# Client-Device Management System Technical Documentation

## System Overview

The Client-Device Management System is an enterprise-grade web application designed to facilitate the monitoring and management of IoT devices and their associated client relationships. The system provides real-time status tracking of device parameters, particularly focused on gas concentration measurements, with comprehensive client organization capabilities.

## Architecture

### Frontend Architecture

The frontend is built as a responsive web application using modern web standards:

* HTML5 for structure
* CSS3 with Bootstrap 5 framework for styling and responsive design
* JavaScript for client-side logic and dynamic content rendering
* RESTful API consumption for server communication

The interface follows a modular design with separate components for client management, device management, and device details viewing.

### Backend Architecture

The backend implements a multi-protocol server architecture:

* HTTP Server (Express.js): Handles web requests, API endpoints, and serves static content
* TCP Server (Node.js Net module): Maintains persistent connections with IoT devices for real-time data transmission
* MySQL Database: Provides persistent storage for client and device information

## Database Schema

### Clients Collection

The clients collection stores information about system users with the following fields:

* id: Unique client identifier (auto-incrementing integer)
* name: Client's full name
* email: Client's contact email address
* phone: Client's contact phone number
* street: Street address for physical location
* status: Client account status (active, inactive, or pending)
* password: Client authentication password (default: 'Megha@9000')
* device\_id: Associated device identifier for client-device relationship

### Devices Collection

The devices collection maintains device information and current status:

* imei: Unique device identifier (primary key)
* device\_name: Human-readable name for the device
* concentration: Current gas concentration reading (float)
* alarm\_level: Status indicator (normal, warning, or danger)
* gas\_level: Gas level measurement (float)
* unit: Measurement unit (ppm, ppb, etc.)
* UTC: UTC timestamp of last device update
* local\_time: Local timestamp of last device update

## Core Features

### Authentication System

The system implements a session-based authentication mechanism with the following features:

* Username and password validation
* Session timeout after 60 minutes of inactivity
* Session warning notification 5 minutes before timeout
* Session extension capability
* Automatic logout for inactive sessions

### Client Management

The client management module provides:

* Client creation with required fields validation
* Client listing with filtering by status
* Status tracking (active, inactive, pending)
* Client-to-device association
* Contact information management

### Device Management

The device management module delivers:

* Real-time device status monitoring
* Device listing with status filtering
* Status visualization with color-coded indicators
* Support for various measurement units
* UTC and local time tracking for updates

### TCP Communication Protocol

Devices communicate with the system using a JSON-based protocol over TCP:

1. Device connects to server on port 2025
2. Device sends JSON payload containing status information
3. Server validates and processes the data
4. Server responds with confirmation and updated information
5. Connection may be maintained or closed depending on implementation

JSON payload format includes:

* imei: Device identifier
* device\_name: Device name
* concentration: Gas concentration reading
* alarm\_level: Current status
* gas\_level: Measured gas level
* unit: Measurement unit

## API Reference

### Client Management Endpoints

* GET /api/clients: Retrieve all clients
* POST /api/clients: Create a new client
* GET /api/clients/:id: Retrieve a client by ID
* GET /api/clients/device/:deviceId: Retrieve client by associated device ID

### Device Management Endpoints

* GET /api/devices: Retrieve all devices
* POST /api/devices: Register a new device
* GET /api/devices/:imei: Retrieve device by IMEI
* PUT /api/devices/:imei: Update device information
* DELETE /api/devices/:imei: Remove a device
* GET /api/devices/name/:deviceName: Retrieve device by name

### Session Management Endpoints

* POST /api/session/refresh: Extend current session
* GET /api/session/check: Validate session status

## User Interface Components

### Login Interface

The login interface provides secure access to the application with:

* Email and password entry fields
* Visual feedback for validation errors
* Session status messaging
* Automatic redirection for authenticated users

### Client Management Dashboard

The client dashboard offers:

* Searchable client listing
* Status filtering options
* Add/edit client functionality
* Visual indicators for client status
* Device association information

### Device Management Dashboard

The device dashboard presents:

* Comprehensive device listing
* Status filtering by alarm levels
* Real-time status indicators
* Action buttons for viewing details and editing devices
* Search capabilities for finding specific devices

### Device Detail View

The device detail view shows:

* Comprehensive device information display
* Current status with color-coded indicators
* Measurement values with units
* Timestamp information for last updates
* Navigation options to related views

## System Requirements

### Server Requirements

* Node.js v14 or higher
* MySQL 5.7 or higher
* Minimum 1GB RAM
* Minimum 10GB storage
* Network connectivity for both HTTP and TCP protocols

### Client Requirements

* Modern web browser with JavaScript enabled
* Internet connectivity
* Minimum screen resolution of 1024x768

## Installation and Deployment

### Database Setup

1. Install MySQL database server
2. Create a new database named "client\_table1"
3. Configure database connection in [app.js](vscode-file://vscode-app/c:/Users/Lenovo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) with appropriate credentials
4. The system will automatically create required tables on first run

### Application Deployment

1. Install Node.js on the server
2. Clone or copy application files to server
3. Run npm install to install dependencies
4. Configure port settings if necessary in [app.js](vscode-file://vscode-app/c:/Users/Lenovo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)
5. Start server using [node app.js](vscode-file://vscode-app/c:/Users/Lenovo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)
6. Access web interface at http://[server-address]:2026
7. Default login: [megha@gmail.com](vscode-file://vscode-app/c:/Users/Lenovo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) / Megha@9000

### Device Configuration

Devices connecting to the system should be configured to:

1. Connect to server IP address on TCP port 2025
2. Send properly formatted JSON data
3. Handle connection errors and implement reconnection logic
4. Process server responses appropriately

## Security Considerations

* All passwords are stored in plain text in the current implementation; consider implementing hashing in production
* Session timeout provides basic security against unauthorized access
* Consider implementing HTTPS for production deployments
* Review database permissions to restrict access to necessary operations only
* Implement input validation on all user-provided data

## Troubleshooting

### Common Issues

1. **Device Connection Failures**
   * Verify TCP port 2025 is accessible and not blocked by firewalls
   * Check network connectivity between device and server
   * Validate JSON formatting in device messages
2. **Database Connection Issues**
   * Verify MySQL service is running
   * Confirm database credentials are correct
   * Check database existence and permissions
3. **User Interface Problems**
   * Clear browser cache and cookies
   * Verify JavaScript is enabled
   * Check for browser console errors

## Maintenance

Regular maintenance tasks include:

1. Database backups
2. Log rotation and monitoring
3. Security updates for Node.js and dependencies
4. Performance monitoring
5. User account management

## Future Enhancements

Potential system improvements include:

1. Enhanced security with password hashing and role-based access control
2. Expanded device type support
3. Graphical reporting and data visualization
4. Mobile application development
5. Integration with third-party notification systems
6. Historical data analysis and trending