VECTROLINK
TECHNOLOGIES

vMQT-02

Modbus to MQTT Gateway (Modbus RTU & TCP)

API Reference

HTTP JSON API for System Integrators

Firmware version: 0.1.0

Document version: 1.0

November 18, 2026

Confidential — intended for the customer and integrators of this device.

Contents

1 Overview & Conventions **3**
- 1.1 Connecting . . . 3
- 1.2 Configuration Changes Take Effect After a Restart . . . 3
- 1.3 Response Envelope . . . 4
- 1.4 HTTP Status Codes . . . 4
- 1.5 Field Naming and Types . . . 4

2 Authentication & Sessions **6**
- 2.1 Login Flow . . . 6
  - 2.1.1 POST /api/auth/login . . . 6
  - 2.1.2 Using the token . . . 6
  - 2.1.3 POST /api/auth/logout . . . 7
  - 2.1.4 GET /api/auth/session . . . 7
- 2.2 User Accounts . . . 7
  - 2.2.1 GET /api/auth/users . . . 9
  - 2.2.2 POST /api/auth/users – create a user . . . 9
  - 2.2.3 PUT /api/auth/users – edit a user . . . 10
  - 2.2.4 DELETE /api/auth/users . . . 10
  - 2.2.5 POST /api/auth/password – change a password . . . 10

3 Status & System **11**
- 3.1 GET /api/status . . . 11
- 3.2 GET /api/version . . . 12
- 3.3 POST /api/system/restart . . . 13
- 3.4 Configuration backup: GET /api/backup, POST /api/backup . . . 13
- 3.5 POST /api/system/factory_reset . . . 14
- 3.6 POST /api/system/temp_wifi . . . 14
- 3.7 POST /api/system/firmware_update_allowed . . . 15

4 Gateway, RS-485 & Modbus TCP Settings **16**
- 4.1 GET /api/gateway/config . . . 16
  - 4.1.1 How Modbus TCP devices are polled . . . 17
- 4.2 POST /api/gateway/config . . . 17

5 MQTT **19**
- 5.1 Topics and Payloads . . . 19
  - 5.1.1 Point payload . . . 19
  - 5.1.2 Publishing rules . . . 20
  - 5.1.3 Home Assistant discovery . . . 20
- 5.2 GET /api/mqtt/config . . . 20
- 5.3 POST /api/mqtt/config . . . 21
- 5.4 POST /api/mqtt/test . . . 22

5.5 GET /api/mqtt/status . . . 22

6 Points **24**
6.1 Point Fields . . . 24
6.1.1 Rules checked on save . . . 25
6.2 GET /api/points . . . 25
6.3 GET /api/points/values . . . 25
6.4 GET /api/points/csv – export . . . 26
6.5 POST /api/points/csv – replace the table . . . 26
6.5.1 CSV format rules . . . 27

7 Diagnostics **28**
7.1 GET /api/gateway/stats . . . 28
7.1.1 Per-device counters . . . 28
7.2 Point Status . . . 29
7.3 Reading the counters . . . 29

8 Network **30**
8.1 GET /api/network . . . 30
8.2 POST /api/network . . . 30
8.3 GET /api/wifi/scan . . . 31
8.4 Saved Wi-Fi Networks . . . 31
8.4.1 GET /api/wifi/networks . . . 31
8.4.2 POST /api/wifi/networks – add . . . 32
8.4.3 DELETE /api/wifi/networks/{index} . . . 32

9 Enum Reference **33**
9.1 type – Modbus data type . . . 33
9.2 order – byte order of 32-bit values . . . 33
9.3 function – Modbus area . . . 33
9.4 port – Modbus channel . . . 34
9.5 payload . . . 34
9.6 quality . . . 34
9.7 mqtt.state . . . 34
9.8 parity . . . 34
9.9 wifi_mode . . . 34

Chapter 1

Overview & Conventions

Every function available in the vMQT-02's web interface is also available as a JSON HTTP API on the device itself, so you can configure and monitor the gateway from your own tools – a commissioning script that loads the point tables of many gateways, a fleet-management system, or a monitoring dashboard – without going through the web UI or Vectrolink Cloud.

The vMQT-02 has two separate interfaces towards other systems:

- **MQTT** is how the gateway delivers the Modbus values: it connects to a broker and publishes to topics. The topics and payloads are described in Chapter 5 and in the *vMQT-02 User Manual*.
- **This HTTP JSON API** is how the gateway itself is configured and diagnosed: the device name, the RS-485 channels and Modbus TCP devices, the broker connection, the Modbus → MQTT point table, live values and counters.

This document is a precise, example-driven reference for the HTTP API: every endpoint, its exact request and response bodies, and the behaviour you need to know about in practice. It does not cover firmware update (OTA), cloud provisioning or superadmin-only maintenance endpoints – those are internal/administrative and out of scope for this document.

1.1 Connecting

All endpoints are served over HTTPS directly by the device, on whatever IP address it has on your network (Ethernet or Wi-Fi). The device uses a self-signed certificate, so your HTTP client or script needs to either trust that specific certificate or disable certificate verification for this host – for example, `curl -k` for quick testing, or your language's HTTP library equivalent for production code.

```
curl -k https://<device-ip>/api/version
```

`GET /api/version` (Section 3.2) is the one endpoint in this API that needs no session token, so it is the right quick check that you can reach the device at all before dealing with authentication (Chapter 2).

1.2 Configuration Changes Take Effect After a Restart

The gateway builds its Modbus polling plan, its topics and its broker connection once, at start-up. Every endpoint that changes the gateway configuration (Chapters 4, 5 and 6) therefore:

1. validates the request and **saves it to the configuration file** on the device,
2. answers with `"restart_required": true`,

3. keeps running with the *previous* configuration until you call `POST /api/system/restart` (Section 3.3).

Several changes can be saved one after another before a single restart – each save starts from the last saved (not the running) configuration, so a points upload followed by a broker settings change both survive. While a restart is pending, the read endpoints (`GET /api/gateway/config`, `GET /api/mqtt/config`, `GET /api/points`, `GET /api/points/csv`) return the *saved* configuration and the flag `"restart_pending": true`.

A restart takes about 10 seconds. The gateway publishes `offline` to its status topic before it restarts, and `online` and all point values once it is connected to the broker again.

1.3 Response Envelope

Read-only `GET` endpoints that return data return that data directly as a JSON object or array (or CSV text for `GET /api/points/csv`). Write endpoints respond with a small status object:

```
{"status": "ok", "message": "Saved - restart to apply", "
   restart_required": true}
```

or, on failure:

```
{"status": "error", "message": "<human-readable description>"}
```

Always check the HTTP status code first, then the `status` field if present. Errors detected while reading the request itself (empty or oversized body, broken connection) are answered with a plain-text body instead of JSON.

1.4 HTTP Status Codes

Code	Meaning
200	Success.
400	Bad request – invalid JSON or CSV, a value outside its valid range, or an empty/oversized body. The `message` says which field or which CSV line.
401	No `Authorization` header, or the session token is missing, invalid, or expired.
403	Authenticated, but this account isn't allowed to perform this action (user management, see Chapter 2).
404	Unknown URL, or the referenced resource (user, saved Wi-Fi network) doesn't exist.
500	Internal error on the device (e.g. the configuration file could not be written).

1.5 Field Naming and Types

- All JSON keys are `snake_case`.
- Enumerated point fields (data type, byte order, function code) use short upper-case strings such as `"F32"` or `"CDAB"`; the complete tables are in Appendix 9.

- The Modbus channel of a point (`port`) is a number in JSON – 1, 2 for RS-485, 3–10 for Modbus TCP devices 1–8 – and `1`, `2`, `TCP1`…`TCP8` in CSV.
- `uptime` is in milliseconds; other timestamps are Unix epoch seconds.
- Secrets (the broker password, a CA certificate) are write-only: read endpoints say whether one is set, never what it is.

This reference was written directly against the firmware source code, but the API is not formally versioned. If a response includes fields not documented here, ignore unknown fields rather than failing.

Chapter 2

Authentication & Sessions

2.1 Login Flow

1. **Log in** to obtain a session token.
2. **Send the token** on every subsequent request in the `Authorization` header.
3. **Log out** (optional) to invalidate the token early, or just let it expire.

Sessions expire automatically after **1 hour of inactivity**; each successful authenticated request renews the timer. There's no refresh-token mechanism – simply log in again once a request comes back `401`.

2.1.1 POST /api/auth/login

Auth required: none.

```
POST /api/auth/login
Content-Type: application/json

{"username": "user", "password": "user"}
```

Success (200):

```
{"status": "ok", "token": "a1b2c3...64 hex chars", "username": "user"}
```

Errors: 400 (missing username/password), 401 (wrong credentials), 403 (see below).

Optional `access` field. Add `"access": "api"` to the login body if you only want to verify this account has *API access* (rather than *web* access, which is checked by default). This matters because an account can have one without the other – see the `web_access`/`api_access` flags in Section 2.2. If the account lacks the access mode being requested, login fails with `403` even though the password is correct.

2.1.2 Using the token

Attach it as a Bearer token on every request:

```
curl -k https://<device-ip>/api/status \
     -H "Authorization: Bearer a1b2c3...64 hex chars"
```

The device also accepts the raw token without the `Bearer` prefix, but sending it with the prefix is the conventional and recommended form.

The `web_access`/`api_access` flags on an account are only checked at *login* time (which mode you request via the `access` field). Once you hold a valid token, it works identically against every endpoint in this document regardless of those flags – they gate *how you can log in*, not what a given session can subsequently call.

2.1.3 POST /api/auth/logout

Auth required: any authenticated session.

```
POST /api/auth/logout
Authorization: Bearer <token>
```

Success (200): `{"status": "ok"}`. Invalidates the token immediately; no request body needed.

2.1.4 GET /api/auth/session

Auth required: any authenticated session. Use this to check whether a token you're holding is still valid, and to find out what the account can do.

```
GET /api/auth/session
Authorization: Bearer <token>
```

Success (200):

```
{
  "status": "ok",
  "authenticated": true,
  "username": "user",
  "is_admin": true,
  "is_superadmin": false
}
```

Expired/invalid (401):

```
{"status": "error", "message": "Invalid or expired session", "
   authenticated": false}
```

2.2 User Accounts

The device ships with a small, fixed set of accounts you can use directly or extend:

Username	Default password	Web access	API access	Notes
superadmin	Not documented here	Yes	Yes	Full control; hidden from GET /api/auth/users entirely; can't be edited/deleted via the API at all. Reserved for Vectrolink support/administrative use – not intended for integrators, and its credentials are out of scope for this reference.
user	user	Yes	Yes	General-purpose account for day-to-day web UI use.
api	api	No	Yes	Meant for integrations – change its password before relying on it.

Change the default passwords before deploying the device. user/user and api/api are well-known defaults – identical on every device of this model until changed, not device-specific.

You can create up to **5 additional managed accounts** beyond these built-in ones (see the X-Max-Managed-Users / X-Managed-Users-Count response headers on GET /api/auth/users to check how many you have left).

Auth required: any authenticated session for all four endpoints below, but what you can actually do depends on whether you're superadmin:

- **Creating/editing a user with is_admin: true requires superadmin.** A non-superadmin caller can still call POST/PUT, but is_admin in the request body is silently ignored and forced to false – this does *not* fail with an error, it just doesn't grant what you asked for. Check the is_admin field in the response of a follow-up GET if you need to confirm what actually landed.
- **Only superadmin can touch the built-in accounts** – superadmin, cloud, user, and api themselves. A non-superadmin session gets 403 Forbidden ("Cannot manage/delete system account") if it tries to PUT/DELETE any of these four usernames. Creating a user with one of these names always fails anyway (they already exist) regardless of caller.
- Anything else – creating, editing, or deleting one of the up to **5 additional managed accounts** – works for any authenticated session, superadmin or not.

In short: any account can manage plain managed users, but only superadmin can grant admin privileges or touch the four built-in accounts. is_admin itself is otherwise informational – shown back to you in GET /api/auth/users/GET /api/auth/session, but nothing else in this API reads it.

2.2.1 GET /api/auth/users

```
GET /api/auth/users
Authorization: Bearer <admin-token>
```

Success (200) – a JSON array, plus two response headers:

```
X-Max-Managed-Users: 5
X-Managed-Users-Count: 2

[
  {
    "username": "user",
    "is_admin": true,
    "web_access": true,
    "api_access": true,
    "manageable": true,
    "created_at": 1720000000
  },
  {
    "username": "api",
    "is_admin": false,
    "web_access": false,
    "api_access": true,
    "manageable": true,
    "created_at": 1720000001
  }
]
```

created_at is a Unix timestamp. The built-in `superadmin` account never appears in this list under any circumstances. `manageable` reflects what *you*, the caller, are allowed to do with that entry: `true` for any of the 5 additional managed accounts, and for the `user`/`api`/`cloud` built-ins only if you're calling as `superadmin` – for a non-superadmin caller, `user` and `api` still appear in the list (unlike `cloud`, which is hidden entirely unless you're `superadmin`) but with `manageable: false`, since `PUT`/`DELETE` against them will 403.

2.2.2 POST /api/auth/users – create a user

```
POST /api/auth/users
Authorization: Bearer <admin-token>
Content-Type: application/json

{
  "username": "integration1",
  "password": "a-strong-password",
  "is_admin": false,
  "web_access": false,
  "api_access": true
}
```

`is_admin` defaults to `false`, `web_access` and `api_access` both default to `true` if omitted. Sending `"is_admin": true` only has an effect if you're calling as `superadmin` – see the note above.

Success (200): `{"status": "ok"}`. **Errors:** 400 (missing username/password, username already taken, or the 5-account managed-user limit reached).

For a pure machine-to-machine integration account with no interactive web login, set `"web_access": false, "api_access": true` as shown above.

2.2.3 PUT /api/auth/users – edit a user

```
PUT /api/auth/users
Authorization: Bearer <admin-token>
Content-Type: application/json

{
  "username": "integration1",
  "is_admin": false,
  "web_access": false,
  "api_access": true,
  "password": "only-set-this-field-to-change-the-password"
}
```

`username`, `is_admin`, `web_access`, and `api_access` are all required in every request (this call replaces the full permission set, it doesn't patch individual fields). `password` is optional – omit it or send an empty string to leave the password unchanged.

Success (200): `{"status": "ok"}`. **Errors:** 400 (missing required fields), 403 (attempting to edit a protected system account you don't have rights to).

2.2.4 DELETE /api/auth/users

```
DELETE /api/auth/users
Authorization: Bearer <admin-token>
Content-Type: application/json

{"username": "integration1"}
```

Success (200): `{"status": "ok"}`. **Errors:** 400 (missing username), 403 (`"Cannot delete system account"` – protected account), 404 (user not found).

2.2.5 POST /api/auth/password – change a password

Auth required: any authenticated session. There's no separate admin/non-admin distinction on this endpoint – `username` can be any account, not just your own, and any authenticated session can target any account. What actually gates this is `old_password`: the request only succeeds if it matches the *target* account's current password, which is what makes this safe to leave open rather than restricted to `superadmin` like the four endpoints above – proving you already know an account's password is treated as sufficient grounds to change it, regardless of which account you're currently logged in as.

```
POST /api/auth/password
Authorization: Bearer <token>
Content-Type: application/json

{
  "username": "user",
  "old_password": "current-password",
  "new_password": "new-password"
}
```

Success (200): `{"status": "ok"}`. **Errors:** 400 (missing fields), 401 (`old_password` incorrect for the target account).

Chapter 3

Status & System

3.1 GET /api/status

Auth required: any authenticated session.

One call that returns everything a dashboard needs: firmware, uptime, security state, network, the gateway summary, the broker connection and the Modbus counters. Poll it every few seconds for a status view.

```
GET /api/status
Authorization: Bearer <token>
```

Success (200):


```
{
  "status": "running",
  "version": "0.1.0",
  "build_date": "Nov 12 2026 17:56:20",
  "uptime": 845321,
  "last_reset_reason": "poweron",
  "crash_count": 0,
  "security": {
    "secure_boot": true,
    "flash_encryption": "release",
    "uart_download_disabled": true,
    "jtag_disabled": true
  },
  "wifi_connected": false,
  "wifi_ap": false,
  "wifi_sta": false,
  "wifi_mode": "OFF",
  "wifi_ip": "N/A",
  "eth_connected": true,
  "eth_ip": "192.168.1.74",
  "internet_access": true,
  "device_time": "2026-11-13 09:12:44 UTC",
  "config_timestamp": 0,
  "firmware_update_allowed": true,
  "serial_number": 3381941032,
  "gateway": {
    "topic_base": "vectrolink/plant-room/",
    "point_count": 120,
    "max_points": 500,
    "restart_pending": false
  },
  "mqtt": {
    "state": "connected",
```

```
    "broker": "192.168.1.57:8883",
    "tls": true,
    "connected_s": 84210,
    "connects": 1,
    "published": 96112,
    "dropped": 0,
    "last_error": ""
  },
  "modbus": {
    "polls": 48211,
    "poll_errors": 21,
    "writes": 0,
    "write_errors": 0,
    "cycle_ms": 412
  }
}
```

Field	Meaning
`last_reset_reason`	`poweron`, `external_pin`, `software_reset`, `panic`, `interrupt_watchdog`, `task_watchdog`, `other_watchdog`, `brownout`, `sdio` or `unknown`.
`crash_count`	Number of boots that followed a crash (panic, watchdog, brownout).
`security`	What the chip itself enforces, read from its eFuses.
`device_time`	UTC from SNTP, or `"not synced"` without internet access.
`serial_number`	Device serial number as an integer; the label and the web UI show it as 8 hex digits (here `C9945328`).
`gateway.topic_base`	Beginning of every topic of this gateway: `<prefix>/<device name>/`.
`gateway.restart_pending`	A saved configuration is waiting for a restart (Section 1.2).
`mqtt`	Broker connection, the same object as `GET /api/mqtt/status` (Section 5.5).
`modbus.polls`	Modbus read requests sent (one request reads a whole block of neighbouring points).
`modbus.poll_errors`	Of those, requests that failed (timeout, CRC, exception).
`modbus.writes / write_errors`	Modbus writes and the failed ones (0: this firmware only reads).
`modbus.cycle_ms`	Duration of the last complete poll of all points.

`uptime` is in **milliseconds**. `wifi_ip`/`eth_ip` are `"N/A"` when that interface has no address.

3.2 GET /api/version

Auth required: none.

```
GET /api/version
```

Success (200):

```
{"version": "0.1.0", "build_date": "Nov 12 2026 17:56:20"}
```

Use this for a simple "is the device up, and what firmware is it running" check. Everything else needs a session.

3.3 POST /api/system/restart

Auth required: any authenticated session.

Reboots the device – this is how a saved configuration change is applied (Section 1.2).

```
POST /api/system/restart
Authorization: Bearer <token>
```

Success (200):

```
{"status": "ok", "message": "System restarting..."}
```

> The response is sent *before* the device restarts (there is a fixed 500 ms delay to guarantee delivery). Expect the device to be unreachable for about 10 seconds; poll `GET /api/version` until it answers again.

3.4 Configuration backup: GET /api/backup, POST /api/backup

Auth required: any authenticated session.

One JSON file with the whole configuration of the device: device name, poll cycle, RS-485 channels, Modbus TCP devices, MQTT settings (without password and CA certificate) and the point table. The same file format and the same two endpoints exist on every Vectrolink product; a file of another product is refused.

```
GET /api/backup
Authorization: Bearer <token>
```

Success (200): the file, with `Content-Disposition: attachment; filename="vMQT-02_<serial>_config_<date>.json"`:

```
{
  "vectrolink_backup": 1,
  "product": "vMQT-02",
  "firmware": "0.1.0",
  "serial": "C9945328",
  "exported": "2026-11-13T15:20:00Z",
  "config": {
    "gateway": {"device_name": "plant-room", "poll_period_ms": 1000, "
       ports": [...], "tcp": [...]},
    "mqtt": {"enabled": true, "host": "192.168.1.57", "port": 8883, "
       tls": true, ...},
    "points_csv": "name,description,unit,enabled,port,slave,..."
  }
}
```

Not part of a backup: passwords and certificates, user accounts, network settings and the update consent – the file may be sent by e-mail, and these belong to the individual device.

```
POST /api/backup
Authorization: Bearer <token>
Content-Type: application/json
```

```
<the file as downloaded>
```

The whole file is checked before anything is saved; on an error nothing changes. Keys that are missing in `config` keep the device's current values. The restored configuration is applied after a restart (`POST /api/system/restart`).

Success (200): `{"status": "ok", "message": "Configuration restored - restart to apply", "restart_required": true}`

Errors (400), for example:

```
{"status": "error", "message": "This backup is from vMBT-02, this
   device is vMQT-02"}
{"status": "error", "message": "Backup format 2 is newer than this
   firmware - update the firmware first"}
{"status": "error", "message": "Not a Vectrolink configuration backup"}
```

3.5 POST /api/system/factory_reset

Auth required: any authenticated session.

Returns the device to its delivery state and restarts – the same as holding the device's button for 10 seconds: network settings, user accounts (passwords back to defaults), crash statistics, the point table and the gateway, Modbus TCP and MQTT broker settings are erased. The device identity (serial number), its TLS certificate and its Vectrolink Cloud provisioning are kept.

> Export the point table first (`GET /api/points/csv`) if you may need it again.

```
POST /api/system/factory_reset
Authorization: Bearer <token>
```

Success (200): `{"status": "ok", "message": "Factory reset starting - device is restarting..."}`

> After a factory reset the device is back on DHCP with default passwords. Do not call this over a remote or tunneled connection unless someone on site can reach the device afterwards.

3.6 POST /api/system/temp_wifi

Auth required: any authenticated session.

Opens a temporary (10-minute) Wi-Fi window, exactly like a short (`sta`) or double (`ap`) press of the device's button: `"sta"` connects to the saved Wi-Fi networks, `"ap"` opens the device's own service access point.

```
POST /api/system/temp_wifi
Authorization: Bearer <token>
Content-Type: application/json

{"mode": "ap"}
```

Success (200): `{"status": "ok", "message": "Temporary WiFi started for 10 minutes"}`. **Errors:** 400 (`mode` is not `"sta"` or `"ap"`), 500 (Wi-Fi could not be started).

3.7 POST /api/system/firmware_update_allowed

Auth required: any authenticated session.

Turns automatic firmware updates via Vectrolink Cloud on or off for this device (default: on). When off, the device keeps reporting to the update server but does not download or apply any update until the flag is turned back on. Manual firmware upload in the web UI is not affected.

```
POST /api/system/firmware_update_allowed
Authorization: Bearer <token>
Content-Type: application/json

{"allowed": false}
```

Success (200): `{"status": "ok", "firmware_update_allowed": false}`. **Errors:** 400 (missing `allowed`), 500 (`"Failed to update setting"`). The current value is the `firmware_update_allowed` field of `GET /api/status`.

Chapter 4

Gateway, RS-485 & Modbus TCP Settings

These settings define the device name used in the topics, the poll cycle, the two RS-485 channels and the Modbus TCP devices. Changes are saved immediately and applied after a restart (Section 1.2).

4.1 GET /api/gateway/config

Auth required: any authenticated session.

```
GET /api/gateway/config
Authorization: Bearer <token>
```

Success (200):

```
{
  "device_name": "plant-room",
  "topic_base": "vectrolink/plant-room/",
  "poll_period_ms": 1000,
  "ports": [
    {"baud": 9600, "parity": 0, "stop_bits": 1, "timeout_ms": 300},
    {"baud": 19200, "parity": 1, "stop_bits": 1, "timeout_ms": 300}
  ],
  "tcp": [
    {"host": "192.168.10.21", "port": 502, "timeout_ms": 1000},
    {"host": "inverter-2.local", "port": 502, "timeout_ms": 1500},
    {"host": "", "port": 502, "timeout_ms": 1000},
    {"host": "", "port": 502, "timeout_ms": 1000},
    {"host": "", "port": 502, "timeout_ms": 1000},
    {"host": "", "port": 502, "timeout_ms": 1000},
    {"host": "", "port": 502, "timeout_ms": 1000},
    {"host": "", "port": 502, "timeout_ms": 1000}
  ],
  "restart_pending": false
}
```

Field	Meaning
`device_name`	Topic level after the prefix, max 31 characters, without `/`, `+` and `#`. Empty = `vMQT-02_<serial>` (serial as 8 hex digits).

Field	Meaning
topic_base	Read-only: <topic prefix>/<device name>/, the beginning of every topic of the gateway (prefix: Chapter 5).
poll_period_ms	Target duration of one Modbus poll cycle (100–60000). If reading all points takes longer, the next cycle starts immediately.
ports[0], ports[1]	Modbus RTU settings of RS-485-1 and RS-485-2.
ports[].baud	1200–115200 bit/s.
ports[].parity	0 = none, 1 = even, 2 = odd.
ports[].stop_bits	1 or 2. Data bits are always 8.
ports[].timeout_ms	Modbus response timeout, 50–5000 ms.
tcp[0] ... tcp[7]	Modbus TCP devices 1–8; points refer to them with port 3–10. Always 8 entries.
tcp[].host	IPv4 address or host name (letters, digits, . and -, max 39 characters). Empty = not used.
tcp[].port	TCP port, 1–65535, normally 502.
tcp[].timeout_ms	Connection and response timeout, 100–10000 ms.
restart_pending	The values shown are saved but not yet running (Section 1.2).

4.1.1 How Modbus TCP devices are polled

- The gateway keeps one TCP connection open per device and sends all requests over it, one at a time. Many devices allow only a few connections; the gateway never uses more than one.
- Requests carry an MBAP transaction identifier; a late answer to an earlier request is recognised and skipped.
- If the device closes an idle connection, the gateway reconnects and repeats the request once.
- If the device cannot be reached within timeout_ms, its points are marked invalid and no new connection is attempted for 5 seconds, so an unreachable device does not slow down the poll cycle of the others.
- A host name is resolved at each new connection (DNS needed, see Chapter 8).

4.2 POST /api/gateway/config

Auth required: any authenticated session.

Send only the fields you want to change; the others keep their saved values. Inside ports and tcp the array index selects the channel or device, and again only the given fields change – to set TCP device 3 only, send three entries and leave the first two empty ({}).

```
POST /api/gateway/config
Authorization: Bearer <token>
Content-Type: application/json

{
  "device_name": "plant-room",
  "poll_period_ms": 2000,
  "ports": [{"baud": 19200, "parity": 1}],
```

```
  "tcp": [{}, {}, {"host": "192.168.10.40", "port": 502, "timeout_ms":
     800}]
}
```

To stop using a TCP device, set its `host` to `""`; points that refer to it then show a communication failure.

Success (200):

```
{"status": "ok", "message": "Saved - restart to apply", "
   restart_required": true}
```

Errors (400):

- `{"status": "error", "message": "Invalid JSON"}`
- `{"status": "error", "message": "Invalid value: tcp.host"}` – the name of the first field outside its range (`device_name`, `poll_period_ms`, `ports.baud`, `ports.parity`, `ports.stop_bits`, `ports.timeout_ms`, `tcp.host`, `tcp.port`, `tcp.timeout_ms`). Nothing is saved when any field is invalid.
- Plain-text `Body empty or too large` – the body is limited to 4 KB.

Error (500): `{"status": "error", "message": "Saving failed"}`.

Chapter 5

MQTT

The broker connection, the topic layout, the payload format and Home Assistant discovery. Like the other settings, changes are saved immediately and applied after a restart (Section 1.2); `POST /api/mqtt/test` tries a connection right away.

5.1 Topics and Payloads

Topic	Payload
`<topic_prefix>/<device>/<point name>`	Point value, QoS 0, retained if the point's `retain` is 1
`<topic_prefix>/<device>/status`	`online` / `offline`, QoS 1, retained; `offline` is the last will
`<ha_prefix>/<component>/<node>/<point>/config`	Home Assistant discovery, QoS 1, retained

`<device>` is `device_name` of the gateway settings (Chapter 4); the full beginning of the topics is `topic_base` in `GET /api/gateway/config` and `GET /api/status`, and every point of `GET /api/points` carries its complete `topic`.

5.1.1 Point payload

With `"payload": "json"`:

```
{"value":230.5,"unit":"V","quality":"good","ts":1794573479}
{"value":null,"unit":"V","quality":"comm_failure","ts":1794573491}
```

Key	Meaning
`value`	Engineering value, raw × `scale` + `offset`, up to 7 significant digits; 0/1 for `BIT`; `null` on a failed read or a float register holding NaN/infinity.
`unit`	The point's `unit`; left out when empty.
`quality`	`"good"` or `"comm_failure"` (no answer, CRC error, Modbus exception, TCP device unreachable).
`ts`	Time of the reading, Unix epoch seconds; left out while the device time is not synchronised by SNTP.

With `"payload": "value"` the message is the bare number, e.g. `230.5`, and nothing is published for a point while its reads fail.

5.1.2 Publishing rules

A point is published (1) every `period` seconds of the point, or `publish_period_s` when the point's period is 0; (2) after a poll cycle in which its value differs from the last published value by at least `deadband` (any change when 0); (3) when its quality changes; (4) once after each connection to the broker. Messages that cannot be sent while the broker is unreachable are counted as `dropped`; after the reconnection the current values are published.

5.1.3 Home Assistant discovery

With `ha_discovery` on, one retained configuration message is published per point: `<component>` is `sensor` for numeric points and `binary_sensor` for `BIT` points, `<node>` is `vmqt02_<serial>` (8 lower-case hex digits). The message names the state topic, the availability topic (`.../status`), the unit, a device class and state class derived from the unit, a `value_template` for the JSON payload, and a `device` block (name, manufacturer, model, firmware version, serial number) that groups all points of the gateway into one Home Assistant device. Configuration messages of points that are no longer in the table, or of all points when discovery is switched off, are cleared with an empty retained message after each connection: the gateway listens to its own discovery topics for a few seconds and clears every retained configuration that does not belong to the current table. Discovery messages of other gateways are not touched.

5.2 GET /api/mqtt/config

Auth required: any authenticated session.

```
GET /api/mqtt/config
Authorization: Bearer <token>
```

Success (200):

```
{
  "enabled": true,
  "host": "192.168.1.57",
  "port": 8883,
  "tls": true,
  "ca_cert_set": true,
  "username": "plant-room-gw",
  "password_set": true,
  "client_id": "",
  "client_id_effective": "vMQT-02_C9945328",
  "keepalive_s": 60,
  "topic_prefix": "vectrolink",
  "payload": "json",
  "publish_period_s": 60,
  "ha_discovery": true,
  "ha_prefix": "homeassistant",
  "restart_pending": false
}
```

Field	Meaning
`enabled`	`false` (default): the gateway polls the Modbus devices but does not connect to a broker.
`host`	Broker IPv4 address or host name, max 63 characters.
`port`	1–65535; default 1883. Use 8883 with TLS.

Field	Meaning
`tls`	`true`: MQTT over TLS 1.2 with verification of the broker certificate (chain and host name/IP address). There is no option to skip the verification.
`ca_cert_set`	A customer CA certificate is stored. Without it, the broker certificate must be issued by a public CA from the built-in list.
`username`	Broker user name, max 31 characters; empty = connect without credentials.
`password_set`	A broker password is stored. The password itself is never returned.
`client_id`	MQTT client identifier, max 31 characters; empty = default.
`client_id_effective`	The identifier used: `client_id`, or `vMQT-02_<serial>`.
`keepalive_s`	MQTT keepalive, 10–3600 s, default 60.
`topic_prefix`	First topic level(s), max 31 characters, default `vectrolink`; may contain `/` but not `+`, `#`, or a leading/trailing `/`.
`payload`	`"json"` (default) or `"value"`.
`publish_period_s`	Default period for points with `period` 0, 1–65535 s, default 60.
`ha_discovery`	Publish Home Assistant discovery messages, default `true`.
`ha_prefix`	Discovery prefix, max 23 characters, default `homeassistant`.
`restart_pending`	The values shown are saved but not yet running.

5.3 POST /api/mqtt/config

Auth required: any authenticated session.

Send only the fields you want to change. The two secrets are set like this:

- `"password"`: a new password; `""` removes the stored one; leave the key out to keep it.
- `"ca_cert"`: a CA certificate in PEM format (one or more `BEGIN CERTIFICATE` blocks, max 4 KB, `\n` line breaks in JSON); `""` removes it; leave the key out to keep it.

```
POST /api/mqtt/config
Authorization: Bearer <token>
Content-Type: application/json

{
  "enabled": true,
  "host": "192.168.1.57",
  "port": 8883,
  "tls": true,
  "ca_cert": "-----BEGIN CERTIFICATE-----\nMIIDBTCCAe2gAwIBAgIU...\n
     -----END CERTIFICATE-----\n",
  "username": "plant-room-gw",
  "password": "s3cret-broker-pass",
```

```
  "payload": "json",
  "ha_discovery": true
}
```

Success (200):

```
{"status": "ok", "message": "Saved - restart to apply", "
   restart_required": true}
```

Errors (400): `"Invalid JSON"`, or `"Invalid value: <field>"` for the first field outside its range (e.g. `host` with a space, `topic_prefix` with `#`, `ca_cert` that is not a PEM certificate). Nothing is saved when any field is invalid. Plain-text `Body empty or too large` above 8 KB. **Error (500):** `"Saving failed"`.

5.4 POST /api/mqtt/test

Auth required: any authenticated session.

Connects to a broker once and disconnects again, without publishing anything – to check address, credentials and certificate before saving. The body has the same fields as `POST /api/mqtt/config`; fields that are left out are taken from the saved configuration (so the stored password is used if `password` is not sent). The call takes at most about 10 seconds.

```
POST /api/mqtt/test
Authorization: Bearer <token>
Content-Type: application/json

{"host": "broker.example.com", "port": 8883, "tls": true, "username": "
   gw1"}
```

Success (200): `{"status": "ok", "message": "Connected to broker.example.com:8883 (TLS)"}`

Failure (200) – the test ran, the connection did not succeed:

```
{"status": "error", "message": "not authorised"}
```

Possible messages: `cannot resolve host`, `host not reachable`, `connection refused`, `timeout`, `certificate verify failed`, `TLS handshake failed`, `bad user name or password`, `not authorised`, `identifier rejected`, `server unavailable`, `connection closed`. The test connects with the client identifier followed by `-test`, so it does not disconnect the gateway's running connection. **Errors (400)** as for `POST /api/mqtt/config`.

5.5 GET /api/mqtt/status

Auth required: any authenticated session.

```
GET /api/mqtt/status
Authorization: Bearer <token>
```

Success (200):

```
{
  "state": "connected",
  "broker": "192.168.1.57:8883",
  "tls": true,
  "connected_s": 84210,
  "connects": 1,
  "published": 96112,
  "dropped": 0,
```

```
  "last_error": ""
}
```

Field	Meaning
`state`	`disabled` (MQTT off), `connecting`, `connected`, or `disconnected` (waiting before the next attempt).
`broker`	`host:port` in use.
`connected_s`	Seconds since the current connection was established; 0 when not connected.
`connects`	Successful connections since start-up. More than 1 means the connection was lost in between.
`published`	Messages handed to the broker since start-up (point values, status, discovery).
`dropped`	Point messages not sent because the broker was not connected.
`last_error`	Reason of the last failed connection or disconnection, same wording as `POST /api/mqtt/test`; empty if none.

After a lost connection the gateway reconnects automatically, waiting 1 s, 2 s, 4 s … up to 60 s between attempts.

Chapter 6

Points

A *point* links one value in a Modbus device to one MQTT topic: for example holding registers 4–5 of slave 3 on RS-485-1, a 32-bit float, are published as `vectrolink/plant-room/supply_temp`. The gateway polls the Modbus side and publishes the engineering value according to the point's publishing settings (Section 5.1).

The gateway holds up to **500 points**. The point table is read as JSON (with live values) and written as CSV – the same file format the web UI uses for import/export and that you can edit in Excel.

6.1 Point Fields

Field	Meaning
`name`	Last topic level, max 31 characters, unique in the table, without `/`, `+`, `#`. Letters, digits, `_` and `-` recommended.
`description`	Optional, max 47 characters; entity name in Home Assistant.
`unit`	Unit text, max 11 characters (UTF-8, e.g. `°C`); in the JSON payload and for Home Assistant.
`topic`	Read-only (JSON only): the full topic of the point.
`enabled`	1 = polled and published, 0 = neither.
`port`	Modbus channel: 1 = RS-485-1, 2 = RS-485-2, 3–10 = Modbus TCP device 1–8 (CSV: `TCP1`–`TCP8`).
`slave`	RS-485: Modbus slave address 1–247. Modbus TCP: unit identifier 0–255.
`function`	Modbus area: 1 = coils (FC01), 2 = discrete inputs (FC02), 3 = holding registers (FC03), 4 = input registers (FC04).
`address`	0-based register or bit address (protocol address, not the "40001" notation: holding register 40001 is `address 0`).
`type`	Data type: `U16`, `S16`, `U32`, `S32`, `F32`, `BIT` (Appendix 9).
`order`	Byte order of 32-bit types: `ABCD`, `CDAB`, `BADC`, `DCBA`. Ignored for 16-bit types and bits.
`bit`	For `BIT` on function 3 or 4: which bit (0–15) of the register.
`scale, offset`	Published value = raw value × `scale` + `offset`. `scale` must not be 0.
`period`	Seconds between periodic publications, 0–65535; 0 = `publish_period_s` of the MQTT settings.
`deadband`	Publish as soon as the value changed by at least this much since the last publication; 0 = any change.
`retain`	1 = published with the MQTT retain flag.
`writable`	Reserved for writing from MQTT; keep 0.

6.1.1 Rules checked on save

- `name` is not empty, contains no `/`, `+`, `#`, and is used only once.
- Functions 1 and 2 (coils, discrete inputs) need type `BIT`.
- `port` is 1–10; on RS-485 `slave` is 1–247.
- `scale` is not 0 (except for `BIT`); `bit` is 0–15.

6.2 GET /api/points

Auth required: any authenticated session.

The whole point table with topics and the latest polled values.

```
GET /api/points
Authorization: Bearer <token>
```

Success (200):

```
{
  "max_points": 500,
  "restart_pending": false,
  "points": [
    {"name": "supply_temp", "description": "AHU-1 supply air", "unit":
       "°C",
     "topic": "vectrolink/plant-room/supply_temp",
     "enabled": 1, "port": 1, "slave": 3, "function": 3, "address": 4,
     "type": "F32", "order": "CDAB", "bit": 0, "scale": 1, "offset": 0,
     "period": 0, "deadband": 0.2, "retain": 0, "writable": 0,
     "value": 21.4, "valid": true},
    {"name": "pv_power", "description": "Inverter 1", "unit": "kW",
     "topic": "vectrolink/plant-room/pv_power",
     "enabled": 1, "port": 3, "slave": 1, "function": 3, "address":
       30775,
     "type": "S32", "order": "ABCD", "bit": 0, "scale": 0.001, "offset
       ": 0,
     "period": 10, "deadband": 0.05, "retain": 1, "writable": 0,
     "value": null, "valid": false}
  ]
}
```

`value` is the engineering value (after scale/offset; 0/1 for bits), `valid` is `false` when the last Modbus read of that point failed (then `value` is `null`). A float register holding NaN or infinity is returned as `"value": null` with `"valid": true`.

> While a restart is pending (Section 1.2), this endpoint returns the *saved* table, and the `value`/`valid` fields are left out – the running values belong to the old table.

6.3 GET /api/points/values

Auth required: any authenticated session.

Only the live values, in table order – a compact call for frequent polling:

```
GET /api/points/values
Authorization: Bearer <token>
```

Success (200): an array of `[value, valid]` pairs, `valid` as 1/0:

```
[[21.4,1],[0,0],[1,1]]
```

These always belong to the *running* table; match them by position to a `GET /api/points` response taken while no restart was pending.

6.4 GET /api/points/csv – export

Auth required: any authenticated session.

```
GET /api/points/csv
GET /api/points/csv?sep=semicolon
Authorization: Bearer <token>
```

Returns the point table as a CSV file (`text/csv`, UTF-8 with a byte-order mark, `Content-Disposition: attachment; filename="vmqt-02-points.csv"`). With `?sep=semicolon` the separator is `;` and numbers use a decimal comma – the format Excel expects in Polish, German and most other European locales.

```
name,description,unit,enabled,port,slave,function,address,type,order,
    bit,scale,offset,period,deadband,retain,writable
"supply_temp","AHU-1 supply air","°C",1,1,3,3,4,F32,CDAB
    ,0,1,0,0,0.2,0,0
"pv_power","Inverter 1","kW",1,TCP1,1,3,30775,S32,ABCD
    ,0,0.001,0,10,0.05,1,0
```

6.5 POST /api/points/csv – replace the table

Auth required: any authenticated session.

Replaces the *whole* point table with the CSV in the request body (max 128 KB). The table is validated completely before anything is saved: one bad line and nothing changes.

```
POST /api/points/csv
Authorization: Bearer <token>
Content-Type: text/csv

name;unit;port;slave;function;address;type;order;scale;deadband
"supply_temp";°C;1;3;3;4;F32;CDAB;1;0,2
"pv_power";kW;TCP1;1;3;30775;S32;;0,001;0,05
```

Success (200):

```
{"status": "ok", "message": "2 points saved - restart to apply", "
    restart_required": true}
```

Errors (400), always with the line number of the file:

```
{"status": "error", "message": "line 3: bad type 'FLOAT'"}
{"status": "error", "message": "line 4: bad port 'TCP9' (1, 2 or TCP1..
    TCP8)"}
{"status": "error", "message": "line 7 (pump_run): FC01/FC02 need type
    BIT"}
{"status": "error", "message": "line 9: name 'pv_power' used twice (it
    is the MQTT topic)"}
```

Plain-text `Body empty or too large` for an empty body or one above 128 KB.

6.5.1 CSV format rules

- The first non-empty line is the header. It names the columns, in any order, case-insensitive. `name` and `address` are required; any other column may be left out.
- Missing columns or empty cells take defaults: `enabled 1`, `port 1`, `slave 1`, `function 3`, `type U16` (`BIT` for functions 1 and 2), `order ABCD`, `bit 0`, `scale 1`, `offset 0`, `period 0`, `deadband 0`, `retain 0`, `writable 0`, empty `description` and `unit`.
- The separator is taken from the header line: `;` if it contains one, otherwise `,`. With `;`, a decimal comma is accepted (`0,1`).
- Text may be quoted with `"`; a quote inside quoted text is written as `""`. A UTF-8 byte-order mark at the start is ignored.
- Empty lines and lines starting with `#` are ignored. Lines may end with CRLF or LF.
- At most 500 data lines.

A convenient way to commission a site: export the CSV from one gateway, adapt it in Excel (slave addresses, names), and upload it to the next gateway with `POST /api/points/csv` followed by `POST /api/system/restart`.

Chapter 7

Diagnostics

7.1 GET /api/gateway/stats

Auth required: any authenticated session.

Counters of the Modbus side, in total and per device. The extttmodbus object is also part of `GET /api/status`; this endpoint returns only them, for frequent polling.

```
GET /api/gateway/stats
Authorization: Bearer <token>
```

Success (200):

```
{
  "modbus": {"polls": 48211, "poll_errors": 21, "writes": 0, "
     write_errors": 0, "cycle_ms": 412},
  "slaves": [ ... ]
}
```

All counters count from the last restart. The `slaves` array is described in Section 7.1.1.

Field	Meaning
`modbus.polls`	Modbus read requests. Neighbouring points of one slave are read together (up to 125 registers per request), so this grows by the number of blocks per cycle, not the number of points.
`modbus.poll_errors`	Read requests that failed: no answer within `timeout_ms`, CRC error or Modbus exception.
`modbus.writes`	Modbus writes (0: this firmware only reads).
`modbus.write_errors`	Of those, the failed ones.
`modbus.cycle_ms`	Duration of the last complete poll of all points.

7.1.1 Per-device counters

The `slaves` array has one entry per Modbus device used by the point table – RS-485 channel + slave address, or Modbus TCP device + unit identifier – in channel order.

```
"slaves": [
  {"port": 1, "slave": 3, "last_ok": true, "requests": 9120, "ok":
     9118,
   "timeouts": 2, "crc_errors": 0, "other_errors": 0,
   "last_ms": 41, "avg_ms": 43, "max_ms": 118},
  {"port": 3, "slave": 1, "last_ok": false, "requests": 300, "ok": 0,
   "timeouts": 300, "crc_errors": 0, "other_errors": 0,
   "last_ms": 0, "avg_ms": 0, "max_ms": 0}
]
```

Field	Meaning
`port, slave`	Channel (1, 2 = RS-485; 3–10 = Modbus TCP device 1–8) and slave address or unit identifier.
`last_ok`	The last request to this device was answered correctly. `false` = the device does not answer right now.
`requests / ok`	Requests sent (reads and writes) / answered correctly.
`timeouts`	No answer within `timeout_ms`: device off, wrong address, baud rate or wiring; for Modbus TCP also a failed connection.
`crc_errors`	RS-485: answer with a wrong checksum – noise, missing termination, two devices with the same address. Always 0 for Modbus TCP.
`other_errors`	Modbus exception (e.g. register does not exist) or a malformed answer.
`last_ms, avg_ms, max_ms`	Response time of successful requests – from sending the request to a complete answer, including the bus turnaround.

The web interface shows the same table on the Dashboard (*Modbus Devices*).

7.2 Point Status

Per point, the communication state is the `valid` field of `GET /api/points` and `GET /api/points/values` (Chapter 6). On MQTT the same state is the `quality` of the JSON payload: `good` while the Modbus reads succeed, `comm_failure` after a failed read (Section 5.1).

7.3 Reading the counters

- `poll_errors` **grows by one per cycle** – one block (one slave) is not answering. Look for points with `valid: false`: they share port and slave.
- `poll_errors` **grows with every request** – the whole channel is silent: wiring, A/B swapped, baud rate or parity, or the devices are not powered.
- `cycle_ms` **is much larger than** `poll_period_ms` – too many slow slaves or many timeouts; lower `timeout_ms` or remove points of devices that are not installed. An unreachable Modbus TCP device costs at most one connection timeout every 5 seconds.

Chapter 8

Network

8.1 GET /api/network

Auth required: any authenticated session for the response in general; `wifi_password` specifically is only populated for the `superadmin` account.

```
GET /api/network
Authorization: Bearer <token>
```

```
{
  "wifi_mode": 2,
  "wifi_ssid": "Factory-Floor-2G",
  "wifi_password": "",
  "wifi_enabled": true,
  "wifi_dhcp_enabled": true,
  "wifi_static_ip": "",
  "wifi_netmask": "",
  "wifi_gateway": "",
  "wifi_dns1": "",
  "wifi_dns2": "",
  "eth_dhcp_enabled": false,
  "eth_static_ip": "192.168.10.111",
  "eth_netmask": "255.255.255.0",
  "eth_gateway": "192.168.10.1",
  "eth_dns1": "8.8.8.8",
  "eth_dns2": ""
}
```

`wifi_mode` is an integer: 0 = AP only, 1 = STA only, 2 = AP+STA. Ethernet has no on/off flag – it's always active; only its IP configuration is here.

> For customer accounts `wifi_password` is always empty (`""`) in this response: the configured Wi-Fi password is write-only through the API. To change it, send the new value with `POST /api/network`.

8.2 POST /api/network

Auth required: any authenticated session.

```
POST /api/network
Authorization: Bearer <token>
Content-Type: application/json

{
  "wifi_mode": 2,
```

```
  "wifi_enabled": true,
  "wifi_ssid": "Factory-Floor-2G",
  "wifi_password": "new-password",
  "wifi_dhcp_enabled": true,
  "eth_dhcp_enabled": false,
  "eth_static_ip": "192.168.10.111",
  "eth_netmask": "255.255.255.0",
  "eth_gateway": "192.168.10.1",
  "eth_dns1": "8.8.8.8"
}
```

Success (200): `{"status": "ok", "message": "Network config saved"}`. **Errors:** 500 (`"Failed to save network config"`).

This call **loads the existing network configuration first** and only overwrites the fields you include – so a partial body (e.g. just `{"eth_static_ip": "..."}`) is safe and won't blank out your Wi-Fi settings. The `saved_networks` list (Section 8.4) is always preserved regardless of what you send here. Two fields default to `true` whenever they're entirely absent from the body, even on an update: `wifi_dhcp_enabled` and `eth_dhcp_enabled`.

Changing network settings (especially `wifi_mode`, static IP settings, or disabling DHCP with an incorrect address) can make the device unreachable at its previous address. Confirm you have another way to reach the device (serial console, or the AP-mode fallback network) before applying a network change you're not certain about, particularly over a remote/tunneled connection.

8.3 GET /api/wifi/scan

Auth required: any authenticated session.

Triggers a live Wi-Fi scan (blocking – this call takes a few seconds) and returns nearby access points.

```
GET /api/wifi/scan
Authorization: Bearer <token>
```

```
[
  {"ssid": "Factory-Floor-2G", "rssi": -52, "auth": 3, "channel": 6},
  {"ssid": "Guest-WiFi", "rssi": -71, "auth": 0, "channel": 11}
]
```

`rssi` is signal strength in dBm (higher/less negative is stronger). `auth` is the raw ESP-IDF `wifi_auth_mode_t` value (0 = open, 3 = WPA2-PSK, etc.) – this reference doesn't reproduce the full ESP-IDF enum; treat `0` as open and any non-zero value as secured if you don't need the exact scheme. Up to 20 access points are returned.

8.4 Saved Wi-Fi Networks

The device can remember up to **5 Wi-Fi networks** with a priority order, independent of the single active `wifi_ssid`/`wifi_password` pair in `GET /api/network`.

8.4.1 GET /api/wifi/networks

```
GET /api/wifi/networks
Authorization: Bearer <token>
```

```
[
  {"ssid": "Factory-Floor-2G", "priority": 0},
  {"ssid": "Backup-Router", "priority": 1}
]
```

Unlike `GET /api/network`, saved networks' passwords are **not** included in this response – only `ssid` and `priority` (0 = highest priority). An item's position in the array is its index, used by the delete call below.

8.4.2 POST /api/wifi/networks – add

```
POST /api/wifi/networks
Authorization: Bearer <token>
Content-Type: application/json

{"ssid": "Backup-Router", "password": "backup-password", "priority": 1}
```

`password` defaults to an empty string (open network) if omitted; `priority` defaults to `0`.

Success (200): `{"status": "ok"}`. **Errors:** 400 (missing `ssid`), 500 with `{"status": "error", "error": "Maximum networks reached"}` once you already have 5 saved, or `{"status": "error", "error": "Failed to add network"}` on any other failure.

Note the error key here is `error`, not `message` – inconsistent with most other endpoints in this document (see Chapter 1's general envelope note). Check for either key defensively if you centralize error handling across endpoints.

8.4.3 DELETE /api/wifi/networks/{index}

```
DELETE /api/wifi/networks/1
Authorization: Bearer <token>
```

The index is part of the URL path, not the body. **Success (200):** `{"status": "ok"}`. **Errors:** 500 (`"Failed to remove network"` – out-of-range index).

Chapter 9

Enum Reference

9.1 type – Modbus data type

Value	Registers	Meaning
U16	1	Unsigned 16-bit, 0–65535.
S16	1	Signed 16-bit (two's complement), −32768–32767.
U32	2	Unsigned 32-bit, byte order per order.
S32	2	Signed 32-bit, byte order per order.
F32	2	IEEE 754 single-precision float, byte order per order.
BIT	1 bit	A coil or discrete input (functions 1, 2), or bit bit of a register (functions 3, 4).

9.2 order – byte order of 32-bit values

The 32-bit value is written as bytes A B C D, A being the most significant. The order says how these bytes arrive in the two registers R0 (lower address) and R1:

Value	R0 R1	Typical use
ABCD	AB CD	Big-endian, the Modbus standard (most meters).
CDAB	CD AB	Word swap – low word first (common for floats in PLCs and some meters).
BADC	BA DC	Byte swap within each register.
DCBA	DC BA	Little-endian.

Example: the float 1500.5 is 0x44BB9000. With ABCD the device sends R0=0x44BB, R1=0x9000; with CDAB it sends R0=0x9000, R1=0x44BB. If a value looks absurd (e.g. 10^{-38} or 10^{30}), try the other orders.

9.3 function – Modbus area

Value	Read with	Write (reserved)
1	FC01 Read Coils	FC05 Write Single Coil
2	FC02 Read Discrete Inputs	read-only
3	FC03 Read Holding Registers	FC06 / FC16; register bit: FC03 + FC06
4	FC04 Read Input Registers	read-only

9.4 port – Modbus channel

JSON	CSV	Channel
1	1	RS-485-1, Modbus RTU
2	2	RS-485-2, Modbus RTU
3 ... 10	TCP1 ... TCP8	Modbus TCP device 1–8 (tcp[0]–tcp[7] of the gateway settings)

9.5 payload

"json" = {"value":..., "unit":..., "quality":..., "ts":...}, "value" = the bare number (Section 5.1).

9.6 quality

"good" = read successfully in the last poll cycle, "comm_failure" = the last read failed.

9.7 mqtt.state

disabled, connecting, connected, disconnected (Section 5.5).

9.8 parity

0 = none, 1 = even, 2 = odd.

9.9 wifi_mode

0 = access point only, 1 = station only, 2 = access point + station (see Chapter 8).