readme.md for @push.rocks/smartsocket

Easy and secure WebSocket communication with native WebSocket support □□

Features

- □□ **Native WebSocket** - Uses native WebSocket API (browser) and `ws` library (Node.js)
- □□ **Auto Reconnection** - Exponential backoff with configurable retry limits
- □□ **RPC-style Function Calls** - Define and call functions across server/client
- □□ **Connection Tagging** - Tag connections for easy identification and routing
- □□ **Smartserve Integration** - Works seamlessly with `@push.rocks/smartserve`
- □□ **Secure Communication** - WSS support for encrypted connections

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

```
npm install @push.rocks/smartsocket --save
```

or with pnpm:

```
pnpm add @push.rocks/smartsocket
```

Usage

Quick Start - Server

```
import { Smartsocket, SocketFunction } from '@push.rocks/smartsocket';

// Create server
const server = new Smartsocket({
  alias: 'myServer',
  port: 3000
});

// Define a function that clients can call
const greetFunction = new SocketFunction({
  funcName: 'greet',
  funcDef: async (data, socketConnection) => {
    console.log(`Received greeting from ${data.name}`);
    return { message: `Hello, ${data.name}!` };
  }
});

server.addSocketFunction(greetFunction);

// Start the server
await server.start();
console.log('WebSocket server running on port 3000');
```

Quick Start - Client

```
import { SmartsocketClient } from '@push.rocks/smartsocket';

// Create client
const client = new SmartsocketClient({
  url: 'http://localhost',
  port: 3000,
  alias: 'myClient',
```

```
  autoReconnect: true
});

// Connect to server
await client.connect();

// Call server function
const response = await client.serverCall('greet', { name: 'Alice' });
console.log(response.message); // "Hello, Alice!"
```

Connection Options

The `SmartsocketClient` supports several configuration options:

```
const client = new SmartsocketClient({
  url: 'http://localhost',           // Server URL (http/https)
  port: 3000,                         // Server port
  alias: 'myClient',                  // Client identifier
  autoReconnect: true,                // Auto-reconnect on disconnect
  maxRetries: 100,                    // Max reconnection attempts (default: 100)
  initialBackoffDelay: 1000,          // Initial backoff in ms (default: 1000)
  maxBackoffDelay: 60000              // Max backoff in ms (default: 60000)
});
```

Two-Way Function Calls

Both server and client can define and call functions on each other:

```
// Server calling client
const clientFunction = new SocketFunction({
  funcName: 'clientTask',
  funcDef: async (data) => {
    return { result: 'Task completed' };
  }
});

// On client
client.addSocketFunction(clientFunction);
```

```
// On server - call the client
const socketConnection = server.socketConnections.findSync(conn => conn.alias === 'myClient');
const result = await server.clientCall('clientTask', { task: 'doSomething' },
socketConnection);
```

Connection Tagging

Tag connections to identify and group them:

```
// On client
await client.addTag({
  id: 'role',
  payload: 'admin'
});

// On server - find tagged connections
const adminConnections = server.socketConnections.getArray().filter(async conn => {
  const tag = await conn.getTagById('role');
  return tag?.payload === 'admin';
});

// Server can also tag connections
await socketConnection.addTag({
  id: 'verified',
  payload: true
});
```

Integration with Smartserve

Use smartsocket with `@push.rocks/smartserve` for advanced HTTP/WebSocket handling:

```
import { SmartServe } from '@push.rocks/smartserve';
import { Smartsocket } from '@push.rocks/smartsocket';

// Create smartsocket without a port (hooks mode)
const smartsocket = new Smartsocket({ alias: 'myServer' });
```

```
// Get WebSocket hooks and pass them to SmartServe
const wsHooks = smartsocket.getSmartserveWebSocketHooks();
const smartserve = new SmartServe({
  port: 3000,
  websocket: wsHooks
});

// Add socket functions as usual
smartsocket.addSocketFunction(myFunction);

// Start smartserve (smartsocket hooks mode doesn't need start())
await smartserve.start();
```

Handling Disconnections

The client automatically handles reconnection with exponential backoff:

```
const client = new SmartsocketClient({
  url: 'http://localhost',
  port: 3000,
  alias: 'myClient',
  autoReconnect: true,
  maxRetries: 10,
  initialBackoffDelay: 500,
  maxBackoffDelay: 5000
});

// Listen for connection status changes
client.eventSubject.subscribe(status => {
  console.log('Connection status:', status);
  // Status can be: 'new', 'connecting', 'connected', 'disconnecting', 'timedOut'
});

await client.connect();

// Manually disconnect without auto-reconnect
await client.disconnect();

// Stop the client completely (disables auto-reconnect)
```

```
await client.stop();
```

Secure Connections (WSS)

For secure WebSocket connections, use HTTPS URLs:

```
const client = new SmartsocketClient({
  url: 'https://secure.example.com',  // HTTPS triggers WSS
  port: 443,
  alias: 'secureClient'
});
```

TypedRequest Integration

For strongly-typed RPC calls, define interfaces:

```
interface IGreetRequest {
  method: 'greet';
  request: { name: string };
  response: { message: string };
}

// Type-safe server call
const response = await client.serverCall<IGreetRequest>('greet', { name: 'Bob' });
// response is typed as { message: string }
```

API Reference

Smartsocket (Server)

Method	Description
`start()`	Start the WebSocket server (not needed in hooks mode)
`stop()`	Stop the server and close all connections
`addSocketFunction(fn)`	Register a function that clients can call

Method	Description
`clientCall(funcName, data, connection)`	Call a function on a specific client
`getSmartserveWebSocketHooks()`	Get hooks for smartserve integration

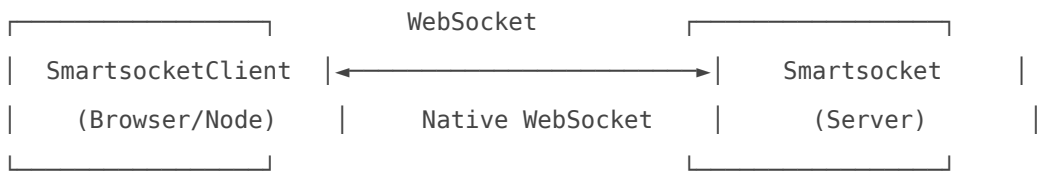
SmartsocketClient

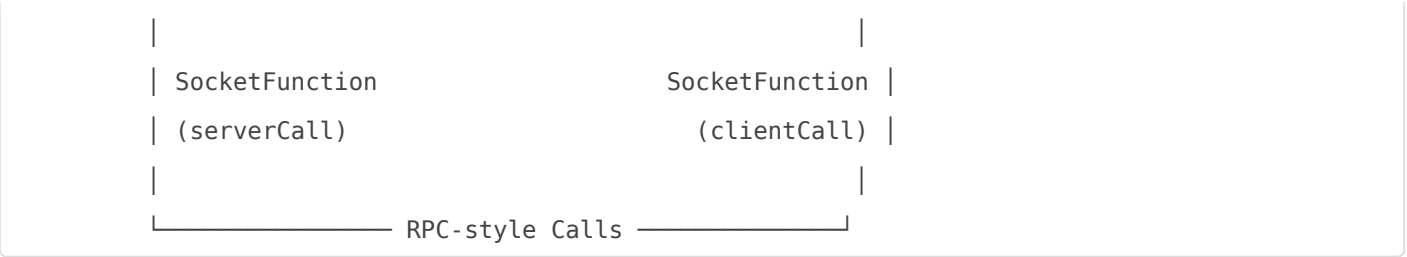
Method	Description
`connect()`	Connect to the server
`disconnect()`	Disconnect from the server
`stop()`	Disconnect and disable auto-reconnect
`serverCall(funcName, data)`	Call a function on the server
`addSocketFunction(fn)`	Register a function the server can call
`addTag(tag)`	Add a tag to the connection
`getTagById(id)`	Get a tag by its ID
`removeTagById(id)`	Remove a tag by its ID

SocketFunction

```
const fn = new SocketFunction({
  funcName: 'myFunction',
  funcDef: async (data, socketConnection) => {
    // data: the request payload
    // socketConnection: the calling connection
    return { result: 'response' };
  }
});
```

Architecture





License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:12:33 UTC by foss.global Team
Updated 2026-03-28 12:19:19 UTC by foss.global Team