Wise Timetable

REST API — Developer & Integration Manual

`Server: https://wise-tt.com`

API version 2.0 · Manual revision 1.5 · August 2026

Wise Technologies — wise-t.com

Contents

1. Introduction

1.1 About Wise Timetable

Wise Timetable is a professional academic scheduling system developed by Wise Technologies. It is used by universities, faculties and schools worldwide to plan and publish course timetables, room bookings, exams and other academic events. Each institution maintains its schedule with the Wise Timetable desktop application; the published timetable is then made available to students, lecturers and third-party systems through the web application, the mobile application and the REST API described in this manual.

1.2 About this manual

This manual is written for software developers and integration partners who want to read timetable data from the Wise Timetable server programmatically: mobile applications, information screens, campus portals, room-occupancy dashboards, calendar synchronisation tools and similar integrations.

It documents the complete REST API served at `https://wise-tt.com`, from authentication to retrieving fully populated schedules. Every endpoint is described with its parameters, an example request and a realistic example response. Complete, ready-to-run integration examples in curl, Python, JavaScript and C are provided in chapter 7.

1.3 The API at a glance

The API is read-mostly: every endpoint that returns timetable data is an HTTP GET request over HTTPS that returns JSON. Two endpoints accept data instead — POST /v2/eventlinks (chapter 11) and POST /v2/student-registration (chapter 12) — and both are off unless your account is explicitly permitted to use them. There is nothing to install on your side and no SDK is required — any HTTP client in any language can consume it.

Two API surfaces are served by the same service:

Surface	Base URL	Purpose
API v2 (recommended)	`https://wise-tt.com/api/v2/`	The modern surface. Complete event objects (rooms, lecturers, groups always filled in), holidays, notifications as structured objects, meaningful error messages, ETag caching. Use this for all new integrations.
Legacy-compatible	`https://wise-tt.com/api/`	A faithful reproduction of the original Wise Timetable REST API, kept for existing clients. Frozen: its paths and response shapes will not change. Documented in chapter 6.

A typical integration needs only four steps: obtain a token from `/v2/login`, discover the school's code lists (groups, lecturers, rooms or courses), request the schedule for a date range with `/v2/schedule`, and render the returned events. The walkthrough in chapter 2 does exactly this.

1.4 Obtaining access credentials

Access to the API requires an API account (username and password) issued by Wise Technologies. Each account is scoped to the schools it may read: a token obtained with your credentials will only be accepted for the school codes assigned to your account.

To obtain credentials, or to have additional schools added to an existing account, write to info_wise@wise-t.com or contact your Wise Technologies representative. Please state which institution(s) you need to read and a short description of your integration.

> ***Note:*** *Credentials are confidential. Store them server-side, never in a mobile or web client that is distributed to end users, and rotate them if you suspect they have leaked. See section 4.4 for security recommendations.*

2. Getting started — a five-minute walkthrough

The following walkthrough uses `curl` and the public demonstration school (school code `demo`). Replace `API_USER` and `API_PASS` with your own credentials.

2.1 Step 0 — check the service

`/v2/meta` requires no authentication and lists every endpoint the server currently serves. It is the quickest way to verify connectivity:

```
curl https://wise-tt.com/api/v2/meta

{
  "service": "Wise Timetable API",
  "version": "2.0.0",
  "legacy": {
    "central": "/WTTWebRestAPI/ws/rest",
    "note": "Legacy paths are served for compatibility and are frozen."
  },
  "endpoints": [
    "GET /v2/login          Basic auth, returns a bearer token",
    "GET /v2/schools",
    "GET /v2/schedule?school=&by=groups|tutor|room|course&id=1_2&from=&to=",
    "..."
  ]
}
```

2.2 Step 1 — log in and obtain a token

Send your credentials as HTTP Basic authentication to `/v2/login`. The response contains a bearer token valid for 24 hours:

```
curl -u "API_USER:API_PASS" https://wise-tt.com/api/v2/login

{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3dHRf...",
  "expiresAt": "2026-08-03T10:15:22+00:00"
}
```

Every subsequent request carries this token in the `Authorization` header:

```
TOKEN=$(curl -s -u "API_USER:API_PASS" https://wise-tt.com/api/v2/login | jq -
r .token)

curl -H "Authorization: Bearer $TOKEN" \
     "https://wise-tt.com/api/v2/schools"
```

2.3 Step 2 — find the groups (or lecturers, rooms, courses) to follow

A schedule is always requested for something: one or more student groups, a lecturer, a room or a course. The code-list endpoints return the ids you will need:

```
curl -H "Authorization: Bearer $TOKEN" \
     "https://wise-tt.com/api/v2/groups?school=demo"

[
  { "id": "1", "label": "AFM1-1" },
  { "id": "2", "label": "AFM1-2" },
  { "id": "4", "label": "BAM2-1" }
]
```

2.4 Step 3 — request the schedule

Ask for a date range (at most one term at a time) and pass the ids joined with underscores. The response contains every event in the range, with rooms, lecturers and groups already filled in, plus the school's holidays and the bounds of the published academic year:

```
curl -H "Authorization: Bearer $TOKEN" \
     "https://wise-tt.com/api/v2/schedule?school=demo&by=groups&id=1_2&from=2025-09-
22&to=2025-09-28"
```

The whole school. Send from and to and NEITHER by NOR id, and the response carries every lesson and every reservation in the range - the replacement for the legacy scheduleAll. The range stays required and is still capped at 400 days: that is the one difference from scheduleAll, and it is what makes the worst case knowable. Measured on the largest school on the server, one week is about 100 kB and 75 ms. Use the ETag (section 3.7) if you poll: an unchanged timetable answers 304 with no body.

```
{
  "school": "demo",
  "from": "2025-09-22",
  "to": "2025-09-28",
  "events": [
    {
      "id": "S10",
      "start_time": "2025-09-22T08:00:00",
      "end_time": "2025-09-22T10:00:00",
      "course": "Making Managerial Decisions Using Accounting Information",
      "eventType": "",
      "note": "",
      "executionType": "tutorial",
      "branches":  [ { "id": 4,  "name": "AFM" } ],
      "rooms":     [ { "id": 12, "name": "B-002" } ],
      "groups":    [ { "id": 1,  "name": "AFM1-1" } ],
      "lecturers": [ { "id": 31, "name": "Teddy Thompson" } ],
      "showLink": "",
      "color": "feff9b",
      "colorText": "tutorial",
      "courseId": 88,
      "forced": false
    }
  ],
  "holidays": [],
  "year": { "from": "2025-09-22", "to": "2026-06-14", "firstWeek": 1, "lastWeek":
38 }
}
```

That is the whole integration loop. The rest of this manual is the reference: conventions and error handling (chapter 3), authentication in detail (chapter 4), every v2 endpoint (chapter 5), the legacy surface (chapter 6) and complete code examples (chapter 7).

3. General conventions

3.1 Base URL and transport

All examples in this manual use the production server `https://wise-tt.com/api`. The API is served exclusively over HTTPS. Responses are JSON encoded as UTF-8, with `Content-Type: application/json`.

3.2 HTTP methods

Every endpoint in chapters 5 and 6 accepts GET, HEAD and OPTIONS (CORS pre-flight) only; any other method receives 405 method_not_allowed, and all of their parameters travel in the query string. The two write endpoints, POST /v2/eventlinks (chapter 11) and POST /v2/student-registration (chapter 12), take a JSON request body in addition to their query-string parameters; they are the only exceptions in this manual. There are no path parameters anywhere.

3.3 Parameter formats

Kind	Format	Example
Dates	ISO 8601 calendar date, `YYYY-MM-DD`. Invalid dates are rejected with 400.	`from=2025-09-22`
Id lists	Numeric ids joined with underscores (commas also accepted). Duplicates are ignored.	`id=1_2_3`
School code	The short code of the institution, as shown on its Wise Timetable web page.	`school=demo`
Timestamps in responses	Local wall-clock time at the school, without a timezone designator.	`2025-09-22T08:00:00`

> ***Note:*** *Event timestamps are the school's local time. When importing into calendar systems, attach the school's timezone (for European institutions typically `Europe/Ljubljana` / CET–CEST) rather than treating the values as UTC.*

3.4 Event identifiers

Every event id is a **string** whose first character identifies the kind of event:

Prefix	Kind	Example
S	A lesson (regular timetabled teaching, from the weekly pattern).	`"S10"`
R	A reservation: exams, room bookings, meetings, lab sessions. The rest of the id is a GUID.	`"R2cce85da-d5e1-4fe2-bc6e-19247583a351"`

Always treat ids as opaque strings — never parse them as integers. Exams arrive in the same schedule responses as lessons; the `R` prefix together with the `eventType` field (for example `"Exam"`) is how they are distinguished.

> ***Note:*** *Numeric ids of groups, lecturers, rooms and courses are regenerated when a school republishes its master data from the desktop application. Do not persist them long-term as stable references; re-read the code lists when the school's data changes (see `lastChangeDate`, section 6.2).*

3.5 Colours

Event colours are sent as six lower-case hexadecimal digits **without** a leading `#` (for example `"color": "feff9b"`), or as an empty string when the school assigned no colour. The accompanying `colorText` field carries the display label of the colour's meaning. Prefer colouring by `color` and treating `executionType` / `eventType` as display labels, not enums — they are localised free text defined by each school.

3.6 Error responses

Every failure returns a meaningful HTTP status code and a JSON body that says what went wrong:

```
{
  "error": "unknown_school",
  "message": "No school is registered under the code 'nosuch'.",
  "hint": "Check the code shown on the school's Wise Timetable web page."
}
```

Field	Description
`error`	A stable, machine-readable error code. Branch your error handling on this.
`message`	A human-readable sentence describing the failure.
`hint`	Optional: a suggestion for resolving the problem.

The complete list of status codes and error codes:

HTTP	error code	Meaning
400	`bad_request`	A parameter is missing or malformed (bad date, non-numeric id, range too large, …). The message names the parameter.
400	`unknown_school`	The school code is not registered on this server at all — usually a typo.
400	`school_not_served`	The school exists but runs Wise Timetable on its own server. Ask the legacy directory endpoint /url for that school's server address (section 6.2).
401	`unauthorized`	Missing, malformed or expired credentials/token. Log in (again).
403	`school_not_permitted`	Your credentials are valid but this school is not in your account's school list. Contact Wise Technologies to have it added.
404	`not_found`	The requested object (event id, student number) does not exist.
404	`no_such_endpoint`	The path is not an endpoint on this server. Check /v2/meta.
404	`student_lookup_disabled`	The school has not enabled student-number lookup (section 5.13).
405	`method_not_allowed`	Only GET is supported.
429	`rate_limited`	Too many student-number lookups from one address. Retry later.
500	`internal_error`	Unexpected server error. The hint contains a reference id — quote it when reporting the problem.
503	`rate_limiter_unavailable`	Student-number lookup temporarily unavailable.

> ***Note: Always branch on the HTTP status code and the `error` field, never on an empty body.*** *The three cases "wrong school code" (400), "not permitted" (403) and "genuinely no events" (200 with an empty list) are different situations and should produce different messages in your application.*

3.7 Caching with ETag

Every response carries an `ETag` header. Repeat the request with `If-None-Match` and the server answers `304 Not Modified` with no body when nothing changed — which is the common case for a timetable polled every few minutes:

```
curl -i -H "Authorization: Bearer $TOKEN" \
     "https://wise-tt.com/api/v2/schedule?school=demo&by=groups&id=1&from=2025-09-
22&to=2025-09-28"
# ...
# ETag: "6d0a2f..."

curl -i -H "Authorization: Bearer $TOKEN" -H 'If-None-Match: "6d0a2f..."' \
     "https://wise-tt.com/api/v2/schedule?school=demo&by=groups&id=1&from=2025-09-
22&to=2025-09-28"
# HTTP/1.1 304 Not Modified
```

Additionally, the legacy directory endpoint `schoolCode` returns `lastChangeDate` — the timestamp of the school's last data change. While it does not move, no timetable in that school has changed, so a client may skip refreshing entirely (section 6.2).

3.8 Cross-origin requests (CORS)

Browser-based clients (single-page applications, dashboards) require CORS headers. The API sends them for origins that are explicitly registered — it does not answer `*`. If you are building a browser application, tell Wise Technologies the origin(s) it will be served from so they can be added to the allow-list. Server-side and native clients are unaffected.

3.9 Limits

Limit	Value	Notes
Token lifetime	24 hours	Re-login when expired; /v2/login returns the expiry so you can refresh proactively.
Schedule date range	400 days per request	Ask for one term (or one academic year) at a time.
Student-number lookups	30 per hour per address	Applies only to /v2/student-groups.

4. Authentication

4.1 Exchanging credentials for a token

Authentication is a two-step scheme: your API username and password are exchanged once for a signed bearer token (JWT), and every data request carries the token. Send the credentials with HTTP Basic authentication to `/v2/login`:

```
GET /api/v2/login HTTP/1.1
Host: wise-tt.com
Authorization: Basic QVBJX1VTRVI6QVBJX1BBU1M=

{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "expiresAt": "2026-08-03T10:15:22+00:00"
}
```

The legacy login (`/api/WTTWebRestAPI/ws/rest/login`) returns the same token but only the `token` field, without `expiresAt`. Both logins accept the same central API accounts and their tokens are interchangeable across both surfaces.

4.2 Using the token

Send the token on every request in the `Authorization` header:

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
```

A missing or expired token produces `401 unauthorized`. Tokens are valid for 24 hours; a long-running service should either re-login on the first 401 and retry once, or refresh proactively shortly before `expiresAt`.

4.3 School scoping

The token encodes the list of school codes your account may read (in the JWT `aud` claim, separated by semicolons). Requesting a school outside that list yields `403 school_not_permitted`; requesting a school code that does not exist yields `400 unknown_school`. The distinction is deliberate — a typo should not read as a permissions problem.

4.4 Security recommendations

Keep credentials server-side. Anything compiled into a distributed mobile or web application can be extracted by its users. If your product is an end-user app, place a small backend of your own between the app and the Wise Timetable API, or discuss per-install token issuing with Wise Technologies.

Use one account per integration. Separate accounts per partner and per product make rotation painless and let permissions be scoped precisely.

Rotate on suspicion. Passwords can be changed server-side without downtime; a new login simply starts returning tokens signed for the new credentials.

Do not log tokens or passwords. A token is a bearer credential: whoever holds it can read every school in its scope until it expires.

5. API v2 reference

All paths below are relative to `https://wise-tt.com/api`. Endpoints marked “token” require the `Authorization: Bearer` header (chapter 4). Unless stated otherwise, parameters named `school` take the school code and are validated against your token’s school list.

5.1 Service description

```
GET  /v2/meta
```

No authentication. Returns the service name, version, and the list of endpoints the server currently serves. Use it as a health check and to discover features added after this manual was written.

```
curl https://wise-tt.com/api/v2/meta
```

5.2 Login

GET /v2/login

Authentication: HTTP Basic. Exchanges your API credentials for a bearer token (see chapter 4).

```
curl -u "API_USER:API_PASS" https://wise-tt.com/api/v2/login

{
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...",
  "expiresAt": "2026-08-03T10:15:22+00:00"
}
```

Field	Type	Description
token	string	The bearer token to send on every subsequent request.
expiresAt	string	ISO 8601 timestamp (UTC) at which the token stops being accepted.

5.3 Schools

GET /v2/schools

Authentication: token. Lists the schools your token may read. An account scoped to specific schools sees exactly those; unrestricted accounts see every school in the central directory.

```
curl -H "Authorization: Bearer $TOKEN" https://wise-tt.com/api/v2/schools

[
  { "code": "demo", "name": "New School" },
  { "code": "wtt",  "name": "Wise Technologies Test" }
]
```

5.4 Schedule — the core endpoint

GET /v2/schedule

Authentication: token. Returns every event in a date range for the groups, lecturer, room or course you follow — with rooms, lecturers, groups and branches filled in on every event — together with the school's holidays in the range and the bounds of the published academic year. One request per view; no follow-up calls are needed.

Parameter	Required	Description
school	yes	School code.
by	no	What the id list refers to: groups, tutor, room or course. Omit it, and id with it, to get the whole school.
id	no	One or more numeric ids joined with underscores, e.g. 1_2_3. Ids come from the code-list endpoints (5.11). Required when by is given.
from	yes	First day of the range, YYYY-MM-DD.
to	yes	Last day of the range, YYYY-MM-DD. At most 400 days after from.

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/v2/schedule?school=demo&by=groups&id=1_2_3&from=2025-09-
22&to=2025-09-28"

{
  "school": "demo",
  "from": "2025-09-22",
  "to": "2025-09-28",
  "events": [
    {
```

```
      "id": "S10",
      "start_time": "2025-09-22T08:00:00",
      "end_time": "2025-09-22T10:00:00",
      "course": "Making Managerial Decisions Using Accounting Information",
      "eventType": "",
      "note": "",
      "executionType": "tutorial",
      "branches":  [ { "id": 4,  "name": "AFM" } ],
      "rooms":     [ { "id": 12, "name": "B-002" } ],
      "groups":    [ { "id": 1,  "name": "AFM1-1" } ],
      "lecturers": [ { "id": 31, "name": "Teddy Thompson" } ],
      "showLink": "",
      "color": "feff9b",
      "colorText": "tutorial",
      "courseId": 88,
      "forced": false
    },
    {
      "id": "R2cce85da-d5e1-4fe2-bc6e-19247583a351",
```


New in 1.2: Revision 1.2 adds four fields to every event — foreignId, linkEvent, linkResolved and isOnline — documented in chapter 11.


```
      "start_time": "2025-09-24T10:00:00",
      "end_time": "2025-09-24T12:00:00",
      "course": "",
      "eventType": "Exam",
      "note": "Midterm exam",
      "executionType": "",
      "branches":  [],
      "rooms":     [ { "id": 5, "name": "A-001" } ],
      "groups":    [ { "id": 1, "name": "AFM1-1" } ],
      "lecturers": [ { "id": 17, "name": "Andrew Altman" } ],
      "showLink": "",
      "color": "ff0000",
      "colorText": "Exam",
      "courseId": 0,
      "forced": false
    }
  ],
  "holidays": [
    { "date": "2025-09-26", "name": "Faculty day", "allDay": true, "fromSlot": 0,
"slots": 0 }
  ],
  "year": { "from": "2025-09-22", "to": "2026-06-14", "firstWeek": 1, "lastWeek":
38 }
}
```


Ordering. Events are ordered by start time; events starting at the same time follow the order of the ids you asked for (the first group's lesson first). This keeps day views stable.

Following several dimensions. To follow, say, a lecturer **and** a room, make two requests and merge client-side; `by` takes a single dimension per request.

> ***Note:*** *Filtering by `course` returns lessons only — reservations are not attached to a course dimension and legitimately yield none.*

5.5 The event object

Every event in a schedule response — lesson or reservation — has the same shape:

Field	Type	Description
`id`	string	Event id with the S/R kind prefix (section 3.4).
`start_time`	string	Start, local school time, YYYY-MM-DDTHH:MM:SS.
`end_time`	string	End, local school time. Read from the school's own bell schedule —

Field	Type	Description
		never assume fixed slot lengths.
`course`	string	Course name, or "" for reservations not linked to a course.
`eventType`	string	Reservation type label (`Exam`, `Labs`, `Meeting`, `Other`, `Reserved`); "" for lessons.
`note`	string	The scheduler's note for this slot. Important: this is where cancellations and moved hours are announced (e.g. "Cancelled"). Display it prominently when present.
`executionType`	string	How the course is taught: lecture, tutorial, seminar, … Localised display text defined by the school — treat as a label, not an enum.
`branches`	array	Study branches this event reaches: `[{id, name}]` where name is the branch code (e.g. "AFM").
`rooms`	array	`[{id, name}]` — the room(s) where the event takes place.
`groups`	array	`[{id, name}]` — the student groups attending.
`lecturers`	array	`[{id, name}]` — the lecturer(s), full display name.
`showLink`	string	Online-lecture URL when the course has one, otherwise "".
`color`	string	Six hex digits without #, or "" (section 3.5).
`colorText`	string	Display label associated with the colour.
`courseId`	number	v2 only. Numeric id of the course, for cross-referencing with /v2/courses and /v2/notifications.
`forced`	boolean	v2 only. True when the school explicitly forced this lesson into this week — it takes place even on a holiday. A client that blanks holiday days must still show forced events.

> ***Note:*** *The field names (`start_time`, `executionType`, …) intentionally match the legacy API, so a legacy client can migrate to v2 without changing its parser; `courseId` and `forced` are appended for v2 only.*

5.6 Event detail

`GET` `/v2/event`

Authentication: token. Everything known about one event, including the full programme → year → branch → group hierarchy for every attending group. Use it for a detail view after the user taps an event.

Parameter	Required	Description
`school`	yes	School code.
`id`	yes	The event id from a schedule response, e.g. `S10` or `R2cce85da-…`.

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/v2/event?school=demo&id=S10"

{
  "id": "S10",
  "decoration": "false",
  "startDate": "22.09.2025",
  "endDate": "30.12.2025",
  "startHour": "08:00",
```

```
  "endHour": "10:00",
  "note": "",
  "course": "Making Managerial Decisions Using Accounting Information",
  "executionType": "tutorial",
  "roomIds": [12],
  "rooms": ["B-002"],
  "groupIds": [1],
  "groups": [
    {
      "id": 1, "name": "AFM1-1",
      "branch_id": 4, "branch_name": "Accounting for Management",
      "year": 1,
      "program_id": 1, "program_name": "undergraduate"
    }
  ],
  "lecturerIds": [31],
  "lecturers": ["Teddy Thompson"]
}
```


`startDate` / `endDate` (format `dd.MM.yyyy`) are the validity range of the recurring lesson — the first and last date on which it runs — not the dates of one occurrence. For a reservation they are the booking's first and last day.

5.7 Academic year bounds

GET `/v2/year`

Authentication: token. The first and last date of the published academic year. Use it to stop the user navigating into weeks that cannot contain data. The same object is included in every `/v2/schedule` response.

Parameter	Required	Description
`school`	yes	School code.


```
{ "from": "2025-09-22", "to": "2026-06-14", "firstWeek": 1, "lastWeek": 38 }
```


Wise Timetable publishes one academic year at a time; `from`/`to` are `null` when the school has not published yet.

5.8 Capabilities

GET `/v2/capabilities`

Authentication: token. What this school allows a client application to offer. Some institutions treat lecturer engagements as confidential and do not want a browse-by-lecturer picker shown; this endpoint tells you which pickers to display. Ask once per school and cache.

Parameter	Required	Description
`school`	yes	School code.


```
{
  "browse": { "groups": true, "tutor": true, "room": true, "course": true },
  "studentLookup": false,
  "source": { "customer": "", "school": "" }
}
```

Field	Type	Description
`browse`	object	Per dimension, whether a picker for it should be offered. groups is always true.
`studentLookup`	boolean	Whether /v2/student-groups is enabled for this school (5.13).
`source`	object	Diagnostic echo of the configuration; useful when discussing behaviour with support.

weekDays (number, added in rev 1.1) How many days the school's week displays: 5, 6 or 7. Render weekend columns only when this says so.

> ***Note:*** *This is a presentation setting, not access control — hide the pickers it disables, but do not rely on it to protect data.*

5.9 Holidays

`GET /v2/holidays`

Authentication: token. Non-teaching days in a date range, so an empty day can be labelled ("1 May — public holiday") instead of looking like missing data. Also included in every `/v2/schedule` response.

Parameter	Required	Description
`school`	yes	School code.
`from`	yes	First day, YYYY-MM-DD.
`to`	yes	Last day, YYYY-MM-DD.

```
[
  { "date": "2026-01-01", "name": "New Year", "allDay": true,  "fromSlot": 0,
"slots": 0 },
  { "date": "2026-02-08", "name": "Culture day", "allDay": true, "fromSlot": 0,
"slots": 0 }
]
```

Field	Type	Description
`date`	string	The holiday's date, YYYY-MM-DD.
`name`	string	The school's label. May be a bare date string when the school entered no name — fall back to your own wording.
`allDay`	boolean	True when the whole teaching day is off.
`fromSlot / slots`	number	For partial holidays: the first bell-schedule slot affected and how many slots. Rarely used.

> ***Note:*** *A lesson with `"forced": true` (section 5.5) takes place even on a holiday. When blanking holiday days, keep forced events visible.*

5.10 Notifications

`GET /v2/notifications`

Authentication: token. Messages the institution's administrators have attached to groups, courses or lecturers, plus school-wide announcements. Send the ids currently on the user's screen; the server

returns the notices that concern them. All id parameters are optional — with none, only school-wide messages are returned.

Parameter	Required	Description
`school`	yes	School code.
`groups`	no	Group ids, e.g. `1_2`. Notices for these groups are returned; unless `courses` is also given, notices of the courses those groups attend are included too.
`tutors`	no	Lecturer ids.
`courses`	no	Course ids (use the `courseId` field of the events on screen). When present, exactly these courses are read.

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/v2/notifications?school=demo&groups=1_2&courses=88"

[
  {
    "id": "course:88",
    "level": "course",
    "subject": "Enterprise Resource Planning",
    "text": "Lecture of 5 May moved to hall B-007.",
    "date": null
  },
  {
    "id": "school:3",
    "level": "school",
    "subject": "",
    "text": "The library is closed during exam week.",
    "date": "2026-05-02T09:12:00"
  }
]
```

Field	Type	Description
`id`	string	Stable id, level-prefixed. Deduplicated — each notice appears once.
`level`	string	Where the notice is attached: `group`, `course`, `lecturer` or `school`.
`subject`	string	The name of the group/course/lecturer the notice belongs to; "" for school-wide.
`text`	string	The message.
`date`	string?	Creation time when known (school-level messages), otherwise null.

5.11 Code lists

Five sibling endpoints return the id/label lists a client needs to build pickers and to translate ids into names. All take a single required parameter, the school code, and require a token.

GET `/v2/programmes`

```
[ { "id": "1", "name": "undergraduate", "year": "3" } ]
```

Study programmes. `year` is the **number of study years** the programme runs — not a calendar year.

GET `/v2/lecturers`

```
[ { "id": "17", "firstName": "Andrew", "lastName": "Altman",
    "email": "aaltman@example.edu", "isAssistant": false } ]
```

Alphabetically ordered by surname.

GET `/v2/rooms`

```
[ { "id": "1", "label": "A-001", "code": "A001", "email": "a001@example.edu", "link":
null, "seats": 30 } ]
```

Rooms in the school's own display order. code is the room's identifier in the school's own system — what an SIS maps against; email is the room's mailbox, used to reach whoever manages it and to book it as a resource in Exchange or Outlook; link is a page the school hangs on the room, such as a floor plan or a booking page; seats is its capacity. code, email and link are null when the school has not filled them in, which is the common case — do not treat a null as an error. email is also null when the stored value is not a valid address: the field is free text in the desktop application and a few schools have typed something else into it. Added in manual revision 1.3; a server deployed before August 2026 sends id and label only, and the legacy /{db}/ws/rest/rooms endpoint (section 6.4) is unchanged.

GET `/v2/courses`

```
[ { "id": "88", "label": "Enterprise Resource Planning", "code": "ERP1",
"optionalCourse": false, "centralAdmin1": false, "centralAdmin2": false } ]
```

Every subject the school teaches, alphabetical — each one ONCE. Enables browsing the timetable by subject. code is the school's own identifier for the subject and is null when it has none. optionalCourse marks an elective; centralAdmin1 and centralAdmin2 are the school's two "centrally administered" marks, and either one set means students must not be enrolled onto that subject through the API. For the same subjects listed once per branch, with programme and year, see 12.1. New in revision 1.4.

GET `/v2/groups`

```
[ { "id": "1", "label": "AFM1-1" }, { "id": "4", "label": "BAM2-1" } ]
```

The flat list of all active student groups. For the hierarchical walk programme → year → branch → group, use the legacy endpoints `branchAllForProgrmmeYear` and `groupAllForBranch` (section 6.3).

5.12 Bell schedule

GET `/v2/times`

Authentication: token. The school's bell schedule with real clock times — every line a timetable grid should draw. Slot lengths differ between schools (15, 30 or 60 minutes); never assume one.

Parameter	Required	Description
`school`	yes	School code.

```
[
  { "id": 1, "start": "07:00", "end": "07:30", "label": "7.00" },
  { "id": 2, "start": "07:30", "end": "08:00", "label": "7.30" },
  { "id": 3, "start": "08:00", "end": "08:30", "label": "8.00" }
]
```

Note: *This endpoint is a recent addition — confirm its presence in `/v2/meta` on the server you target. The legacy `/ws/rest/times` endpoint (section 6.4) returns the school's display labels only.*

5.13 Student-number lookup

`GET /v2/student-groups`

Authentication: token. Turns a student number into the groups that student attends, as a convenience shortcut past the programme → year → branch → group walk. It is not a login and returns nothing personal — only group ids and names, information already printed on the public timetable.

The feature is **off by default** and enabled per school; check `studentLookup` in `/v2/capabilities` before offering an input field. Lookups are rate-limited (30 per hour per address).

Parameter	Required	Description
`school`	yes	School code.
`number`	yes	The student number, at most 32 characters.

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/v2/student-groups?school=demo&number=64250004"

[
  {
    "id": "4",
    "name": "BAM2-1",
    "programName": "undergraduate",
    "year": 2,
    "branchName": "Business and Management"
  }
]
```

An unknown number answers `404 not_found` — deliberately identical for a number that does not exist and one with no groups. `404 student_lookup_disabled` means the school has not enabled the feature; `429 rate_limited` means too many attempts.

6. The legacy-compatible API

The original Wise Timetable REST API is served, byte-compatible, under the `/api` prefix. It exists so that established clients and integrations keep working unchanged: an existing integration migrates to this server by changing one string — `https://wise-tt.com/…` becomes `https://wise-tt.com/api/…` — and receives the same paths, parameters and JSON shapes as before.

For new integrations, prefer API v2 (chapter 5). The legacy surface is frozen: its quirks (documented in 6.5) are preserved on purpose and will not be fixed. The one improvement it does receive on this server: rooms, lecturers and groups are filled in on schedule events, and errors return meaningful JSON bodies instead of `[]`.

The legacy surface consists of three parts: a central directory (one per server), and two per-school APIs (a “mobile” one and a “per-faculty” one), whose paths embed the school’s database name:

```
https://wise-tt.com/api/WTTWebRestAPI/ws/rest/...     central directory
https://wise-tt.com/api/wtt_demo/ws/restMobile/...    mobile API   (per school)
https://wise-tt.com/api/wtt_demo/ws/rest/...          faculty API  (per school)
```

> ***Note:*** *The per-school prefix is the **database name** (e.g. `wtt_demo`), not the school code (`demo`). You obtain it from the directory’s `/url` and `/schoolCode` endpoints below — clients should follow those answers rather than construct URLs by hand.*

6.2 Central directory

GET /WTTWebRestAPI/ws/rest/login

HTTP Basic in, `{ "token": "…" }` out. Same accounts and same token as `/v2/login` (section 4.1), without the `expiresAt` field.

GET /WTTWebRestAPI/ws/rest/url

Parameter	Required	Description
schoolCode	yes	School code, e.g. demo.
language	no	Locale code (e.g. slo, eng). Accepted for compatibility.

Returns the base URL a client should use for this school's data endpoints. For schools hosted on wise-tt.com the answer points back at this server; a handful of institutions run Wise Timetable on their own servers, and for those the directory returns their address — this endpoint is how a client finds it.

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/WTTWebRestAPI/ws/rest/url?schoolCode=demo&language=slo"

{ "server": "https://wise-tt.com/api/wtt_demo/ws/restMobile/" }
```

GET /WTTWebRestAPI/ws/rest/schoolCode

Parameter	Required	Description
schoolCode	yes	School code.
language	no	Locale code.

```
{
  "schoolCode": "wtt_demo",
  "schoolCity": "London",
  "schoolName": "New School",
  "firstDayOfWeek": 1,
  "lastChangeDate": "06.01.2026 14:52"
}
```

Field	Description
schoolCode	The school's database name — the per-school URL prefix.
firstDayOfWeek	ISO 8601 weekday the school's week starts on: 1 = Monday, 6 = Saturday, 7 = Sunday. Respect it when drawing week views — some institutions start on Saturday or Sunday.
lastChangeDate	Timestamp of the school's last data change (dd.MM.yyyy HH:mm). A free cache key: while it does not move, nothing in the school's timetable has changed.

6.3 Mobile API (per school)

All endpoints below live under `/{database}/ws/restMobile/` and take `schoolCode` (the school code) and `language` on every call, plus a bearer token. Ids in responses are strings.

GET /{db}/ws/restMobile/basicProgrammeAll

```
curl -H "Authorization: Bearer $TOKEN" \
  "https://wise-tt.com/api/wtt_demo/ws/restMobile/basicProgrammeAll?
schoolCode=demo&language=slo"

[ { "id": "1", "name": "undergraduate", "year": "3" } ]
```

Study programmes; `year` is the programme's number of study years.

`GET /{db}/ws/restMobile/basicTutorAll`

```
[ { "id": "1", "firstName": "James", "lastName": "Gilbert" } ]
```

All lecturers, ordered by id.

`GET /{db}/ws/restMobile/branchAllForProgrmmeYear`

Parameter	Required	Description
`programmeId`	yes	Programme id from basicProgrammeAll.
`year`	yes	Year of study, 1-based.

```
[ { "id": "2", "branchName": "Business and Management" } ]
```

Note: *The endpoint name really is spelled `Progrmme` — the typo is part of the deployed URL and is preserved deliberately.*

`GET /{db}/ws/restMobile/groupAllForBranch`

Parameter	Required	Description
`branchId`	yes	Branch id from the previous endpoint.

```
[ { "id": "1", "name": "AFM1-1", "childGroups": [] },
  { "id": "9", "name": "BAM1-1",
    "childGroups": [ { "id": "15", "name": "BAM1-1a", "childGroups": [] } ] } ]
```

Groups of a branch, with split sub-groups nested one level deep. A lesson can be attached to either level.

`GET /{db}/ws/restMobile/scheduleByGroups`

`GET /{db}/ws/restMobile/scheduleByTutor`

Parameter	Required	Description
`schoolCode`	yes	School code.
`dateFrom`	yes	YYYY-MM-DD.
`dateTo`	yes	YYYY-MM-DD.
`language`	no	Locale code.
`groupsId`	yes*	For scheduleByGroups: group ids joined with underscores (1_2_3). Note the plural parameter name.
`tutorId`	yes*	For scheduleByTutor: the lecturer id.

Returns a flat JSON array of event objects — the same event shape as v2 (section 5.5) **without** the `courseId` and `forced` fields. On this server rooms, lecturers and groups are filled in; on the original WildFly server those arrays were always empty.

```
[
  {
    "id": "S10",
    "start_time": "2025-09-22T08:00:00",
    "end_time": "2025-09-22T10:00:00",
    "course": "Making Managerial Decisions Using Accounting Information",
    "eventType": "",
    "note": "",
```

```
    "executionType": "tutorial",
    "branches":  [ { "id": 4, "name": "AFM" } ],
    "rooms":     [ { "id": 12, "name": "B-002" } ],
    "groups":    [ { "id": 1, "name": "AFM1-1" } ],
    "lecturers": [ { "id": 31, "name": "Teddy Thompson" } ],
    "showLink": "",
    "color": "feff9b",
    "colorText": "tutorial"
  }
]
```

`GET /{db}/ws/restMobile/notificationByGroups`

Parameter	Required	Description
`groupsId`	yes	Group ids, underscore-joined.

Returns a flat array of notification strings. **"No notifications" is `[""]` — a one-element array containing an empty string** — not `[]`. Filter empty strings out. For structured notifications with level and subject, use `/v2/notifications` (section 5.10).

6.4 Faculty API (per school)

The second per-school surface lives under `/{database}/ws/rest/`. It powers the school's web timetable. Its login is separate: it authenticates against the school's own web-user accounts, not the central API accounts.

`GET /{db}/ws/rest/login`

HTTP Basic with a web user of that school; returns `{ "token": "…" }`. A school may have this login disabled.

`GET /{db}/ws/rest/times`

```
[ { "id": "1", "label": "07:00" }, { "id": "2", "label": "07:30" } ]
```

The bell schedule as display labels (free text set by the school). For real clock times use `/v2/times`.

`GET /{db}/ws/rest/rooms`

```
[ { "id": "1", "label": "A-001" } ]
```

`GET /{db}/ws/rest/groups`

```
[ { "id": "1", "label": "AFM1-1" } ]
```

`GET /{db}/ws/rest/lecturers`

```
[ { "id": "17", "firstName": "Andrew", "lastName": "Altman",
    "email": "aaltman@example.edu", "isAssistant": false } ]
```

Alphabetical by surname (unlike basicTutorAll, which is ordered by id).

`GET /{db}/ws/rest/scheduleDetail`

Parameter	Required	Description
`id`	yes	Event id, e.g. S10 or R<guid>.

Full detail of one event — the legacy equivalent of `/v2/event`, with the same response shape (section 5.6). See 6.5 for a known date quirk.

6.5 Legacy quirks to be aware of

These behaviours are preserved on purpose for byte-compatibility. New integrations using v2 are unaffected.

Quirk	Detail
Types are display text	Legacy `executionType` / `eventType` arrive as localised prose, not stable codes. Colour by `color`, display the text as-is.
Empty notifications	`notificationByGroups` answers `[""]` when there is nothing to say.
scheduleDetail validity dates	The original server reports `startDate`/`endDate` one day late (a historical off-by-one). The legacy endpoint reproduces it so that existing screens do not shift; `/v2/event` always reports the correct dates.
Two orderings of the same list	`basicTutorAll` is ordered by id, `/ws/rest/lecturers` alphabetically. Both are as originally deployed.
Misspelled endpoint	`branchAllForProgrmmeYear` — the typo is part of the URL.
language parameter	Translations are returned only where the school has entered them; many schools maintain names in one language only. Do not rely on `language` for localisation of your own UI.

7. Complete integration examples

Each example below performs the full loop: log in, fetch the group list, fetch one week of the schedule, and print the events. Replace `API_USER` / `API_PASS` with your credentials and `demo` with your school code.

7.1 Shell (curl + jq)

```
#!/bin/sh
# Complete Wise Timetable API round trip using curl and jq.

BASE="https://wise-tt.com/api"

# 1. Log in: exchange Basic credentials for a 24-hour bearer token.
TOKEN=$(curl -s -u "API_USER:API_PASS" "$BASE/v2/login" | jq -r .token)

# 2. List the school's groups and pick the ids you need.
curl -s -H "Authorization: Bearer $TOKEN" \
     "$BASE/v2/groups?school=demo" | jq .

# 3. Fetch one week of the schedule for groups 1 and 2.
curl -s -H "Authorization: Bearer $TOKEN" \
     "$BASE/v2/schedule?school=demo&by=groups&id=1_2&from=2025-09-22&to=2025-09-28" \
  | jq -r '.events[] | "\(.start_time)  \(.course)  room: \(.rooms[0].name // "-")"'
```

7.2 Python

```
"""Wise Timetable API — minimal integration example (Python 3, requests)."""
import requests

BASE = "https://wise-tt.com/api"
AUTH = ("API_USER", "API_PASS")
SCHOOL = "demo"

# 1. Log in. The token is valid for 24 hours; a long-running service
#    should re-login when it receives 401, or shortly before expiresAt.
login = requests.get(f"{BASE}/v2/login", auth=AUTH, timeout=15)
login.raise_for_status()
token = login.json()["token"]
```

```
headers = {"Authorization": f"Bearer {token}"}

def api(path, **params):
    """GET one endpoint and return parsed JSON, with real error handling."""
    r = requests.get(f"{BASE}{path}", headers=headers,
                     params={"school": SCHOOL, **params}, timeout=15)
    if r.status_code != 200:
        # Every error body carries a machine-readable code and a sentence.
        err = r.json()
        raise RuntimeError(f"{r.status_code} {err['error']}: {err['message']}")
    return r.json()

# 2. Which groups exist?
groups = api("/v2/groups")
print("groups:", [(g["id"], g["label"]) for g in groups[:5]])

# 3. One week of the timetable for groups 1 and 2 -
#    rooms, lecturers and groups are already filled in on every event.
week = api("/v2/schedule", by="groups", id="1_2",
           **{"from": "2025-09-22", "to": "2025-09-28"})

for e in week["events"]:
    rooms = ", ".join(r["name"] for r in e["rooms"]) or "-"
    kind = e["eventType"] or e["executionType"]      # "Exam" / "lecture" / ...
    print(f'{e["start_time"]}  {e["course"] or e["note"]:45.45}  '
          f'{kind:10.10}  {rooms}')

# 4. Holidays arrived in the same response - label empty days with them.
for h in week["holidays"]:
    print("holiday:", h["date"], h["name"])
```

7.3 JavaScript (Node.js)

```
// Wise Timetable API — minimal integration example (Node 18+, built-in fetch).
const BASE = "https://wise-tt.com/api";
const SCHOOL = "demo";

async function main()
{
    // 1. Log in with HTTP Basic and keep the bearer token.
    const basic = Buffer.from("API_USER:API_PASS").toString("base64");
    const login = await fetch(BASE + "/v2/login",
        { headers: { Authorization: "Basic " + basic } });
    if (!login.ok) throw new Error("login failed: " + login.status);
    const { token } = await login.json();

    // Small helper: call an endpoint, branch on the status code.
    async function api(path, params)
    {
        const qs = new URLSearchParams({ school: SCHOOL, ...params });
        const r = await fetch(BASE + path + "?" + qs,
            { headers: { Authorization: "Bearer " + token } });
        if (!r.ok)
        {
            const err = await r.json();      // { error, message, hint? }
            throw new Error(r.status + " " + err.error + ": " + err.message);
        }
        return r.json();
    }

    // 2. The code lists you need for pickers.
    const groups = await api("/v2/groups", {});
    console.log("first groups:", groups.slice(0, 3));

    // 3. One week of events for groups 1 and 2.
    const week = await api("/v2/schedule",
        { by: "groups", id: "1_2", from: "2025-09-22", to: "2025-09-28" });

    for (const e of week.events)
```

```
    {
        const rooms = e.rooms.map(r => r.name).join(", ") || "-";
        console.log(e.start_time, e.course || e.note, "@", rooms);
    }
}

main().catch(err => { console.error(err.message); process.exit(1); });
```

7.4 C (libcurl)

For integrations written in C, the API is equally simple to consume: libcurl for HTTP and any JSON parser (the example prints the raw JSON; in production pair it with a parser such as cJSON or jansson). Build with `gcc wtt_client.c -lcurl -o wtt_client`.

```
/* wtt_client.c — Wise Timetable API round trip in plain C with libcurl.
 *
 * 1. exchange Basic credentials for a bearer token at /v2/login
 * 2. request one week of the schedule with the token
 *
 * The token arrives inside a small JSON object; to keep the example
 * dependency-free it is extracted with a simple string scan. In real
 * code use a JSON library (cJSON, jansson) for all parsing.
 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <curl/curl.h>

#define BASE "https://wise-tt.com/api"

/* Growable receive buffer for curl's write callback. */
struct buf
{
    char   *data;
    size_t  len;
};

static size_t on_data(void *chunk, size_t size, size_t n, void *userp)
{
    struct buf *b = (struct buf *) userp;
    size_t add = size * n;

    char *grown = realloc(b->data, b->len + add + 1);
    if (grown == NULL)
    {
        return 0;                       /* out of memory: abort transfer */
    }

    b->data = grown;
    memcpy(b->data + b->len, chunk, add);
    b->len += add;
    b->data[b->len] = '\0';

    return add;
}

/* One GET request; returns the HTTP status, body lands in *out. */
static long http_get(const char *url, const char *bearer, struct buf *out)
{
    CURL *h = curl_easy_init();
    struct curl_slist *hdrs = NULL;
    long status = 0;

    out->data = NULL;
    out->len  = 0;

    if (bearer != NULL)
    {
```

```
        char line[4096];
        snprintf(line, sizeof line, "Authorization: Bearer %s", bearer);
        hdrs = curl_slist_append(hdrs, line);
        curl_easy_setopt(h, CURLOPT_HTTPHEADER, hdrs);
    }

    curl_easy_setopt(h, CURLOPT_URL, url);
    curl_easy_setopt(h, CURLOPT_WRITEFUNCTION, on_data);
    curl_easy_setopt(h, CURLOPT_WRITEDATA, out);

    if (curl_easy_perform(h) == CURLE_OK)
    {
        curl_easy_getinfo(h, CURLINFO_RESPONSE_CODE, &status);
    }

    curl_slist_free_all(hdrs);
    curl_easy_cleanup(h);

    return status;
}

int main(void)
{
    struct buf body;
    char token[3072] = "";
    long status;

    curl_global_init(CURL_GLOBAL_DEFAULT);

    /* --- 1. login: Basic credentials go inside the URL for brevity;
     *        use CURLOPT_USERPWD in production code.                */
    status = http_get("https://API_USER:API_PASS@wise-tt.com"
                      "/api/v2/login", NULL, &body);
    if (status != 200)
    {
        fprintf(stderr, "login failed: HTTP %ld\n%s\n", status,
                body.data ? body.data : "");
        return 1;
    }

    /* Extract "token":"..." from the response (JSON library in real code). */
    {
        const char *k = strstr(body.data, "\"token\":\"");
        if (k != NULL)
        {
            k += 9;
            const char *end = strchr(k, '"');
            if (end != NULL && (size_t)(end - k) < sizeof token)
            {
                memcpy(token, k, (size_t)(end - k));
                token[end - k] = '\0';
            }
        }
    }
    free(body.data);

    /* --- 2. one week of the schedule for groups 1 and 2. */
    status = http_get(BASE "/v2/schedule?school=demo&by=groups"
                      "&id=1_2&from=2025-09-22&to=2025-09-28",
                      token, &body);
    if (status != 200)
    {
        /* The body is a JSON object: { "error", "message", "hint" }. */
        fprintf(stderr, "schedule failed: HTTP %ld\n%s\n", status,
                body.data ? body.data : "");
        return 1;
    }

    printf("%s\n", body.data);        /* hand to your JSON parser here */
```

```
    free(body.data);
    curl_global_cleanup();

    return 0;
}
```

8. Best practices

Practice	Why
Branch on HTTP status codes and the error field	A wrong school code (400), a permission problem (403) and an empty week (200, empty list) are different situations. Show different messages for each; never infer errors from an empty body.
Ask for one term at a time	A term (or academic year) per request is efficient — the server materialises events fast — but requests over 400 days are rejected. Use /v2/year to know the bounds.
Use ETag / If-None-Match when polling	A repeated fetch of an unchanged week costs a 304 with no body. Combined with lastChangeDate, a well-behaved client transfers almost nothing on quiet days.
Treat ids as strings and labels as labels	Event ids carry an S/R prefix; executionType and eventType are localised display text. Colour by color, branch on the id prefix.
Respect firstDayOfWeek	Some institutions start the week on Saturday or Sunday. Read it from the directory's schoolCode endpoint (6.2).
Honour capabilities	Hide the pickers a school has disabled (5.8) and only offer student-number entry where studentLookup is true.
Show the note field prominently	Cancellations and moved lessons are announced in note. An integration that hides it will show students lessons that will not take place.
Render holidays	An empty day with a holiday label reads as intended; an unexplained empty day reads as a broken app. Keep forced events visible on holidays.
Re-read code lists after data changes	Numeric ids are regenerated when the school republishes. Watch lastChangeDate and refresh your cached lists when it moves.

9. Troubleshooting

Symptom	Likely cause and fix
`401 unauthorized` on /v2/login	Wrong username or password, or credentials sent in the wrong form — they must arrive as an HTTP Basic Authorization header. Verify with the curl example in 2.2.
`401` on data endpoints after working for a while	The token expired (24-hour lifetime). Log in again; design your client to re-login on 401 and retry once.
`403 school_not_permitted`	Your account does not include that school. Contact Wise Technologies to extend the account's school list.
`400 unknown_school`	Typo in the school code, or the code differs from what you expect — codes are assigned by Wise Technologies and shown on the school's timetable web page.
`400 school_not_served`	That institution hosts Wise Timetable on its own server. Call the directory /url endpoint (6.2) and send data requests to the server it returns.
Empty events array with status 200	Genuinely nothing scheduled: dates outside the published year (check /v2/year), a week the school has not published, or ids that match nothing.

Symptom	Likely cause and fix
	Also check holidays.
Times seem shifted	Timestamps are the school's local wall-clock time without a timezone designator. Do not parse them as UTC.
Browser app: requests blocked by CORS	Your origin is not on the allow-list. Send Wise Technologies the exact origin (scheme + host + port) your app is served from.
`500 internal_error`	Unexpected server-side failure. The hint contains a reference id — include it when reporting, it lets the exact log lines be found immediately.

10. Support

For credentials, school onboarding, CORS origin registration and technical questions about this API, write to info_wise@wise-t.com. When reporting a problem, include: the full request URL (without credentials), the HTTP status and JSON error body, the reference id for 500 errors, and the approximate time of the request.

Check `GET https://wise-tt.com/api/v2/meta` for the current API version and endpoint list — new capabilities are added there first.

11 Per-event online links (rev 1.2)

11.1 New fields on every event

GET /v2/schedule and GET /v2/event now return four additional fields on every event - lessons (S ids) and reservations (R ids) alike:

Field	Type	Meaning
foreignId	string	Stable identifier (GUID) of this event. Survives timetable re-publishes - use it as the key when pushing links back. Also returned by the legacy scheduleAll API.
linkEvent	string	The online link attached to this specific event (may be empty).
linkResolved	string	The link a client should display. Lessons: room link, else course link, else event link. Reservations: event link, else room link.
isOnline	boolean	true when linkResolved is non-empty.

Field availability: the underlying columns are created automatically the first time a school publishes with Wise Timetable 15.2 or newer. Until then the fields are returned empty/false - clients need no special handling.

11.2 POST /v2/eventlinks — push links onto events

Writes online links onto individual events, keyed by foreignId. Intended for integrations that generate meeting links externally (e.g. MS Graph / Teams) and attach them to timetable events.

11.2.1 Request

```
POST /v2/eventlinks?school=<code>
Authorization: Bearer <token from /v2/login>
Content-Type: application/json

[
  { "foreignId": "9f6c1c2e-...", "link": "https://teams.microsoft.com/l/meetup-
join/...",
    "startTime": "2026-04-24T16:00:00", "endTime": "2026-04-24T20:00:00" },
  { "foreignId": "b21a77d0-...", "link": "" }
```

```
]
```

Rules: link must be at most 256 characters and start with http:// or https://; an empty string clears the link. startTime/endTime are accepted for the caller's convenience and ignored. The body may also be wrapped as {"links":[…]}.

11.2.2 Response

```
{
  "ok": 1, "notFound": 1, "invalid": 0,
  "items": [
    { "foreignId": "9f6c1c2e-...", "status": "ok",       "table": "schedule"    },
    { "foreignId": "b21a77d0-...", "status": "notFound"                          }
  ]
}
```

Each foreignId is looked up among lessons first, then among reservations ("table" reports which one matched). Errors follow the standard shapes from rev 1.1 chapter 3; a school whose database does not yet carry the link columns answers 409 links_not_supported_yet.

11.2.3 Round-trip guarantee

Links pushed through this endpoint are preserved when the school re-publishes its timetable from the desktop application: a link entered manually in the desktop wins, otherwise the pushed link is kept. Deleting an event invalidates its foreignId; the next push for it returns notFound.

11.3 Examples

11.3.1 curl

```
TOKEN=$(curl -s -u API_USER:API_PASS https://wise-tt.com/api/v2/login | jq -
r .token)

curl -s -X POST "https://wise-tt.com/api/v2/eventlinks?school=demo" \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
  -d '[{"foreignId":"9f6c1c2e-1234-4a5b-8c9d-0e1f2a3b4c5d",
        "link":"https://teams.microsoft.com/l/meetup-join/..."}]'
```

11.3.2 Python

```
import requests

BASE = "https://wise-tt.com/api/v2"
token = requests.get(f"{BASE}/login", auth=("API_USER", "API_PASS")).json()
["token"]
hdrs = {"Authorization": f"Bearer {token}"}

# read events incl. the new fields
events = requests.get(f"{BASE}/schedule", headers=hdrs, params={
    "school": "demo", "from": "2026-04-20", "to": "2026-04-26", "groups":
"1_2"}).json()
online = [e for e in events if e["isOnline"]]

# push a Teams link onto one event
r = requests.post(f"{BASE}/eventlinks", headers=hdrs, params={"school": "demo"},
    json=[{"foreignId": events[0]["foreignId"], "link":
"https://teams.microsoft.com/..."}])
print(r.json())
```

11.4 Legacy API (scheduleAll)

For compatibility, the frozen legacy surface gained the same four read-only fields on every event returned by scheduleAll and its sibling endpoints. New integrations should nevertheless use /v2 - the push endpoint exists only there.

12 Integration endpoints (student information systems)

New in revision 1.4. These three endpoints, together with the whole-school form of /v2/schedule (5.5), are what an SIS integration needs in order to leave the frozen legacy surface. Each replaces one named legacy endpoint and keeps that endpoint's field names, so the migration is a change of URL rather than a change of parser.

12.1 GET /v2/course-branches — replaces courseAll

GET /v2/course-branches?school=<code>

Authentication: token. Every subject ONCE PER BRANCH it is taught to, with the programme and year of that branch. A subject taught to four branches appears four times; a subject attached to no branch appears once with the branch fields null. This is the list an SIS maps its own courses onto, usually on code.

```
[ { "id": "911", "course": "Systems Analysis", "code": "SA-2",
    "optionalCourse": false, "centralAdmin1": false, "centralAdmin2": false,
    "programmeId": "1", "programme": "undergraduate", "year": "2",
    "branchId": "38", "branch": "Business Research and Consultancy" } ]
```

Ids are strings here as everywhere else in v2 — the legacy courseAll sent them as strings too. /v2/courses (5.11) remains the one-row-per-subject list for building a picker; this is the branch-wise one. Neither lists the placeholder row with id 0 that every database carries and no school teaches.

12.2 GET /v2/branch-groups — replaces groupsForBranches

GET /v2/branch-groups?school=<code>&branch=2_4_5

Authentication: token. The groups of one branch or of several, sub-groups nested under their parent. branch takes an underscore-joined id list, so a caller walking a programme asks once rather than once per branch. Every group carries its own branchId, so a multi-branch answer stays unambiguous, and maxStudentNum — the group's capacity from the desktop, not a live count of who is in it.

```
[ { "id": "3", "name": "AFM1-3", "branchId": "4", "maxStudentNum": 30,
    "childGroups": [] },
  { "id": "10", "name": "BCIT1-1", "branchId": "5", "maxStudentNum": 30,
    "childGroups": [] } ]
```

branchId and maxStudentNum are new in revision 1.4, as is accepting more than one branch. Branch ids come from /v2/branches?school=&programme=&year=, or from the branchId of /v2/course-branches (12.1).

12.3 POST /v2/student-registration — replaces studentGroupsCourses

POST /v2/student-registration?school=<code>

Authentication: token, AND the account must be permitted to write student data. This is the only endpoint on v2 that changes anything outside room booking, and it can empty a student's enrolment, so it is granted per account rather than to everyone holding a valid token. An account without the permission gets 403 not_permitted. Ask Wise Technologies to enable it for the account you integrate with.

The body is a JSON array of students. For each one, EVERYTHING that student already had is removed and exactly what you send is written — you state the final state, you do not send a diff. A subject dropped in your system therefore disappears here without a deletion call.

```
[ { "studentNum": "123456",
    "groupIds": [101, 102],
    "courses": [ { "courseCode": "1397", "executionType": "LV" },
                 { "courseCode": "1397", "executionType": "PR" } ] } ]
```

courseCode is tbcourse.Code — the same code /v2/course-branches sends. executionType is matched against the teaching type's code OR its name, case-insensitively, because schools put the string they use in different columns; /v2/course-types lists what a given school accepts.

The response is one entry per item, in request order, each either created or carrying an error. Items are independent: a mistyped course code does not stop the groups from being written.

```
[ { "studentNum": "123456", "groupId": 101, "status": "created" },
  { "studentNum": "123456", "groupId": 102, "status": "created" },
  { "studentNum": "123456", "courseCode": "1397", "executionType": "LV",
    "status": "created" },
  { "studentNum": "123456", "courseCode": "1397", "executionType": "ONLINE",
    "error": "Invalid execution type" },
  { "studentNum": "789012", "error": "Student not found" } ]
```

One safeguard the legacy did not have: if a student's entry carries items and NOT ONE of them resolves, nothing is deleted and the errors are reported. A payload that is wholly wrong therefore cannot empty a registration. An entry with empty groupIds and empty courses still clears the student, because that is a deliberate instruction rather than a mistake.

Errors: Student not found, Group not found, Invalid group id, Course not found, Invalid execution type. A 503 writes_not_configured means the server has no writing account for that school's student tables yet — contact support; the message names the exact grants.

13 Changelog

Rev	Changes
1.5	Corrected: the API is no longer read-only — sections 1.3 and 3.2 now name the two POST endpoints documented in chapters 11 and 12. Support and credential requests now carry an address (info_wise@wise-t.com) instead of pointing at a representative.
1.4	SIS integration (chapter 12): /v2/course-branches, branch lists and maxStudentNum on /v2/branch-groups, POST /v2/student-registration; /v2/schedule without by/id returns the whole school; code and the three course flags on /v2/courses.
1.3	/v2/rooms also returns code, email, link and seats (room mailbox, room code, room page, capacity).
1.2	This supplement: foreignId/linkEvent/linkResolved/isOnline on events; POST /v2/eventlinks; stable-GUID guarantee documented.
1.1	Scoped /v2/schools; weekDays field in capabilities.
1.0	Initial manual.