Testing Guide

This guide helps you test and validate the WebSocket implementation.

🧪 Test Scenarios

1. Basic Connection Test ✅

Objective: Verify handshake and connection establishment

Steps:

1. Start the server: `node server.mjs`
2. Open `index.html` in a browser
3. Expected result:
    - Server logs: `🔄 Upgrading to WebSocket connection`
    - Client shows: `✅ Connected to server`

Success Criteria:

- ✅ Server accepts WebSocket upgrade
- ✅ Client connection status shows "Connected"
- ✅ No errors in browser console
- ✅ No errors in server logs

---

2. Message Echo Test 📨

Objective: Test bidirectional communication

Steps:

1. With connection established, type "Hello WebSocket" in input field
2. Click Send or press Enter
3. Expected result:
    - Client displays: `📤 Sent: Hello WebSocket`
    - Server logs: `📩 Received: "Hello WebSocket"`
    - Server logs: `📤 Sent: "Echo: Hello WebSocket"`
    - Client displays: `📨 Message from server: Echo: Hello WebSocket`

Success Criteria:

- ✅ Message sent successfully
- ✅ Server receives and logs message
- ✅ Server sends echo response
- ✅ Client receives echo response
- ✅ All messages timestamped

---

3. Short Message Test (< 126 bytes) 🔤

Objective: Test 7-bit length encoding

Test Messages:

```
"Hi"
"Testing 123"
"The quick brown fox jumps over the lazy dog"
"A" (single character)
"😀🎉🚀" (emojis)
```

Expected Frame Structure:

```
Client→Server: [0x81][0x80 + length][4-byte mask][masked payload]
Server→Client: [0x81][length][payload]
```

Success Criteria:

- ✅ All messages echoed correctly
- ✅ Payload length fits in 7 bits
- ✅ UTF-8 characters handled properly

---

4. Medium Message Test (126-65535 bytes) 📦

Objective: Test 16-bit length encoding

Test Messages:

```
// 200 bytes
"A".repeat(200);

// 1000 bytes
"Lorem ipsum... ".repeat(50);

// 5000 bytes
"x".repeat(5000);
```

Expected Frame Structure:

```
Client→Server: [0x81][0xFE (126)][2-byte length][4-byte mask][payload]
Server→Client: [0x81][0x7E (126)][2-byte length][payload]
```

Success Criteria:

- ✅ Messages > 125 bytes use 16-bit length
- ✅ All bytes received correctly
- ✅ Echo matches original message

---

5. Large Message Test (> 65535 bytes) 🗄️

Objective: Test 64-bit length encoding

Test Message:

```
// 100KB message
"x".repeat(100000);
```

Expected Frame Structure:

```
[0x81][0xFF (127)][8-byte length][4-byte mask][payload]
```

Success Criteria:

- ✅ Large messages handled correctly
- ✅ No buffer overflow
- ✅ Complete message received

---

6. Special Characters Test 🌍

Objective: Test UTF-8 encoding/decoding

Test Messages:

```
"Hello 世界"
"Emoji time! 😀🎉🚀🌟💻"
"Symbols: ©®™€£¥"
"Math: ∑∫∂√π∞"
"Arrows: ←↑→↓"
```

Success Criteria:

- ✅ All UTF-8 characters preserved
- ✅ Multi-byte characters handled correctly
- ✅ No encoding errors

---

7. Rapid Fire Test ⚡

Objective: Test high-frequency messages

Steps:

1. Send 10 messages quickly
2. Expected: All messages echoed in order

Test Script:

```
// Run in browser console
for (let i = 0; i < 10; i++) {
  socket.send(`Message ${i}`);
}
```

Success Criteria:

- ✅ All messages received
- ✅ Messages in correct order
- ✅ No messages lost
- ✅ No server errors

---

8. Empty Message Test 🚫

Objective: Test edge case handling

Test Messages:

```
""
" "
"    "
```

Expected Behavior:

- Client should prevent sending empty messages
- UI validation should work

Success Criteria:

- ✅ Empty messages not sent
- ✅ Input validation works
- ✅ No server errors

---

9. Connection Close Test 🔌

Objective: Test graceful disconnection

Steps:

1. Close browser tab or window

2. Expected server log: 👋 `Client disconnected`

Alternative:

```
// Run in browser console
socket.close(1000, "Normal closure");
```

Success Criteria:

- ✅ Connection closes gracefully
- ✅ Server detects disconnection
- ✅ No hanging connections
- ✅ Resources cleaned up

---

10. Reconnection Test 🔄

Objective: Test connection recovery

Steps:

1. Close connection
2. Refresh browser page
3. Expected: New connection established

Success Criteria:

- ✅ New connection successful
- ✅ Messages work after reconnect
- ✅ No stale connections

🛠️ Testing Tools

Using wscat (Command Line)

```
# Install
npm install -g wscat

# Connect
wscat -c ws://localhost:1338

# Send messages
> Hello from wscat!
< Echo: Hello from wscat!

# Close connection
Ctrl+C
```

Using websocat (Rust)

```
# Install (macOS)
brew install websocat

# Install (Linux)
cargo install websocat

# Connect
websocat ws://localhost:1338
```

Using Browser DevTools 🔍

1. Open DevTools (F12)
2. Network tab
3. Filter: WS
4. Click on WebSocket connection
5. View:
    - Messages tab: See all frames
    - Headers tab: See handshake
    - Timing tab: See connection duration

📊 Performance Testing

Latency Test

```
// Run in browser console
const start = performance.now();
socket.send("ping");

socket.addEventListener("message", function handler(e) {
  if (e.data.includes("ping")) {
    const latency = performance.now() - start;
    console.log(`Round-trip time: ${latency.toFixed(2)}ms`);
    socket.removeEventListener("message", handler);
  }
});
```

Throughput Test

```
# Send 1000 messages
for i in {1..1000}; do echo "Message $i"; done | websocat
ws://localhost:1338
```

🐛 Debugging Checklist

Server Issues

- [ ] Check if port 1338 is available
- [ ] Verify Node.js version (>= 14)
- [ ] Check server logs for errors
- [ ] Ensure proper file permissions
- [ ] Verify network firewall settings

```
# Check port availability
lsof -i :1338
netstat -tuln | grep 1338

# Test basic HTTP
curl http://localhost:1338
```

Client Issues

- [ ] Check browser console for errors
- [ ] Verify WebSocket URL is correct
- [ ] Check browser WebSocket support
- [ ] Clear browser cache
- [ ] Try different browser

```
// Check WebSocket support
if ("WebSocket" in window) {
  console.log("✅ WebSocket is supported");
} else {
  console.log("❌ WebSocket is not supported");
}
```

Connection Issues

- [ ] Server is running
- [ ] Firewall not blocking
- [ ] Correct protocol (ws:// not http://)
- [ ] Port number matches
- [ ] No proxy interference

📈 Expected Logs

Successful Connection Sequence

Server Console:

```
🚀 WebSocket Server is listening on port 1338
📡 Connect via: ws://localhost:1338

🔄 Upgrading to WebSocket connection
🔑 Client Key: dGhlIHNhbXBsZSBub25jZQ==
✅ Handshake complete, WebSocket connection established

📩 Received: "Hello from WebSocket client!" (29 bytes, opcode: 1)
📤 Sent: "Echo: Hello from WebSocket client!" (34 bytes)
```

Browser Console:

```
WebSocket connection established
WebSocket ReadyStates: {CONNECTING: 0, OPEN: 1, CLOSING: 2, CLOSED: 3}
Message received: Echo: Hello from WebSocket client!
```

✅ Test Checklist

Basic Functionality

- ☐ Server starts without errors
- ☐ Client connects successfully
- ☐ Messages sent from client
- ☐ Messages received by server
- ☐ Server echoes messages
- ☐ Client receives echoed messages

Edge Cases

- ☐ Empty messages handled
- ☐ Very long messages (>64KB)
- ☐ Special characters (UTF-8)
- ☐ Rapid message sending
- ☐ Connection interruption
- ☐ Reconnection after disconnect

Error Handling

- ☐ Invalid frames rejected
- ☐ Malformed data handled
- ☐ Connection errors logged
- ☐ Graceful shutdown works
- ☐ No memory leaks

Performance

- [ ] Low latency (<10ms local)
- [ ] Handles multiple messages
- [ ] No message loss
- [ ] Consistent throughput

🎯 Test Matrix

Test Case	Message Size	Expected Length Encoding	Status
Tiny	1-10 bytes	7-bit	✅
Small	11-125 bytes	7-bit	✅
Medium	126-1000 bytes	16-bit	✅
Large	1001-65535 bytes	16-bit	✅
XLarge	>65535 bytes	64-bit	✅

🔬 Advanced Testing

Frame Inspection

```
// Capture raw frames in DevTools
const ws = new WebSocket("ws://localhost:1338");
const originalSend = ws.send;

ws.send = function (data) {
  console.log("Sending:", data);
  console.log("Length:", data.length);
  return originalSend.call(this, data);
};
```

Hex Dump

```
// Server-side frame logging
function logFrame(buffer) {
  console.log("Frame hex:", buffer.toString("hex"));
  console.log("Frame bytes:", [...buffer]);
}
```

📝 Test Results Template

```
# Test Results - [Date]
```

```
## Environment

- Node.js Version: v18.x.x
- Browser: Chrome 120
- OS: Linux/macOS/Windows

## Test Results

| Test          | Status  | Notes               |
| ------------- | ------- | ------------------- |
| Connection    | ✅ Pass | Connected in 50ms   |
| Echo          | ✅ Pass | All messages echoed |
| UTF-8         | ✅ Pass | Emojis work         |
| Large Message | ✅ Pass | 100KB handled       |
| Rapid Fire    | ✅ Pass | 100 msgs in 200ms   |

## Issues Found

- None

## Performance

- Average latency: 5ms
- Messages/sec: 500
- CPU usage: <5%
- Memory usage: 20MB
```

Happy Testing! 🎉

Run all tests before deploying to production.