📘 Documentation API TypeScript/JavaScript

Toutes les APIs disponibles dans l'interpréteur Goja pour exécuter du TypeScript/JavaScript côté serveur.

🚀 Utilisation

Exécuter du code inline

bash

```
POST /api/scripts/execute
Authorization: Bearer YOUR_TOKEN

{
  "code": "const result = db.findAll('posts', 'isPublic=true', '-created', 10); return result;"
}
```

Exécuter un script stocké

bash

```
POST /api/scripts/:scriptId/run
Authorization: Bearer YOUR_TOKEN
```

Exécuter une fonction spécifique

bash

```
POST /api/scripts/function
Authorization: Bearer YOUR_TOKEN

{
  "script_id": "script_abc",
  "function": "calculateEngagementScore",
  "args": ["post_123"]
}
```

📚 APIs Disponibles

1. db - Database Operations

db.findById(collection, id)

Récupère un record par son ID.

```typescript
const post = db.findById("posts", "post_123");
console.log(post.id, post.content, post.likesCount);
```

db.findOne(collection, filter)

Récupère le premier record correspondant au filtre.

```typescript
const user = db.findOne("users", "email = 'user@example.com'");
```

db.findAll(collection, filter?, sort?, limit?)

Récupère tous les records correspondant aux critères.

```typescript
// Tous les posts publics
const posts = db.findAll("posts", "isPublic = true", "-created", 20);

// Tous les articles d'un utilisateur
const articles = db.findAll("articles", "user = 'user_123'");
```

db.create(collection, data)

Crée un nouveau record.

```typescript
const post = db.create("posts", {
  user: "user_123",
  type: "html",
  content: "Mon nouveau post!",
  isPublic: true,
  likesCount: 0,
  commentsCount: 0
});

console.log("Created post:", post.id);
```

db.update(collection, id, data)

Met à jour un record existant.

```typescript
db.update("posts", "post_123", {
  likesCount: 42,
  content: "Contenu mis à jour"
});
```

db.delete(collection, id)

Supprime un record.

```typescript
db.delete("posts", "post_123");
```

db.count(collection, filter)

Compte les records correspondant au filtre.

```typescript
const totalPosts = db.count("posts", "isPublic = true");
console.log("Total public posts:", totalPosts);
```

db.query(sql, params...)

Exécute une requête SQL brute.

```typescript
const results = db.query(`
  SELECT user, COUNT(*) as total
  FROM posts
  WHERE isPublic = true
  GROUP BY user
`);
```

---

2. webrtc - WebRTC Operations

webrtc.getRoom(roomId)

Récupère les informations d'une room.

```typescript
const room = webrtc.getRoom("room_123");
console.log(`Room ${room.id} (${room.type}) has ${room.count} participants`);
```

webrtc.listRooms()

Liste toutes les rooms actives.

```typescript
const rooms = webrtc.listRooms();
for (const room of rooms) {
  console.log(`${room.id}: ${room.participants} participants`);
}
```

webrtc.broadcast(roomId, eventType, data)

Envoie un événement à tous les participants d'une room.

```typescript
webrtc.broadcast("room_123", "announcement", {
  message: "Server maintenance in 5 minutes",
  priority: "high"
});
```

webrtc.kickParticipant(roomId, participantId)

Expulse un participant d'une room.

```typescript
webrtc.kickParticipant("room_123", "participant_456");
```

---

3. pubsub - Publish/Subscribe

pubsub.publish(topic, payload)

Publie un message sur un topic.

```typescript
pubsub.publish("post_events", {
  type: "new_post",
  post_id: "post_123",
  user_id: "user_456"
});

pubsub.publish("notifications", {
  type: "achievement",
  user_id: "user_123",
  achievement: "100_posts"
});
```

pubsub.subscribe(topic, callback)

S'abonne à un topic (callback asynchrone).

```typescript
pubsub.subscribe("post_events", function(data) {
  console.log("New event:", data.type);

  if (data.type === "new_post") {
    // Traiter le nouveau post
    moderateNewPost(data.post_id);
  }
});
```

Topics disponibles:

- `post_events` - Événements liés aux posts
- `sales` - Événements de vente
- `reactions` - Réactions dans les rooms
- `notifications` - Notifications générales
- `admin_notifications` - Notifications admin

---

4. social - Social Media Operations

social.likePost(userId, postId, reaction)

Ajoute un like/réaction à un post.

```typescript
social.likePost("user_123", "post_456", "fire");

// Réactions disponibles: like, love, fire, wow, sad, angry
```

social.commentPost(userId, postId, content, parentComment?)

Ajoute un commentaire à un post.

```typescript
// Commentaire simple
social.commentPost("user_123", "post_456", "Super post! 🔥");

// Réponse à un commentaire
social.commentPost("user_123", "post_456", "Merci!", "comment_789");
```

social.getPostStats(postId)

Récupère les statistiques d'un post.

```typescript
const stats = social.getPostStats("post_123");
console.log(`Likes: ${stats.likesCount}, Comments: ${stats.commentsCount}`);
```

social.getTrendingPosts(limit)

Récupère les posts tendances.

```typescript
const trending = social.getTrendingPosts(10);
for (const post of trending) {
  console.log(`${post.id}: ${post.likesCount} likes`);
}
```

---

5. marketplace - Marketplace Operations

marketplace.createSale(userId, articleId, amount)

Crée une vente (achat d'article).

```typescript
const sale = marketplace.createSale("user_123", "article_456", 99.99);
console.log("Sale created:", sale.vente_id);
```

marketplace.getSalesStats(userId)

Récupère les statistiques de vente d'un vendeur.

```typescript
const stats = marketplace.getSalesStats("user_123");
console.log(`Total sales: ${stats.total_sales}`);
console.log(`Paid sales: ${stats.paid_sales}`);
console.log(`Pending: ${stats.pending_sales}`);
console.log(`Number of sales: ${stats.count}`);
```

marketplace.getWalletBalance(userId)

Récupère la balance du wallet d'un utilisateur.

```typescript
const balance = marketplace.getWalletBalance("user_123");
console.log(`Balance: ${balance} coins`);
```

---

6. utils - Utility Functions

utils.jsonEncode(data)

Encode un objet en JSON.

```typescript
const json = utils.jsonEncode({ name: "John", age: 30 });
console.log(json); // '{"name":"John","age":30}'
```

utils.jsonDecode(str)

Décode une chaîne JSON.

```typescript
const obj = utils.jsonDecode('{"name":"John"}');
console.log(obj.name); // "John"
```

utils.msgpackEncode(data)

Encode en MsgPack (format binaire).

```typescript
const binary = utils.msgpackEncode({ message: "Hello" });
```

utils.msgpackDecode(data)

Décode du MsgPack.

```typescript
const obj = utils.msgpackDecode(binaryData);
```

utils.generateId()

Génère un ID unique.

```typescript
const id = utils.generateId();
```

utils.hashPassword(password)

Hash un mot de passe.

```typescript
const hashed = utils.hashPassword("mypassword");
```

utils.random(min, max)

Génère un nombre aléatoire.

```typescript
const randomNum = utils.random(1, 100);
```

utils.uuid()

Génère un UUID.

```typescript
const uuid = utils.uuid();
```

---

7. storage - In-Memory Storage

storage.set(key, value)

Stocke une valeur en mémoire.

```typescript
storage.set("user_sessions", { user_123: "active" });
storage.set("cache_posts", posts);
```

storage.get(key)

Récupère une valeur.

```typescript
const sessions = storage.get("user_sessions");
```

storage.delete(key)

Supprime une clé.

```typescript
storage.delete("cache_posts");
```

storage.keys()

Liste toutes les clés.

```typescript
const allKeys = storage.keys();
console.log("Stored keys:", allKeys);
```

---

8. cron - Scheduled Tasks

cron.schedule(interval, callback)

Planifie une tâche récurrente (interval en secondes).

```typescript
// Exécuter toutes les heures (3600 secondes)
cron.schedule(3600, function() {
  log("Running hourly task");
  checkLowStock();
});

// Exécuter toutes les 5 minutes
cron.schedule(300, function() {
  log("5-minute task");
});
```

cron.setTimeout(delay, callback)

Exécute une fonction après un délai (delay en millisecondes).

```typescript
cron.setTimeout(5000, function() {
  log("Executed after 5 seconds");
});
```

---

9. auth - Authentication

auth.verifyToken(token)

Vérifie un token JWT.

```typescript
const result = auth.verifyToken("eyJhbGc...");
if (result.valid) {
  console.log("User ID:", result.userId);
}
```

auth.getUser(userId)

Récupère un utilisateur par ID.

```typescript
const user = auth.getUser("user_123");
console.log(user.email, user.name);
```

---

10. http - HTTP Client (Simplifié)

http.get(url)

Effectue une requête GET.

```typescript
const response = http.get("https://api.example.com/data");
console.log(response.status, response.body);
```

http.post(url, data)

Effectue une requête POST.

```typescript
const response = http.post("https://api.example.com/webhook", {
  event: "test",
  data: { foo: "bar" }
});
```

---

11. Global Functions

log(...args)

Affiche dans les logs du serveur.

```typescript
log("Debug message");
log("User", userId, "performed action");
```

sleep(ms)

Met en pause l'exécution (en millisecondes).

```typescript
log("Starting...");
sleep(1000);
log("1 second later");
```

timestamp()

Retourne le timestamp Unix actuel.

```typescript
const now = timestamp();
console.log("Current time:", now);
```

🎯 Cas d'Usage Complets

1. Modération automatique des posts

typescript

```typescript
function moderatePost(postId: string) {
  const post = db.findById("posts", postId);
  const content = post.content.toLowerCase();

  const badWords = ["spam", "scam"];
  for (const word of badWords) {
    if (content.includes(word)) {
      db.update("posts", postId, { isPublic: false });
      pubsub.publish("admin_notifications", {
        type: "moderation",
        post_id: postId,
        reason: "Suspicious content detected"
      });
      return { moderated: true };
    }
  }

  return { moderated: false };
}
```

2. Calculer l'engagement d'un post

typescript

```typescript
function calculateEngagement(postId: string) {
  const post = db.findById("posts", postId);

  const score = (post.likesCount * 1) + (post.commentsCount * 3);
  const age = timestamp() - new Date(post.created).getTime() / 1000;
  const scorePerHour = score / (age / 3600);

  const isTrending = scorePerHour > 10;

  db.update("posts", postId, {
    dataAction: utils.jsonEncode({
      engagementScore: score,
      isTrending: isTrending
    })
  });

  return { score, isTrending };
}
```

3. Système de récompenses

typescript

```typescript
function rewardTopUsers() {
  const posts = db.findAll("posts", "", "-likesCount", 100);
  const userScores = {};

  for (const post of posts) {
    userScores[post.user] = (userScores[post.user] || 0) + post.likesCount;
  }

  for (const userId in userScores) {
    const score = userScores[userId];
    if (score > 100) {
      db.create("operations", {
        user: userId,
        montant: score * 0.1,
        operation: "cashin",
        desc: "Récompense d'engagement",
        status: "paye"
      });
    }
  }
}
```

4. Bot pour room audio

typescript

```typescript
function createRoomBot(roomId: string) {
  webrtc.broadcast(roomId, "chat", {
    from: "Bot",
    message: "🤖 Bot joined the room!"
  });

  // S'abonner aux événements de la room
  pubsub.subscribe("room_events", function(event) {
    if (event.room_id === roomId && event.type === "message") {
      const msg = event.data.message;

      if (msg.startsWith("/help")) {
        webrtc.broadcast(roomId, "chat", {
          from: "Bot",
          message: "Commands: /stats, /kick, /mute"
        });
      }
    }
  });
}
```

5. Analytics Dashboard

```typescript
function getDashboardStats() {
  const totalPosts = db.count("posts", "");
  const publicPosts = db.count("posts", "isPublic = true");
  const totalUsers = db.count("users", "");

  const recentPosts = db.findAll("posts", "", "-created", 10);
  let totalEngagement = 0;

  for (const post of recentPosts) {
    totalEngagement += post.likesCount + post.commentsCount;
  }

  const rooms = webrtc.listRooms();
  const activeParticipants = rooms.reduce((sum, r) => sum + r.participants, 0);

  return {
    posts: { total: totalPosts, public: publicPosts },
    users: totalUsers,
    engagement: { total: totalEngagement, avg: totalEngagement / 10 },
    realtime: { rooms: rooms.length, participants: activeParticipants }
  };
}
```

🔒 Sécurité

Permissions

- Seuls les utilisateurs authentifiés peuvent exécuter des scripts
- Les scripts ont accès à toutes les collections (utiliser avec précaution)
- Pas d'accès au système de fichiers
- Pas d'accès réseau complet (HTTP limité)

Bonnes pratiques

```typescript
// ✅ BON: Vérifier les erreurs
const post = db.findById("posts", postId);
if (post.error) {
  log("Error:", post.error);
  return { error: post.error };
}

// ✅ BON: Valider les données
if (!userId || !postId) {
  return { error: "Missing parameters" };
}

// ❌ MAUVAIS: Pas de validation
db.update("posts", postId, data); // data pourrait être malicieux
```

📊 Performance

- Les scripts sont exécutés de manière synchrone
- Utiliser `cron.schedule()` pour les tâches longues
- Le storage est en mémoire (données perdues au redémarrage)
- Limiter les boucles et requêtes DB massives

🚀 Déploiement

Créer un script dans PocketBase

```bash
POST /api/collections/scripts/records
Authorization: Bearer TOKEN

{
  "name": "Moderate Posts",
  "code": "function main() { /* code */ }",
  "description": "Auto-moderation script",
  "enabled": true,
  "user": "user_id"
}
```

Exécuter via API

bash

```
POST /api/scripts/:scriptId/run
Authorization: Bearer TOKEN
```

Tous les objets sont exposés globalement dans l'interpréteur TypeScript ! 🎉