Table of Contents

AI solution Ask Directo: just ask, and the AI gives you answers based on your Directo data 3
Directo MCP Server Setup — Claude Desktop and Gemini CLI 3
1. Introduction — why MCP? 3
What is MCP? 3
What's the benefit? 3
Architecture 3
2. Checking prerequisites 4
2.1 Required for everyone 4
2.2 Additionally for Claude Desktop users 4
2.3 Additionally for Gemini CLI users 5
2.4 Admin rights — when are they needed? 5
3. Shared steps (do once) 6
Step 3.1 — Installing Node.js (required) 6
Choose your path 7
PATH A — Quick install with scripts (recommended) 7
PATH B — Manual, step by step (alternative to Path A) 9
Step B1 — Setting the PowerShell execution policy 9
Step B2 — CredentialManager module 9
Step B3 — Creating the MCP server folder 10
Step B4 — Placing directo-mcp.js 10
Step B5 — Creating package.json 11
Step B6 — Installing dependencies 11
Step B7 — Securely saving the API key 11
4. Setting up Claude Desktop 12
Step 4.1 — Configuration file location 12
Step 4.2 — Configuration file 13
Step 4.3 — Restarting Claude Desktop 14
Step 4.4 — Testing 14
5. Setting up Gemini CLI 15
Step 5.1 — Installing Gemini CLI 15
Step 5.2 — Setting up the Google API key 16
Step 5.3 — Configuration file location 16
Step 5.4 — Adding the MCP server 16
Step 5.5 — Trusting the folder (Trusted Folders) 17
Step 5.6 — Testing 18
6. Troubleshooting 18
Common errors (either client) 18
Claude Desktop-specific errors 19
Gemini CLI-specific errors 19
7. MCP tools 20
8. Security 20
9. References 20

AI solution Ask Directo: just ask, and the AI gives you answers based on your Directo data

Directo MCP Server Setup — Claude Desktop and Gemini CLI

This guide explains how to connect the Directo ERP system to an AI assistant (Claude or Gemini) through a local MCP (Model Context Protocol) server. The guide is intended for both Claude Desktop and Gemini CLI users — the shared steps only need to be done once, after which you pick the section for your AI client.

1. Introduction — why MCP?

What is MCP?

MCP (Model Context Protocol) is an open standard that lets AI assistants communicate directly with external systems — databases, APIs (Application Programming Interfaces), and applications.

Without MCP, working with an AI assistant looks like this:

1. You copy data out of Directo
2. You paste it into the AI conversation
3. The AI answers
4. You copy the answer back into Directo

With MCP, it looks like this:

1. You ask in natural language: „Which items are below minimum stock level?“
2. The AI queries the Directo API (Application Programming Interface) itself
3. The AI analyzes the data and answers immediately

What's the benefit?

Situation	Without MCP	With MCP
Stock level check	Open Directo → search → copy → paste into AI	„Which items have fewer than 10 units in stock?“
Invoice analysis	Export CSV → upload to AI → ask	„Which customers haven't paid in the last 90 days?“
Sales report	Build manually → copy → analyze	„Compare this month's sales to last year“
Order tracking	Open several views → copy info	„Which orders are more than 5 days overdue?“

Architecture

```
You (natural language)
        ↕
```

```
AI client (Claude Desktop or Gemini CLI)
        ↕  MCP protocol (local)
MCP Server (on your computer, Node.js)
        ↕  HTTPS
Directo API (cloud)
```

The MCP server runs **on your computer**. The Directo API key never leaves your machine. The data retrieved from Directo does, however, travel onward to the AI provider's cloud so the model can read it and formulate a natural-language answer — the same as any other message content in the conversation.



Local install — important to understand:

- The MCP server is installed **only on your own computer**
- It is **not shared** — colleagues cannot access your MCP server
- Every user must do all the steps **themselves**, on their own computer
- The MCP server only works while **your computer is on** and the AI client is running
- If you want to share an MCP server across multiple users, you need a separate hosted-server solution (out of scope for this guide)

2. Checking prerequisites

Check every prerequisite before installing. A missing prerequisite is the most common cause of failure.

2.1 Required for everyone

Prerequisite	How to check	Fix
Windows 10 or 11 (64-bit)	Start → Settings → System → About	—
Node.js 22 or newer (LTS)	`node –version`	nodejs.org → LTS
npm	`npm –version` (only works after execution policy is set, see below)	Comes bundled with Node.js
PowerShell execution policy	`Get-ExecutionPolicy` → should return RemoteSigned	See below
CredentialManager module	`Get-Module -ListAvailable CredentialManager`	See below
Directo API key	Directo → Admin → Settings → API	Ask your Directo administrator

2.2 Additionally for Claude Desktop users

Prerequisite	How to check	Link
Claude Desktop installed	Claude appears in the Start menu	https://claude.ai/download
Anthropic account	Sign-in works	https://claude.ai

Official documentation: https://docs.claude.ai

2.3 Additionally for Gemini CLI users



⚠ **The Gemini CLI free tier still exists, but has changed.** Depending on your authentication method:

- **Google account (Gemini Code Assist, individual user):** ~60 requests/min and ~1000 requests/day
- **Free Gemini API key (unpaid):** narrower — ~10 requests/min, ~250 requests/day, Flash models only

On June 18, 2026, the Gemini CLI will be replaced by **Antigravity CLI** for the unpaid tier and Google One users — this affects a specific authentication path, not free access as a whole. For higher volume or Pro models you need a Google AI Pro/Ultra subscription or a Gemini Code Assist license. Check your project's actual limits in AI Studio, as they change frequently and depend on region and account status.

Prerequisite	How to check	Link
Google account	gmail.com access works	https://accounts.google.com
Google AI Studio API key	https://aistudio.google.com/apikey	Create an API key
Gemini CLI installed	`gemini –version`	`npm install -g @google/gemini-cli`

Official documentation: https://geminicli.com/docs

2.4 Admin rights — when are they needed?



Good news: almost every step can be done **without** administrator rights. You genuinely need admin rights at only one point — installing Node.js.

Step	Admin rights?	Reason
Installing Node.js	**Yes** — the installer requires it	System-level install (Program Files)
PowerShell execution policy	No	`-Scope CurrentUser` writes to the HKCU registry
CredentialManager module	No	`-Scope CurrentUser` installs under the user's profile
npm install (MCP server)	No	User-level install
Claude Desktop config	No	User folders
Gemini CLI config	No	User folders
Saving the API key	No	Credential Manager, user-level entry

Risks and an important warning:

- Node.js installer — low risk, official software
- **Do not open any of the other steps as administrator**, even if an earlier version of this guide or examples found online suggest it — `-Scope CurrentUser` exists specifically to avoid needing admin rights
- If your everyday account and your admin rights come from **separate Windows accounts** (common in corporate/domain environments), opening something as administrator runs the action under the *other* account's profile — the module, setting, or key then „disappears“ from your regular user's point of view. See the troubleshooting chapter for details.
- Don't run `npm install` as administrator — this creates permission conflicts
- `npm install -g` (e.g. Gemini CLI, Step 5.1) only needs admin rights if npm's global prefix points at the `Program Files` folder — Step 5.1 includes a fix that avoids this

3. Shared steps (do once)



You do these steps **on your own computer**. If a colleague wants to use the same setup, they need to do these steps **on their own computer** independently.

Step 3.1 — Installing Node.js (required)



This is the only step that is **required on both paths** below and cannot be automated (it needs admin rights). Do this before moving on, regardless of which path you choose.

1. Go to https://nodejs.org and download the **LTS** version (22 or newer)
2. Run the downloaded `.msi` file as administrator
3. Install with the default settings — don't change the install location

Verify the install (regular PowerShell, not admin):

```
node --version
```

Expected result:

```
v22.11.0 or newer
```



We deliberately don't check `npm –version` here yet — npm is a PowerShell script (`npm.ps1`) and will give a `running scripts is disabled` error before the execution policy is set (Path A script, or Step B1), even if Node.js and npm themselves are installed correctly. This is not a bug, it's expected behavior at this stage. `npm –version` will work after completing Path A/B.



If PowerShell says „not recognized“ — close the window and reopen it. Node.js only

adds itself to PATH in new PowerShell sessions.

Choose your path

Below are **two alternative paths** to reach the same result — a working MCP server, with the API key and AI client set up. **Do only ONE of them, not both.**

	Path A — Scripts	Path B — Manual
Time	~5 minutes	~20 minutes
Best for	Most users	You want to understand every step yourself, or Path A doesn't work in your environment
Covers steps	A1—A2	B1—B7

If you choose Path A and it succeeds, **skip Path B (B1—B7) entirely** — those steps describe, manually, exactly what the scripts already did for you. Path B is troubleshooting reference material, not an additional required step.

PATH A — Quick install with scripts (recommended)

Step A1 — Downloading the files

Download `directo-mcp-files.zip` and extract it:

directo-mcp-files.zip

The files are attached to the wiki as a **ZIP archive**, not as individual links — DokuWiki blocks `.js` and `.ps1` file uploads by default (for security, see `conf/mime.conf`), but `.zip` is allowed by default. Don't rename the files to a different extension (e.g. `.txt`) as a „fix“ — the wrong extension is exactly what causes `MODULE_NOT_FOUND`-type errors later.

⚠ **Current limitation:** the scripts inside this archive still print their status messages in Estonian (e.g. „Kontrollin Node.js-i“). They work identically regardless of the message language — only the on-screen text differs. An English-language version of the scripts is planned; until then, refer to the step descriptions in this guide to know what each stage of the output means.

Extract it (right-click the ZIP → **Extract All**, or in PowerShell):

```
Expand-Archive "$env:USERPROFILE\Downloads\directo-mcp-files.zip" -
DestinationPath "$env:USERPROFILE\Downloads\directo-mcp-files"
cd "$env:USERPROFILE\Downloads\directo-mcp-files"
```

Before running the scripts, two Windows default security restrictions must be cleared — both are required, not optional:

1) Allow running scripts in PowerShell at all:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```



Without this, every following script will fail with `running scripts is disabled on this system` / `UnauthorizedAccess`. `-Scope CurrentUser` does not require admin rights.

2) Remove Windows' „downloaded from the internet“ (Mark of the Web) flag:

```
Get-ChildItem "$env:USERPROFILE\Downloads\directo-mcp-files" -Recurse |
Unblock-File
```



This is also required, not optional — files downloaded through a browser are blocked by default **even if the execution policy is already correct**. Windows tags internet-downloaded files with a separate security marker, which also carries over to files extracted from a ZIP archive — `RemoteSigned` requires a digital signature for such files, which our scripts don't have. Without this step you'll see `is not digitally signed` / `UnauthorizedAccess`, which looks confusingly similar to the error from step 1, but has a different cause and fix.

The archive contains four files: `directo-mcp.js` (the MCP server itself), `set-directo-apikey.ps1`, `install-claude-mcp.ps1`, `install-gemini-mcp.ps1`.

Step A2 — Running the scripts

In the same PowerShell window (not as administrator), in the folder you just extracted:

```
# API key (shared by both clients, do this first)
.\set-directo-apikey.ps1

# Pick your AI client (or both):
.\install-claude-mcp.ps1
.\install-gemini-mcp.ps1
```

Each script prints a summary at the end — what's done (□) and what still needs manual action (□).



If the scripts succeeded, you're done with Chapter 3. Go straight to Chapter 4.3 (Claude Desktop) and/or Chapter 5 (Gemini CLI) — Path B (below) is not needed.

PATH B — Manual, step by step (alternative to Path A)



This is an **alternative to Path A**, not an addition to it. Only do this if the Path A scripts didn't work, or if you want to go through every step yourself. If Path A already succeeded, skip this section.

Step B1 — Setting the PowerShell execution policy

Check the current setting:

```
Get-ExecutionPolicy
```

If the answer isn't `RemoteSigned`, fix it in a **regular** PowerShell window (not as administrator):

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```



`-Scope CurrentUser` writes the setting only to the `HKEY_CURRENT_USER` registry hive, not a machine-wide location — that's why it doesn't need admin rights. Admin rights would only be needed for `-Scope LocalMachine`, which this guide does not use.

It will ask for confirmation → press Y → Enter.

Step B2 — CredentialManager module

Open a **regular** PowerShell window (not as administrator) — under exactly the same user account the MCP server will later run under:

```
Install-Module -Name CredentialManager -Force -Scope CurrentUser
```



`-Scope CurrentUser` exists specifically to avoid needing admin rights — the module installs under your own profile (`Documents\WindowsPowerShell\Modules`). If the command asks to confirm the NuGet provider or an „untrusted repository," answer Y — these steps don't need admin rights either.

Verify the install succeeded (in the same regular PowerShell window):

```
Get-Module -ListAvailable CredentialManager
```

Should show the module's name and version.

⚠ If your everyday work account and your admin rights come from **separate Windows accounts** (common in corporate/domain setups), make sure this step and the later Step B7 run under **the same account** that `node directo-mcp.js` will later run under. A PowerShell window opened with elevation may actually run under a different account's identity, which would leave the module or API key under the wrong profile, and the later step will fail invisibly.

Step B3 — Creating the MCP server folder

Open a regular PowerShell window (not admin) and run:

```
New-Item -ItemType Directory -Force -Path "$env:USERPROFILE\directo-mcp"
```

Always use the `$env:USERPROFILE` variable, rather than manually typing `C:\Users\YOURNAME`. On domain-joined computers (corporate/AD environments) your actual profile folder may differ from your sign-in name — for example it may include a domain suffix (`YOURNAME.DOMAIN`). `$env:USERPROFILE` always gives the real, correct path, regardless.

Step B4 — Placing directo-mcp.js

This file contains all 32 Directo API tools and has been **fully schema-verified** — every parameter has been checked against Directo's actual ws_*.json schemas. Don't copy this file manually from Notepad or any other source — always use a fresh, official copy to avoid ending up with an outdated or incorrectly-parameterized version.

Download the `directo-mcp.js` file — it's included in the `directo-mcp-files.zip` archive linked under Path A (extract it, see Step A1) — and place it in this folder:

```
$env:USERPROFILE\directo-mcp\directo-mcp.js
```

The simplest way: if the file is already in the extracted Downloads subfolder, copy it in PowerShell:

```
Copy-Item "$env:USERPROFILE\Downloads\directo-mcp-files\directo-mcp.js"
"$env:USERPROFILE\directo-mcp\directo-mcp.js"
```

Make sure the file name is exactly `directo-mcp.js` — not `directo-mcp.js.txt`. Check:

```
Test-Path "$env:USERPROFILE\directo-mcp\directo-mcp.js"
```

Should return `True`. If `False`, check the exact file name:



```
Get-ChildItem "$env:USERPROFILE\directo-mcp"
```

Step B5 — Creating package.json

Open a new Notepad window and paste:

```
{
  "name": "directo-mcp",
  "version": "1.0.0",
  "type": "module",
  "description": "Directo ERP MCP Server",
  "main": "directo-mcp.js",
  "scripts": {
    "start": "node directo-mcp.js"
  },
  "dependencies": {
    "@modelcontextprotocol/sdk": "^1.0.0",
    "zod": "^3.22.0"
  }
}
```

Save it in the same folder, named `package.json` (file type: All Files).

Step B6 — Installing dependencies

Regular PowerShell (not admin):

```
cd "$env:USERPROFILE\directo-mcp"
npm install
```

A successful install creates a `node_modules` folder and a `package-lock.json` file in this folder.



This is a **local** npm install (goes under the `directo-mcp` folder's own node_modules), not global — it never needs admin rights or any special npm-prefix configuration, regardless of how Node.js was originally installed.

Step B7 — Securely saving the API key

The Directo API key is saved to Windows Credential Manager — encrypted, under your user account only. No one else on the computer can see this key.

```
New-StoredCredential -Target DIRECTO_API_KEY -UserName directo -Password
YOUR_REAL_KEY_HERE -Persist LocalMachine
```

Verify the save succeeded:

```
(Get-StoredCredential -Target
DIRECTO_API_KEY).GetNetworkCredential().Password
```

Should display your API key.

Rotating the API key in the future:

```
Remove-StoredCredential -Target DIRECTO_API_KEY
New-StoredCredential -Target DIRECTO_API_KEY -UserName directo -Password
NEW_KEY -Persist LocalMachine
```



Run this step under **exactly the same user account** that `node directo-mcp.js` will later run under. See the warning under Step B2 about the separate-admin-account risk.



The Path A script (`set-directo-apikey.ps1`) does the same thing more securely (asks for the key without echoing it to the screen) and also works for rotation — use it if you can.

4. Setting up Claude Desktop



First complete Chapter 3 (Step 3.1 + either Path A or Path B). Setting up Claude Desktop assumes the MCP server already exists.



The MCP server only works **on your own computer** and only inside the **Claude Desktop application** — not in the claude.ai web version, and not on other users' computers.



Recommended: if you ran the `install-claude-mcp.ps1` script under Path A (Chapter 3), Steps 4.1—4.2 are already done — the script detects, patches, and backs up the configuration file automatically. Go straight to Step 4.3.

Step 4.1 — Configuration file location

Claude Desktop has **two possible** configuration locations on Windows, depending on how it was installed:

Install type	Location
Direct `.exe` installer (typical)	`%APPDATA%\Claude\`
Microsoft Store / MSIX	`%LOCALAPPDATA%\Packages\Claude_pzs8sxrjxfjjc\LocalCache\Roaming\Claude\`



This is a documented, currently active Claude Desktop issue: with an MSIX install, Windows **virtualizes** the `%APPDATA%\Claude\` location transparently to a different path. If you edit the wrong (standard) file, the change never reaches the app, and the MCP server silently fails to load, with no explanatory error.

Check which one applies on your machine — open File Explorer and try both in the address bar:

```
%APPDATA%\Claude
```

If a `claude_desktop_config.json` file already exists there (or Claude Desktop has already created something there), use this path. If not, try:

```
%LOCALAPPDATA%\Packages\Claude_pzs8sxrjxfjjc\LocalCache\Roaming\Claude
```



If neither folder exists, run Claude Desktop at least once — it creates the necessary folders automatically (usually at the standard path).

Step 4.2 — Configuration file

Create a file named `claude_desktop_config.json` in the (correct, as identified in Step 4.1) folder.



When creating the file with Notepad: File → Save As → file type **All Files (*.*)** → name: `claude_desktop_config.json`

File contents:

```
{
  "mcpServers": {
    "directo": {
      "command": "node",
      "args": [
        "C:\\Users\\YOURNAME\\directo-mcp\\directo-mcp.js"
      ]
    }
```

```
  }
}
```



Replace YOURNAME with your **actual** profile folder name — not your assumed username. On domain-joined (AD) computers, the actual profile folder may differ from your sign-in name (e.g. include a domain suffix). Check the exact value:

```
$env:USERPROFILE
```

Use exactly what the command returns in the JSON (with double backslashes). Example for illustration — if `$env:USERPROFILE` returns `C:\Users\john.user.COMPANY`, the path in the JSON should be `C:\\Users\\john.user.COMPANY\\directo-mcp\\directo-mcp.js`. Whether or not a domain suffix (`.COMPANY`) is present depends on your own computer's setup — always use your own command's actual output, not this example.



If the file already contains other content (Claude Desktop's own settings, e.g. a `preferences` block) — **do not overwrite it**. `mcpServers` must be at the **root level** of the file, at the same level as `preferences`, not inside it. If you're unsure about correct brace placement, use the Path A script instead, which edits the file as a structure (`ConvertFrom-Json`/`ConvertTo-Json`), not as raw text — this completely eliminates brace-placement errors.



The API key is **not** added to the configuration file — the server reads it automatically from Credential Manager (Path A script, or Step B7).

Step 4.3 — Restarting Claude Desktop

1. Right-click the Claude icon in the **system tray** (near the clock, in the taskbar) → **Quit** / **Exit**
2. Simply closing the window is not enough — Claude Desktop keeps running in the background
3. If Quit isn't available — open Task Manager → find the Claude process → End Task
4. Open Claude Desktop again and wait a few seconds

Step 4.4 — Testing

In a Claude Desktop conversation, type:

> Show me all stock levels

Claude should automatically use the `get_stocklevels` tool and return data from Directo.

Checking the logs if something doesn't work:

The log location follows the same two possibilities as Step 4.1:

```
# Direct .exe installer:
Get-Content "$env:APPDATA\Claude\logs\mcp-server-directo.log" -Tail 30

# Microsoft Store / MSIX:
Get-Content
"$env:LOCALAPPDATA\Packages\Claude_pzs8sxrjxfjjc\LocalCache\Roaming\Claude\l
ogs\mcp-server-directo.log" -Tail 30
```

Official Claude Desktop MCP documentation: modelcontextprotocol.io — Connect to local MCP servers

5. Setting up Gemini CLI



First complete Chapter 3 (Step 3.1 + either Path A or Path B). Setting up Gemini CLI assumes the MCP server already exists.



The MCP server only works **on your own computer**, and only inside the terminal-based Gemini CLI application — not the Gemini web version, and not on other users' computers.



⚠ Before continuing, make sure you have valid authentication — either a Google account (Gemini Code Assist), a free Gemini API key (Flash models), or a paid subscription. Limits and model choice depend on the authentication method (see Step 2.3).



Recommended: if you ran the `install-gemini-mcp.ps1` script under Path A (Chapter 3), Steps 5.1 and 5.3—5.4 are already done. Only go through Step 5.2 (Google API key — the script doesn't do this, since it's separate Google, not Directo, authentication) and Steps 5.5—5.6.

Step 5.1 — Installing Gemini CLI



This step does not need admin rights — provided npm's global install location points into your own profile. If not yet configured, do this once (regular PowerShell, not as administrator):

```
mkdir "$env:APPDATA\npm-global" -Force
npm config set prefix "$env:APPDATA\npm-global"
[Environment]::SetEnvironmentVariable("Path", $env:Path +
";$env:APPDATA\npm-global", "User")
```

Close PowerShell completely and open a **new** window (PATH only refreshes in a new session), then install Gemini CLI:

```
npm install -g @google/gemini-cli
```

Verify:

```
gemini --version
```



⚠ If the command still asks for admin rights, the likely cause is that npm's prefix was already set (e.g. by another piece of software) to point at `C:\Program Files\…`. Check the current setting with `npm config get prefix` — after the step above it should show `%APPDATA%\npm-global`, not a Program Files folder.

Step 5.2 — Setting up the Google API key

Go to https://aistudio.google.com/apikey and create an API key.

Save the key as a persistent environment variable:

```
[System.Environment]::SetEnvironmentVariable("GEMINI_API_KEY",
"YOUR_GOOGLE_API_KEY", "User")
```

Close PowerShell and reopen it. Verify:

```
$env:GEMINI_API_KEY
```

Step 5.3 — Configuration file location

The Gemini CLI configuration is located on Windows at:

```
$env:USERPROFILE\.gemini\settings.json
```

Check whether the folder exists:

```
Test-Path "$env:USERPROFILE\.gemini"
```

If the answer is `False`, create the folder:

```
New-Item -ItemType Directory -Force -Path "$env:USERPROFILE\.gemini"
```

Step 5.4 — Adding the MCP server

Create or edit the file `$env:USERPROFILE\.gemini\settings.json`:

```
{
  "mcpServers": {
    "directo": {
      "command": "node",
      "args": [
        "C:\\Users\\YOURNAME\\directo-mcp\\directo-mcp.js"
      ],
      "env": {
        "DIRECTO_API_KEY": "$DIRECTO_API_KEY"
      }
    }
  }
}
```



Replace YOURNAME with your **actual** profile folder name — check the exact value with `$env:USERPROFILE` (see the warning under Step 4.2 on this same topic).



A note on the API key logic: `directo-mcp.js` reads the key from Credential Manager first (Path A script, or Step B7) — this works **exactly the same way for Gemini as for Claude Desktop**, since both run the same server file. The env block here is a **fallback**, not the only path — if Credential Manager is already set up, this block isn't strictly necessary, though it doesn't hurt either. If you do want to use it, save the value as an environment variable:

```
[System.Environment]::SetEnvironmentVariable("DIRECTO_API_KEY", "YOUR_DIRECTO_API_KEY", "User")
```

Gemini CLI supports environment-variable expansion in env values using $VAR_NAME syntax — that's why the JSON contains „$DIRECTO_API_KEY“, not the actual key.

Step 5.5 — Trusting the folder (Trusted Folders)

Gemini CLI has a security feature called **Trusted Folders**, which by default doesn't allow connecting MCP servers from an untrusted folder. If this feature is enabled, the folder needs to be trusted.

The simplest way — trust the server directly in `settings.json`, by adding „`trust“: true` to the `directo` block:

```
{
  "mcpServers": {
    "directo": {
      "command": "node",
      "args": [
        "C:\\Users\\YOURNAME\\directo-mcp\\directo-mcp.js"
      ],
      "env": {
```

```
        "DIRECTO_API_KEY": "$DIRECTO_API_KEY"
      },
      "trust": true
    }
  }
}
```



„trust“: true skips the tool-call confirmation dialogs. Only use this for servers you fully manage yourself — this Directo server is such a case.

Alternative — if Trusted Folders is enabled, Gemini CLI shows a trust dialog the first time it opens this folder. Choose **Trust folder** there. The choice is saved to `~/.gemini/trustedFolders.json` and is only asked once per folder. Use the `/permissions` command inside the CLI to change the trust level later.

Step 5.6 — Testing

Start Gemini CLI:

```
gemini
```

Check the MCP connection:

```
/mcp list
```

Should show:

```
 directo - Ready (32 tools)
```

Test query:

> Show me all stock levels

Official Gemini CLI MCP documentation: geminicli.com/docs/tools/mcp-server

6. Troubleshooting

Common errors (either client)

Error message	Cause	Fix
DIRECTO_API_KEY puudub *(literal log text, still Estonian — see note below)*	Credential Manager save failed	Redo Step B7 (or Path A: `.\set-directo-apikey.ps1`)

Error message	Cause	Fix
`Cannot find module`	npm install failed, OR directo-mcp.js is in the wrong folder/has the wrong name	Redo Step B6; check `Test-Path „$env:USERPROFILE\directo-mcp\directo-mcp.js“`
`SyntaxError`	directo-mcp.js file is corrupted or incomplete	Download the file again, see Step B4 (or Path A: A1)
`Incorrect filter id`	Invalid API filter name	Known limitation — use without that filter
`401 / 403`	Directo API key is invalid or expired	Check the key in Directo admin
`running scripts is disabled`	PowerShell execution policy	`Set-ExecutionPolicy RemoteSigned -Scope CurrentUser` (Step A1 or B1)
`is not digitally signed` / `UnauthorizedAccess`	Windows Mark-of-the-Web (file downloaded via a browser)	`Get-ChildItem <folder> -Recurse | Unblock-File` (see Step A1)
`npm not recognized`	Node.js is not on PATH	Close and reopen PowerShell



The first row above („DIRECTO_API_KEY puudub“) shows exactly what will appear in the log file — it's a literal message printed by `directo-mcp.js`, which is still written in Estonian pending an English version of that file. If you see this text in your logs, it means „DIRECTO_API_KEY is missing,“ and the fix column above still applies.

Claude Desktop-specific errors

Problem	Fix
No hammer icon visible	Check the log file (see below)
MCP server doesn't start	Check that the config file's name doesn't have a `.txt` extension
Config folder is missing	Run Claude Desktop once — the folders are created automatically
Config edited, but nothing changes	Make sure you're editing the RIGHT file — see Step 4.1 (two possible locations, MSIX vs. direct install)

Log file (try both, see Step 4.1):

```
Get-Content "$env:APPDATA\Claude\logs\mcp-server-directo.log" -Tail 30
Get-Content "$env:LOCALAPPDATA\Packages\Claude_pzs8sxrjxfjjc\LocalCache\Roaming\Claude\logs\mcp-server-directo.log" -Tail 30
```

Gemini CLI-specific errors

Problem	Fix
`/mcp list` shows Disconnected	Add `„trust“: true` to the server config (Step 5.4), or trust the folder via the dialog / the `/permissions` command
`GEMINI_API_KEY` error	Redo Step 5.2, open a new PowerShell window
Rate limit / 429 error	Free-tier request limit exceeded — wait for the limit to reset, or consider a paid subscription (see Step 2.3)

Gemini CLI diagnostics:

```
/mcp list
```

7. MCP tools

Both clients (Claude Desktop and Gemini CLI) use the same tools:

Group	Tools
Sales	get_invoices, get_orders, get_sales_quotations, get_sales_contracts, get_receipts
Customers	get_customers, get_suppliers, get_contacts
Products / stock	get_items, get_itemclasses, get_stocklevels, get_stocklevels_sn, get_allocations, get_priceformulas
Purchasing	get_purchase_orders, get_purchase_invoices, get_purchase_payments, get_stock_receipts
Accounting	get_transactions, get_accounts, get_financial_budgets, get_nettings
Logistics	get_deliveries, get_movements
Other	get_projects, get_objects, get_resources, get_events, get_users, get_recipes, get_replacementlogs
Sync	get_deleted

8. Security

Measure	Explanation
Credential Manager (both clients)	API key is encrypted, visible only to your Windows account — Claude and Gemini read the same entry
Environment variable (Gemini fallback)	If Credential Manager is set up, the `env` block isn't strictly necessary — see the note under Step 5.4
Local server	The MCP server only runs on your computer, it is not publicly reachable
Minimal permissions	Restrict the Directo API key's permissions to only the resources you need
OneDrive/Dropbox	Exclude the `directo-mcp` folder from cloud sync
Admin rights	Only for installing Node.js (Step 3.1) — a shared prerequisite for both paths — everything else, including the CredentialManager module, runs under `-Scope CurrentUser` without admin rights (see 2.4)

9. References

Claude Desktop:

- Download Claude Desktop
- Claude Desktop MCP setup
- Guide: local MCP servers on Claude Desktop
- Official MCP documentation

Gemini CLI:

- Gemini CLI documentation
- Gemini CLI MCP setup
- Google AI Studio API keys
- Gemini Code Assist documentation

Directo:

- Directo API documentation

General:

- Download Node.js
- CredentialManager PowerShell module

Quick-install scripts (see Path A, Chapter 3):

- `directo-mcp.js` — MCP server (32 tools, schema-verified)
- `set-directo-apikey.ps1` — API key management (initial setup + rotation)
- `install-claude-mcp.ps1` — automatic Claude Desktop install
- `install-gemini-mcp.ps1` — automatic Gemini CLI install

From:
https://wiki.directo.ee/ - **Directo Help**

Permanent link:
https://wiki.directo.ee/en/ask_directo?rev=1790846617

Last update: **2026/10/01 12:23**