Cologne Open Data MCP Server

A Model Context Protocol (MCP) server providing seamless access to Cologne's Open Data APIs. Built with Node.js, TypeScript, and the @modelcontextprotocol/sdk, this server enables AI assistants and tools to interact with real-time data from the city of Cologne, Germany.

Features

- **Real-time Data Access**: Live parking availability, bike-sharing stations, Rhine water levels, and more
- **Type-Safe**: Fully typed with TypeScript and Zod schema validation
- **Secure**: Built-in SSRF protection, request timeouts, and header injection prevention
- **MCP Compatible**: Works with Claude Desktop, Cursor, and other MCP-compatible clients
- **Well-Documented**: Comprehensive API documentation and usage examples
- **Production-Ready**: Error handling, logging, and timeout management

Deployment Options

This MCP server supports two modes:

1. **Local STDIO Mode** - For Claude Desktop and local MCP clients
2. **Web SSE Mode** - For web deployment on platforms like Render

See DEPLOYMENT.md for detailed deployment instructions.

Available Tools

Tool Name	Description	Data Source
`health`	Server status check	-
`http.get_json`	Generic HTTP GET for JSON/REST endpoints	Any URL
`koeln.parking`	Current parking facility availability	Cologne Parking API
`koeln.baustellen_caps`	Construction sites WFS capabilities	Cologne GeoPortal
`koeln.rheinpegel`	Rhine river water level	Cologne Water Level Service
`koeln.kvb_rad.stations`	KVB bike-sharing stations	Nextbike API
`koeln.oparl.bodies`	Political bodies list	OParl Cologne
`koeln.oparl.body`	Single political body details	OParl Cologne

🚀 Quick Start

Installation

```
npm install cologne-open-data-mcp
```

Or install globally:

```
npm install -g cologne-open-data-mcp
```

Local Usage with Claude Desktop (STDIO)

1. Edit your Claude Desktop configuration file:
   - **macOS**: `~/Library/Application Support/Claude/claude_desktop_config.json`
   - **Windows**: `%APPDATA%\Claude\claude_desktop_config.json`
2. Add the server configuration:

```
{
  "mcpServers": {
    "cologne-open-data": {
      "command": "npx",
      "args": ["cologne-open-data-mcp"]
    }
  }
}
```

3. Restart Claude Desktop

Web Deployment (SSE Mode)

For production deployment on Render or similar platforms:

1. **Deploy to Render:**
   - Follow the DEPLOYMENT.md guide
   - Use the `render.yaml` configuration included
   - Server will be available at `https://your-app.onrender.com`
2. **Connect to SSE Endpoint:**

```
import { SSEClientTransport } from
'@modelcontextprotocol/sdk/client/sse.js';
```

```
const transport = new SSEClientTransport(
  new URL('https://your-app.onrender.com/sse')
);
```

3. **Test Deployment:**

```
# Health check
curl https://your-app.onrender.com/health
```

Usage with Other MCP Clients (STDIO)

The server communicates via STDIO and can be integrated with any MCP-compatible client:

```
import { StdioClientTransport } from
'@modelcontextprotocol/sdk/client/stdio.js';
import { spawn } from 'child_process';

const transport = new StdioClientTransport({
  command: 'npx',
  args: ['cologne-open-data-mcp']
});
```

🛠️ Development

Prerequisites

- Node.js >= 18.0.0
- npm or yarn

Setup

1. Clone the repository:

```
git clone https://github.com/ErtanOz/Cologne-Open-Data-Mcp.git
cd Cologne-Open-Data-Mcp
```

2. Install dependencies:

```
npm install
```

3. (Optional) Configure custom API endpoints:

```
cp .env.example .env
# Edit .env with your preferred endpoints
```

4. Run in development mode:

```
npm run dev
```

5. Build for production:

```
npm run build
npm start
```

🔧 Configuration

Environment variables can be set in a `.env` file:

```
PARKING_URL=https://www.stadt-koeln.de/externe-dienste/open-data/parking.php
BAUSTELLEN_WFS=https://geoportal.stadt-
koeln.de/wss/service/baustellen_wfs/guest?SERVICE=WFS&REQUEST=GetCapabilities
RHEINPEGEL_URL=https://www.stadt-koeln.de/interne-
dienste/hochwasser/pegel_ws.php
NEXTBIKE_URL=https://api.nextbike.net/maps/nextbike-live.xml?city=14
OPARL_BODIES_URL=https://buergerinfo.stadt-koeln.de/oparl/bodies
```

📖 API Examples

Check Server Health

```
// Input
{
  "tool": "health",
  "arguments": {
    "echo": "Hello MCP Server"
  }
}

// Output
"Hello MCP Server"
```

Get Parking Data

```
// Input
{
  "tool": "koeln.parking",
  "arguments": {}
}

// Output (example)
{
  "source": "https://www.stadt-koeln.de/externe-dienste/open-data/parking.php",
  "payload": [
    {
      "name": "Parkhaus Am Dom",
      "free": 245,
      "total": 550,
      "status": "open"
    }
    // ... more parking facilities
  ]
}
```

Get KVB Bike Stations

```
// Input
{
  "tool": "koeln.kvb_rad.stations",
  "arguments": {
    "limit": 10,
    "onlyActive": true
  }
}

// Output (example)
{
  "source": "https://api.nextbike.net/maps/nextbike-live.xml?city=14",
  "totalStations": 287,
  "returnedStations": 10,
  "stations": [
    {
      "id": "123456",
      "name": "Hauptbahnhof",
      "bikes": 8,
      "freeRacks": 12,
      "active": true,
      "lat": 50.9429,
      "lng": 6.9589
    }
    // ... more stations
  ]
}
```

Generic HTTP GET

```
// Input
{
  "tool": "http.get_json",
  "arguments": {
    "url": "https://api.example.com/data",
    "headers": {
      "Accept": "application/json"
    }
  }
}
```

Security Features

- **SSRF Protection**: Validates URLs to prevent requests to localhost and private IP ranges
- **Request Timeouts**: 10-second timeout for all HTTP requests
- **Header Injection Prevention**: Sanitizes custom headers
- **Type Validation**: Zod schema validation for all inputs
- **Error Handling**: Comprehensive error messages without exposing sensitive information

Docker Deployment

A `Dockerfile` is included for containerized deployment:

```
# Build the image
docker build -t cologne-open-data-mcp .

# Run the container
docker run -i cologne-open-data-mcp
```

Project Structure

```
Cologne-Open-Data-Mcp/
├── src/
│   ├── server.ts              # MCP server initialization
│   └── tools/                 # Tool implementations
│       ├── health.ts          # Health check
│       ├── http_get.ts        # Generic HTTP GET
│       ├── parking.ts         # Parking data
│       ├── baustellen.ts      # Construction sites
│       ├── rheinpegel.ts      # Rhine water level
│       ├── kvb_rad.ts         # Bike-sharing stations
│       ├── oparl_bodies.ts    # Political bodies
```

```
│       ├── types.ts           # Type definitions
│       └── utils.ts           # Utility functions
├── dist/                      # Compiled JavaScript (generated)
├── package.json
├── tsconfig.json
├── LICENSE
└── README.md
```

🤝 Contributing

Contributions are welcome! Please feel free to submit a Pull Request.

1. Fork the repository
2. Create your feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- City of Cologne for providing Open Data APIs
- Model Context Protocol team for the MCP SDK
- Nextbike for bike-sharing data
- All contributors to this project

📞 Support

- **Issues**: GitHub Issues
- **Discussions**: GitHub Discussions

⚠️ Important Notes

About ChatGPT Compatibility

ChatGPT does NOT support MCP. This server is designed for:

- ☑ Claude Desktop (Anthropic)
- ☑ Cursor IDE
- ☑ Other MCP-compatible clients

For ChatGPT integration, you would need to create a separate REST API wrapper with OpenAPI specification.

Deployment Modes

- **STDIO Mode**: Local desktop use (Claude Desktop, etc.)
- **SSE Mode**: Web deployment (Render, cloud platforms)

🗺️ Roadmap

- [x] STDIO transport for local use
- [x] SSE transport for web deployment
- [x] Render deployment configuration
- [ ] Add more Cologne Open Data sources
- [ ] Implement caching for improved performance
- [ ] Add rate limiting capabilities
- [ ] Create comprehensive test suite
- [ ] Add monitoring and metrics
- [ ] Support for additional data formats

Made with 💚 for the Cologne community