WebSocket Systems at Scale: A Technical Guide

Introduction

This document demonstrates the universal PDF generation system with dynamic metadata extraction and professional formatting.

Key Features

The system provides:

- **Universal compatibility**: Works with any markdown file
- **Smart metadata extraction**: Pulls title from content or filename
- **Professional output**: Book-quality typography and layout
- **Dark code themes**: Monokai-inspired syntax highlighting

Code Example

Here's a sample WebSocket implementation:

```
const WebSocket = require('ws');

class ScalableWebSocketServer {
  constructor(port) {
    this.wss = new WebSocket.Server({ port });
    this.clients = new Map();
    this.messageCount = 0;
  }

  initialize() {
    this.wss.on('connection', (ws, req) => {
      const clientId = this.generateClientId();
      this.clients.set(clientId, ws);

      ws.on('message', (message) => {
        this.handleMessage(clientId, message);
      });

      ws.on('close', () => {
        this.clients.delete(clientId);
      });
    });
  }

  handleMessage(clientId, message) {
    this.messageCount++;
    // Process message
    this.broadcast(message, clientId);
  }

  broadcast(message, excludeClient) {
    this.clients.forEach((ws, clientId) => {
      if (clientId !== excludeClient && ws.readyState === WebSocket.OPEN) {
        ws.send(message);
      }
    });
  }

  generateClientId() {
    return Math.random().toString(36).substr(2, 9);
  }
}
```

Performance Metrics

Metric	Value	Notes
Connections	100,000+	Concurrent WebSocket connections
Messages/sec	1,000,000	Peak throughput
Latency	< 10ms	99th percentile
Memory	8GB	For 100k connections

Conclusion

This test document demonstrates the PDF generation system’s ability to handle technical content with code blocks, tables, and professional formatting.