# Transférer propriété
POST /api/rooms/:roomId/transfer-ownership
Body: { new_owner_id }

# Modifier paramètres (owner)
PATCH /api/rooms/:roomId/settings
Body: { name, max_participants, price, ... }

# Quitter
POST /api/rooms/:roomId/leave

# Lister
GET /api/rooms/:roomId/members?status=active
GET /api/user/rooms
```

📊 Collections

followSettings

- `user`, `followType`, `price`, `periodDays`
- `description`, `isAcceptingFollowers`

follows

- `follower`, `following`, `status`, `role`
- `expiresAt`, `paidAmount`, `approvedBy`

rooms (étendue)

- `owner`, `joinType`, `price`, `periodDays`
- `maxParticipants`, `isActive`, `isPublic`

roomMembers

- `room`, `user`, `role`, `status`
- `expiresAt`, `paidAmount`, `joinedAt`
- `approvedBy`, `permissions`

⏰ Gestion Expiration

Tâche automatique toutes les heures :

- Expire les follows périodiques
- Expire les accès rooms périodiques
- Notifie les utilisateurs via SSE + User Room

🔔 Notifications Automatiques

```javascript
// Demande de follow
{ type: "follow_request", follower_id: "..." }

// Follow approuvé
{ type: "follow_approved", following_id: "..." }

// Promu admin
{ type: "promoted_to_admin", following_id: "..." }

// Demande rejoindre room
{ type: "join_request", user_id: "...", room_id: "..." }

// Membre approuvé
{ type: "room_join_approved", room_id: "..." }

// Promu admin room
{ type: "promoted_to_room_admin", room_id: "..." }

// Abonnement expiré
{ type: "follow_expired", following_id: "..." }
{ type: "room_membership_expired", room_id: "..." }
```

🆘 Support

Pour toute question, ouvrir une issue sur GitHub.

Happy coding! 🚀