EDUMATCHER • EDUCATIONAL TRADING SYSTEM

The EduMatcher API Gateway

REST and WebSocket access to the exchange — a beginner's guide

pm-api-gwy · Base path /api/v1 · Built for browser UIs, dashboards, bots and teaching examples

What this deck covers

AGENDA

1 Orientation
What the gateway is, what it is not, and where it sits in the system.

2 REST vs WebSocket
Two doors into the exchange, and when to use each one.

3 Getting connected
Configuration, starting the processes, Swagger and your first call.

4 API keys & access
Bearer tokens, read-only vs trading vs admin.

5 What you can send & receive
The six categories of information the gateway carries.

6 Payload structure
Requests, responses, events, pages — the shapes you will meet.

7 Error handling
The error envelope, the code catalogue, and how to react.

8 Client cookbook
Recipes for a trading bot, a dashboard and a research script.

9 Reference appendix
Full endpoint, channel, field and error listings.

01

Orientation

What the API gateway is for, and where it fits in the EduMatcher landscape

EduMatcher in one picture

THE BIG IDEA

An exchange is a machine that matches buyers with sellers. EduMatcher is that machine, built for learning — and the API gateway is the door your own software walks through.

Your client

Browser UI, dashboard, bot or script you write

›

API gateway

Translates HTTP and WebSocket into engine messages

›

Matching engine

Validates, matches and publishes every event

›

Market

Orders meet, trades print, prices move

REST

Ask a question, get an answer

WebSocket

Subscribe once, receive forever

JSON

One format everywhere

Why the gateway exists

PURPOSE

The problem

Talking to the engine directly is hard

- The engine speaks its own message format over ZMQ sockets
- Every new client would need to re-implement that transport
- Browsers cannot open raw exchange sockets at all
- Teaching material would drown in plumbing before reaching trading

The answer

One friendly, standard front door

- Plain HTTP and JSON — the vocabulary every developer already knows
- WebSocket streams for anything that must arrive the moment it happens
- API keys that map to exchange identities and permissions
- Interactive documentation you can click through in a browser

What it is — and what it is not

BOUNDARIES

It IS

- A translator: HTTP and WebSocket in, engine messages out
- A session cache holding your live orders, quote legs and fills
- A read path into recorded history and market statistics
- The same functionality the interactive console offers, exposed for software

It is NOT

- A second matching engine — it never matches an order itself
- A risk system with its own rules; the engine remains the authority
- A tick-by-tick market data feed replacement for high-rate consumers
- A production-hardened, secret-managed deployment out of the box

Who sits on the other side

AUDIENCE

Browser UIs

Trading screens built in any web framework — order tickets, blotters, book views.

Dashboards

Read-only wallboards showing prices, trades and session state. No trading rights needed.

Simple bots

Scripts that watch the market and place, amend or cancel orders automatically.

Teaching examples

Runnable Python and C samples that show a concept in a few dozen lines.

Research scripts

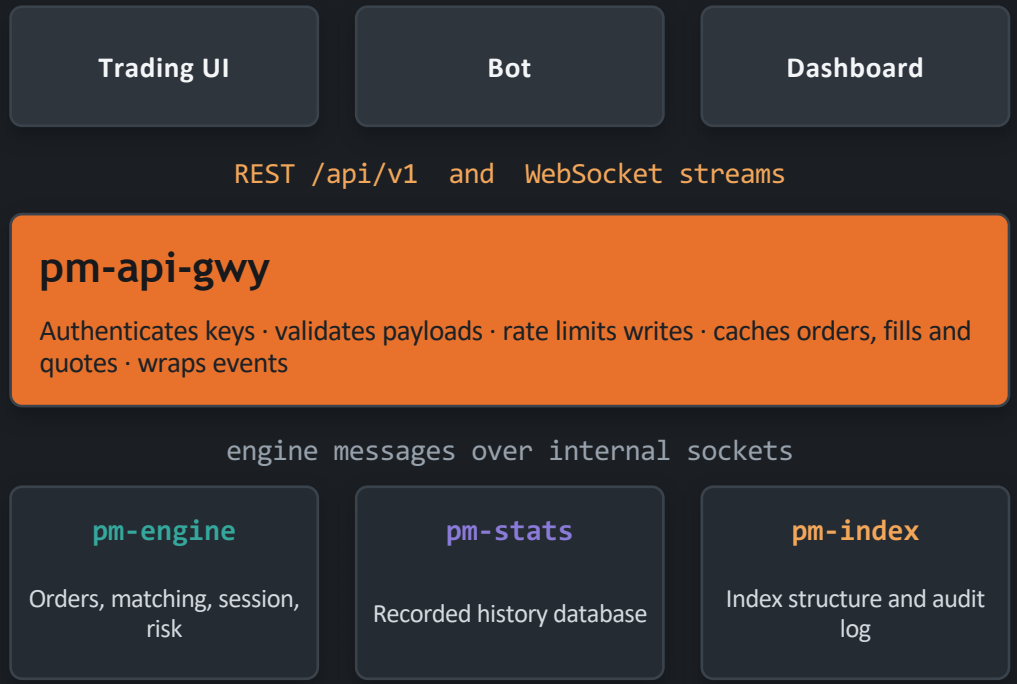
Notebooks pulling daily bars, snapshots and the trade tape for analysis.

Operations tools

Admin clients that move the session, halt a symbol or pull the kill switch.

Inside the gateway

ARCHITECTURE



Read it top to bottom

Your client only ever speaks HTTP and WebSocket.

The gateway is the only component that knows both worlds.

Live trading and market events come from the engine.

Anything historical is read from the statistics database.

Index structure and audit events are answered live by the index process.

The gateway among its siblings

PROTOCOL FAMILY

EduMatcher speaks several external protocols. The API gateway is the web-native one — the others are text or binary socket protocols for specialised consumers.

REST / WebSocket

This gateway. JSON over HTTP for any client, including browsers.

ALF

Human-readable text order entry. Interactive console, or a TCP gateway for bots.

BALF

Binary framed order entry for programmatic clients that want low latency.

CALF

Channelised market-data feed with snapshot, incremental and replay semantics.

RALF

Post-trade dissemination for clearing, drop-copy and audit consumers.

Choosing

Start here if you write software over HTTP. Move to CALF or BALF only when you need their specialisms.

Which protocol should I use?

SELECTION GUIDE

If you need to…	Start with	Why
Build a web or desktop trading UI	API gateway	JSON over HTTP, plus live WebSocket events
Write a simple bot or teaching script	API gateway	Few lines of code, ready-made client examples
Show a read-only price dashboard	API gateway	A read-only key streams public market data
Enter orders by hand at a terminal	ALF console	Interactive, readable command syntax
Run a low-latency programmatic strategy	BALF	Compact binary framing, less parsing overhead
Consume a full external market-data feed	CALF	Sequence-aware channels and recovery
Feed clearing, drop-copy or audit systems	RALF	Role-based post-trade streams with replay

Rule of thumb for beginners: if your program can make an HTTP request, the API gateway is the right starting point.

02

REST and WebSocket

Two doors into the exchange — and a simple rule for choosing between them

Two conversation styles

MENTAL MODEL

REST — the request

You ask, the gateway answers, the conversation ends

- One question, one answer, then the connection is done
- Perfect for actions you initiate: submit, cancel, amend, look up
- Every call carries your API key in a header
- Result arrives as an HTTP status code plus a JSON body
- Easy to test with a browser, curl or any HTTP library

WebSocket — the stream

You connect once and the exchange keeps talking

- The connection stays open for the whole session
- Perfect for things the exchange initiates: fills, price changes, halts
- You authenticate once with a first JSON message
- Events arrive whenever they happen — no polling
- One stream can feed an entire screen of live widgets

Push versus pull, illustrated

CONTRAST

REST · you pull

GET /status → 200 OK | GET /status → 200 OK | GET /status → 200 OK

You keep asking. Between two questions you are blind — and most answers are identical.

WebSocket · the exchange pushes

connect once | order.acked | book.update | order.fill | trade | session.state

Events arrive the instant they occur, in order, with no wasted requests.

Choosing between them

DECISION GUIDE

Your goal	Use	Endpoint family
Place, amend, cancel or replace an order	REST	/orders, /oco, /combos, /quotes
Know the moment your order fills	WebSocket	/events
Read instrument metadata or session state	REST	/symbols, /session
Keep a live order book on screen	WebSocket	/market-data (book, depth)
Show the running trade tape	WebSocket	/market-data (trades)
Chart yesterday's prices	REST	/history/daily, /history/price-snapshots
React to a trading halt or session change	WebSocket	/market-data (always delivered)
Recover state after a restart	REST	/orders, /positions, /status

Actions and lookups go over REST. Anything you would otherwise poll for belongs on a WebSocket.

Real clients use both

THE COMBINED PATTERN

1 · Open stream

Connect and authenticate the events WebSocket before you trade

2 · Snapshot

Call REST once for symbols, session and your live orders

3 · Act

Submit and cancel orders over REST as decisions are made

4 · Listen

Let acks, fills and price moves arrive on the stream

Why this order matters

Connect the stream first, then trade — otherwise the outcome of your first order can arrive before you are listening.

Use REST once at start-up to establish the current picture, then let the stream keep it current.

Re-snapshot over REST after any reconnect, so you never trade on a stale view.

Synchronous or fire-and-forget?

A REST SUBTLETY

Default — 202 Accepted

The gateway hands your order on and returns immediately

- Fast: your code is not blocked waiting for the engine
- The reply confirms receipt, not acceptance
- The real outcome arrives on the events stream
- Best for UIs and any client that already listens to events

Add ?wait=ack

The gateway waits for the engine's acknowledgement for you

- Simple: one call gives you the final answer
- Waits only for your own order, so parallel requests never cross
- Times out and returns an error if the engine stays silent
- Best for scripts, tests and step-by-step teaching examples

Cancel and amend accept ?wait=ack too. Cancel-replace always waits for the cancel before submitting the replacement.

03

Getting connected

From a configuration block to your first successful call

The five-minute path

OVERVIEW

Configure
Add an api_gateways block with a port and API keys

Start
Engine, then statistics, then the gateway process

Verify
Call the health endpoint — no key required

Explore
Open the interactive docs in a browser

Call
Send your first authenticated request

You will need
A running engine, a valid API key, and the gateway's host and port.

You will get
An HTTP server on the configured port serving /api/v1 and live WebSocket streams.

You will check
Health first, then an authenticated status call. If both pass, you are connected.

Configuration lives with the engine

SETUP

```
api_gateways:
  desk:
    enabled: true
    host: 0.0.0.0
    port: 8080
    swagger_enabled: true
    stats_db: data/stats.db
    credentials:
      - api_key: key-trader-demo
        gateway_id: TRADER01
        description: Demo trading client
      - api_key: key-dashboard-demo
        gateway_id: null
        description: Read-only dashboard
    rate_limit:
      writes_per_second: 10
      burst: 20
    timeouts:
      engine_auth_sec: 3.0
      engine_reply_sec: 3.0
      wait_ack_sec: 3.0
```

What matters here

The settings sit in the central engine configuration under api_gateways — the underscore form is the valid one.

Each named block is one gateway process, selected by name when it starts.

A credential with a gateway identity can trade; one without it is read-only.

The default bind address is open to the network, so browsers and dashboards on other machines can reach it.

Write limits and timeouts are per gateway, not per client.

Two identities, two levels of power

CREDENTIALS

Trading credential

gateway_id names a real exchange participant

- Can submit, amend, cancel and replace orders
- Sees its own live orders, positions and private events
- Reads its own historical orders and fills
- Each identity belongs to exactly one gateway process

Read-only credential

no gateway_id — a viewer, not a participant

- Streams public market data and session events
- Reads the public trade tape, daily bars and snapshots
- Cannot submit orders or see anyone's private state
- Safe to hand to a wallboard or a classroom demo

The engine keeps the final say: if its own allow-list does not recognise the identity behind a key, every request using that key is refused.

Starting the processes

RUN ORDER

pm-engine
The matching engine. Nothing trades without it.

›

pm-stats
Records history into the statistics database.

›

pm-api-gwy
The gateway itself, started with an instance name.

```
# installed
pm-engine --verbose
pm-stats
pm-api-gwy --instance desk
```

```
# from a source checkout
poetry run pm-engine --verbose
poetry run pm-stats
poetry run pm-api-gwy --instance desk
```

Order matters: the gateway greets the engine at start-up. Logs go to the console, so redirect them to a file when running unattended.

Command-line overrides

OPTIONS

Option	Default	What it does
`--instance NAME`	auto when only one block exists	Choose which configured gateway to run
`--host ADDR`	config value	Override the address the server binds to
`--port PORT`	config value	Override the listening port
`--engine-host HOST`	config value	Point the gateway at an engine on another machine
`--stats-db PATH`	config value	Choose the history database file
`--log-level LEVEL`	config value	debug, info, warning or error

Useful in a classroom: run several gateways side by side on different ports, each with its own configuration block and its own keys.

Your first two calls

HELLO, EXCHANGE

```
curl -s http://127.0.0.1:8080/api/v1/healthz
{"ok": true, "enabled": true, "active_gateways": ["TRADER01"]}
```

Health needs no key. It confirms the gateway is enabled and its event listener is alive — it does not prove the engine is up.

```
curl -s -H "Authorization: Bearer key-trader-demo" \
     http://127.0.0.1:8080/api/v1/status
→ your gateway role, cache summary and connection state
```

If health passes but status returns an authorisation error, the problem is the key or the identity behind it — not the network.

The interactive documentation

SWAGGER

Open it in a browser

`http://127.0.0.1:8080/docs`

- Every endpoint, with its request and response shape
- All permitted values for each field, listed explicitly
- An Authorize button — paste your bearer key once
- Try any call live and see the real response
- A machine-readable schema for generating client code

```
Authorize →
Bearer key-trader-demo
```

Treat it as your map

The pages that follow explain the ideas. The interactive documentation always shows the exact, current contract for the build you are running — when the two disagree, believe the running system. It can be switched off in configuration, so a gateway without it is not broken.

04

API keys and access

How you prove who you are — over REST and over a stream

One key, two ways to present it

AUTHENTICATION

REST — an HTTP header

Sent on every single request

- Add an Authorization header with the word Bearer and your key
- No login step and no session cookie to manage
- An unknown key is refused before anything else happens
- Only the health endpoint needs no key

```
Authorization: Bearer key-trader-demo
```

WebSocket — a first message

Sent once, right after connecting

- Open the socket, then send a small JSON object holding your key
- Authentication is confirmed before any event flows
- A rejected key is disconnected immediately
- The connection then stays authenticated for its lifetime

```
{ "api_key": "key-trader-demo" }
```

What a key is allowed to do

PERMISSION LEVELS

Read-only

Public market data, the trade tape and history. No orders, no private state.

Trading

Everything read-only can do, plus order entry, management, positions and its own history.

Admin

Everything trading can do, plus session control, halts and the cross-gateway monitor stream.

Capability	Read-only	Trading	Admin
Public market data and trade tape	yes	yes	yes
Daily bars, snapshots, index history	yes	yes	yes
Submit, amend and cancel orders	no	yes	yes
Own live orders, positions and fills	no	yes	yes
Private lifecycle event stream	no	yes	yes
Session transitions, halts, kill switch	no	no	yes
Cross-gateway monitor stream	no	no	yes

Where the admin role comes from

ROLES

Your key

Names an exchange identity in the gateway configuration

The engine

Holds the roles: trader, market maker or admin

First admin call

The gateway asks the engine once and remembers the answer

Decision

Allowed, or refused with a role error

Why it works this way

Roles are a property of the exchange participant, not of the API key. That keeps one source of truth: change an identity's role at the exchange and every client using it follows automatically. The trade-off is a single extra round trip on your first administrative call.

Handling keys sensibly

PRACTICE

Keep them out of code

Read the key from an environment variable or a local settings file — never paste it into a source file you will share.

Never ship them to a browser

A key in front-end JavaScript is public. Put a small server of your own in between.

One key per purpose

A separate key per bot, screen or classroom group makes rate limits and audit trails meaningful.

Prefer the weakest key

If a client only displays prices, give it a read-only key. It then cannot trade by accident.

Expect plain text in labs

Teaching setups keep keys in the configuration file. That is deliberate — and not a production pattern.

Wrap for production

Terminate encryption in front of the gateway and let your platform manage the secrets.

The client library pattern

REUSE

```python
# Python
from api_gateway_client import \
     ApiGatewayClient

client = ApiGatewayClient(
   "http://127.0.0.1:8080",
   "key-trader-demo")

print(client.get_json("/api/v1/status"))
print(client.get_json("/api/v1/symbols"))
```

```c
/* C */
ApiGatewayClient client =
  api_gateway_client(
    "127.0.0.1", 8080,
    "key-trader-demo");

char *body = api_gateway_get(
   &client, "/api/v1/status");
puts(body);
free(body);
```

Wrap the key once

One small client object holds the address and the key so no call has to repeat them.

Runnable samples ship with the system

Both languages have working examples in the documentation's example folders.

Override without editing code

The sample tools read the address and key from environment variables.

05

What flows through

The categories of information you can send and receive

Six categories of information

MAP

1 Order entry

Send intent: single orders, paired orders, multi-leg combos and two-sided quotes.

2 Order management

Change or withdraw intent: amend, replace, cancel one, cancel everything.

3 Reference data

The rules of the game: instruments, session state, your gateway's status.

4 Market data

What the market is doing right now: book, depth, trades, auctions, halts.

5 History

What already happened: your orders and fills, the public tape, daily bars, index levels.

6 Administration

Running the exchange: session transitions, circuit breakers, gateway control.

Direction of travel

SEND AND RECEIVE

You send

Always a request you initiate

- Orders and quotes you want in the market
- Amendments, cancellations and replacements
- Queries for instruments, session state and your own orders
- History queries with filters, ranges and page cursors
- Subscription instructions on the market-data stream
- Administrative commands, if your identity is permitted

You receive

Answers you asked for, and events you did not

- Immediate confirmations that a request was accepted
- Order lifecycle events: acknowledged, amended, cancelled, filled
- Book, depth, trade and auction updates for your subscriptions
- Session changes and circuit-breaker notices, always delivered
- Pages of historical rows with a cursor to continue
- Errors carrying a code that tells you how to react

Order entry, beyond a single order

CATEGORY 1

Single order

One instrument, one side, one quantity. The building block everything else extends.

OCO pair

Two linked exits — a target and a protective stop. When one triggers, the other is withdrawn.

Combo

Several legs submitted as one instruction, for spreads across instruments.

Two-sided quote

A bid and an offer posted together, the market maker's way of showing a price.

Mass cancel

Withdraw everything, or everything in one instrument, in a single call.

Kill switch

The same withdrawal under an emergency name, for the moment something goes wrong.

Every one of these is a single HTTP call with a small JSON body.

The life of an order

CATEGORY 2

Submit

You create the order and receive an identifier

Accepted

The engine accepts it and it becomes live

Amend

Adjust price, quantity, or both

Fill

Wholly or partly matched against the other side

Cancel

Withdrawn by you, or by an exchange event

Amend vs replace

Amending edits the order in place. Replacing withdraws it and submits a fresh one — a new identifier and a new place in the queue.

Reading state

List your live orders, or read one from the gateway's own cache without troubling the engine.

Client identifiers

Give each order your own reference. Reusing one that is still active is rejected as a duplicate.

Reference and account data

CATEGORY 3

Instruments

The tradable universe and each instrument's metadata. Fetch it once at start-up.

Session

Whether the market is pre-open, auctioning, trading continuously or closed — plus the schedule.

Positions

Net quantity per instrument, computed from the fills the gateway has seen this session.

Live orders

Everything of yours currently working in the market.

Quote state

Your active two-sided quotes and their individual legs.

Status

The gateway's own summary: your resolved role and what its caches currently hold.

Positions and cached orders are session-scoped: they describe what this gateway has observed since it started.

Live market data

CATEGORY 4

book

The best prices on each side, updating as orders arrive and leave.

trades

Every execution that prints — the public tape.

depth

Volume available at each price level, for a fuller picture of the book.

auction

The outcome when an auction uncrosses at the open or the close.

Two things you never have to ask for

Session changes — the market opening, auctioning or closing — are delivered to every authenticated stream.

Circuit-breaker events — an instrument halted or resumed — arrive the same way.

Instrument filtering is opt-in: name the instruments you care about, or receive every one on the channels you subscribed to.

Recorded history

CATEGORY 5

Private history

Needs a trading key; always scoped to you

- Your order lifecycle events, filtered by instrument, type or date
- The complete story of one order, from submission to its end
- Your fills, for reconciliation and performance work

Public history

Any valid key, including read-only

- The trade tape, as printed to the whole market
- Daily open, high, low, close and volume rows
- Intraday price snapshots recorded at a fixed interval
- Index levels, daily and intraday, plus structural events

History comes from the recording process. If it has never run, the history endpoints report that the database is not available yet — that is a setup message, not a fault in your client.

Administration

CATEGORY 6

Move the session

Request a transition to pre-open, auction, continuous trading or closed.

Read the schedule

See whether scheduled sessions are enabled and what the timetable is.

List gateways

Every configured participant, its role, description and whether it is connected.

Disconnect one

Drop a participant's connection when it misbehaves.

Halt and resume

Trigger a circuit breaker on an instrument, then bring it back.

Symbol kill switch

Cancel all exposure in one instrument across the exchange.

Adding instruments and editing the index are deliberately not exposed here — those live with dedicated tools.

06

The shape of the data

Requests, responses, events and pages — five envelopes to recognise

Five shapes to learn

OVERVIEW

1 The request

A flat JSON object of fields, sent with your key in a header.

2 The accepted reply

A small confirmation with an identifier and a status.

3 The list envelope

Rows plus a count, a more flag and a cursor to continue.

4 The event envelope

A type, a timestamp, an identity and a data block.

5 The error envelope

An HTTP status paired with a stable machine-readable code.

One rule

Everything is JSON, in both directions, on both REST and the streams.

A request, field by field

SHAPE 1

```
POST /api/v1/orders?wait=ack
Authorization: Bearer key-trader-demo
Content-Type: application/json

{
  "symbol": "AAPL",
  "side": "BUY",
  "order_type": "LIMIT",
  "quantity": 100,
  "tif": "DAY",
  "price": 150.50,
  "smp_action": "NONE",
  "client_order_id": "ui-42"
}
```

How to read it

Instrument, side, type and quantity are always required.

Price is required only for order types that need one — a market order must not carry one at all.

Time in force says how long the order should live; it defaults to the trading day.

Self-match prevention decides what happens if you would trade with yourself.

Your own reference travels with the order and comes back on its events.

Conditional fields, explained

SHAPE 1 CONTINUED

Field	When you must include it	Meaning in plain words
`price`	Limit-style orders	The worst price you are willing to accept
`stop_price`	Stop and stop-limit orders	The trigger level that wakes the order up
`visible_qty`	Iceberg orders	How much of the size the market may see
`trail_offset`	Trailing stops	How far behind the market the trigger follows
`tif`	Optional	How long the order lives if it does not trade
`smp_action`	Optional	What to do if your two orders would match each other
`client_order_id`	Optional but recommended	Your own reference, unique while the order is live

The gateway checks these rules before the engine ever sees your order, so mistakes come back quickly and describe themselves.

What comes back from a write

SHAPE 2

Without waiting

Accepted — the request is on its way

- You receive an order identifier and a pending status
- The engine's verdict arrives later on the events stream
- Your code stays responsive; nothing blocks

Waiting for the acknowledgement

The gateway holds the reply until the engine answers

- You receive the same identifier plus the engine's decision
- Accepted or refused — and if refused, the reason
- Or a timeout error if the engine does not answer in time

```
order_id  : ORD-3a7f1e2c        order_id  : ORD-3a7f1e2c
status    : PENDING             status    : ACKED
                                accepted  : true
```

Lists arrive in pages

SHAPE 3

```
GET /api/v1/history/trades?symbol=EDU100&limit=2

{
  "trades": [ { ... }, { ... } ],
  "count": 2,
  "has_more": true,
  "next_cursor": "eyJyb3dpZCI6MiwidHM..."
}
```

The three signals

count — how many rows this page holds

has_more — true when the page came back full, so more may exist

next_cursor — an opaque marker; hand it straight back to continue

Ask — Send a query with a page size

Read — Use the rows you received

Continue — If more exist, resend with the cursor

Stop — When no more is reported, you have everything

Why cursors, not page numbers

SHAPE 3 CONTINUED

The trap with page numbers

- New rows are written while you are reading
- Everything shifts down by one position
- Page two now repeats a row you already saw — or skips one
- Nobody notices until the totals disagree

How cursors avoid it

- The cursor remembers where you stopped, not how far you counted
- The next page continues from that exact point
- No duplicates and no gaps, even under live writing
- Treat the cursor as opaque — never take it apart

Three exceptions: a single order's full lifecycle, the list of index identifiers, and the index audit log are all returned complete, with no paging. Each is naturally small or naturally bounded by its own limit.

Events look the same every time

SHAPE 4

```
{
  "type": "order.fill",
  "ts": "2026-06-24T10:15:03.221Z",
  "gateway_id": "TRADER01",
  "data": {
    "order_id": "ORD-...",
    "fill_qty": 50,
    "fill_price": 150.50,
    "remaining_qty": 50,
    "status": "PARTIAL"
  }
}
```

Four parts, always

type — what happened, in a dotted name you can switch on

ts — when it happened

gateway_id — whose event it is

data — the details, which vary by type

Write your client as a single loop: read the type, route it to a handler, use the data. A consistent envelope means adding support for a new event never means rewriting the plumbing.

Subscribing to market data

STREAMS IN PRACTICE

```
{ "api_key": "key-dashboard-demo" }

{ "action": "subscribe", "symbols": ["AAPL", "MSFT"],
  "channels": ["book", "trades", "depth", "auction"] }
```

Authenticate first

The key message must be the very first thing you send on the socket.

Then subscribe

Name the channels you want. Ask only for what you will actually display.

Filtering is optional

Leave the instrument list out and you receive every instrument on those channels.

Session and circuit-breaker events arrive regardless of what you subscribed to — you cannot miss the market closing.

07

When things go wrong

One error shape, a small vocabulary of codes, and the right reaction to each

Errors are data, not surprises

PHILOSOPHY

Two signals in every failure

- An HTTP status code — the broad category, understood by every tool
- A stable error code in the body — the specific cause, meant for your code
- Branch on the code, and show the human-readable detail to the user
- Never parse the message text; it may be reworded, the code will not

Three broad families

- It is you — the request was refused: authentication, permission, validation
- It is timing — you were too fast, or the same reference twice
- It is us — a component behind the gateway was slow, silent or unhappy

```
HTTP 429   { "error": { "code": "RATE_LIMIT", "detail": "..." } }
```

The status code at a glance

CATEGORIES

401

Unauthorised — no key, a malformed header, or a key nobody recognises.

403

Forbidden — the key is real, but this identity may not do this.

409

Conflict — you reused a client reference that is still active.

422 / 400

Invalid — the body or the query parameters do not satisfy the rules.

429

Too many requests — your write rate limit for this key was exceeded.

502 / 503

Upstream trouble — a component behind the gateway failed or stayed silent.

A useful split: 4xx means change your request; 5xx means retry sensibly or wait for the system to recover.

The error codes you will actually meet

CATALOGUE

Code	Status	What it really means
AUTH	401	The key is missing, malformed or unknown
ENGINE_AUTH	403	The exchange does not allow this identity
READ_ONLY	403	A viewer key tried to trade
ROLE_DENIED	403	The identity lacks the administrative role
RATE_LIMIT	429	Too many writes with this key, too quickly
DUPLICATE	409	That client reference is already live
VALIDATION	422 / 400	The request body or parameters are wrong
STATS_DB	503	History has not been recorded yet
ENGINE_TIMEOUT	503	The engine did not answer in time
INDEX_TIMEOUT	503	The index process did not answer in time
INDEX_ERROR	502	The index process rejected the request

How to react to each family

PLAYBOOK

Fix and retry never

Authentication, permission and validation failures will fail identically forever. Correct the key, the identity or the payload.

Slow down

On a rate-limit refusal, pause and retry with increasing delays. Bunch your writes less aggressively.

Use a fresh reference

On a duplicate, generate a new client reference rather than resending the old one.

Retry carefully

A timeout does not mean the order was rejected. Check your live orders before resubmitting.

Wait for setup

An unavailable history database means the recorder has not run. Nothing in your client can fix that.

Surface it

Show the human-readable detail in your interface, and log the code alongside it for later.

The most common first-day problems

TROUBLESHOOTING

Symptom	Likely cause	What to do
Connection refused	Gateway not started, or wrong port	Confirm the process is running and check the configured port
Health reports not ok	Disabled in configuration, or its listener stopped	Check the enabled flag and the gateway's own logs
Unauthorised on every call	Missing or wrong bearer header	Send the key exactly as configured
Forbidden on order entry	Using a read-only key	Switch to a key with a trading identity
Not found on every path	Wrong base path or wrong instance	Check the instance name matches the configuration block
Documentation page will not load	Interactive docs switched off	Enable them in configuration and restart
History comes back empty	Recorder not running, or wrong database path	Start the recorder and verify the path
Stream disconnects at once	Engine not running, or a limit was hit	Start the engine and read the disconnect reason in the logs

Checking connectivity from the shell

DIAGNOSTICS

```
# is anything listening?
ss -tlnp 'sport = :8080'
sudo lsof -iTCP:8080 -sTCP:LISTEN
netstat -an | grep 8080
```

```
# does it answer?
curl -s .../api/v1/healthz
curl -s -H "Authorization: Bearer KEY" \
     .../api/v1/status
```

```
# does the stream upgrade?
websocat ws://127.0.0.1:8080/api/v1/market-data
```

Work outward: port, then health, then an authenticated call, then the stream. The first step that fails tells you which layer to investigate — and saves you guessing.

08

Client recipes

Three complete clients, described end to end

Recipe 1 — a trading bot

COOKBOOK

1 · Listen

Open the private events stream and authenticate

2 · Learn

Fetch instruments and session state

3 · Decide

Apply your own logic to incoming prices

4 · Send

Submit orders with your own reference

5 · React

Handle acknowledgements, fills and cancels

Do

Connect the stream before your first order. Use a fresh reference each time. Cancel everything on shutdown.

Don't

Poll your order list in a tight loop, or treat a timeout as proof the order never reached the market.

Remember

Writes are rate limited per key, so pace bursts rather than firing everything at once.

Recipe 2 — a live dashboard

COOKBOOK

1 · Read-only key

No trading rights, nothing to lose if the screen is public

2 · Market stream

Connect, authenticate, subscribe to the channels you draw

3 · Choose

Name your instruments, or take the whole market

4 · Render

Update widgets as events arrive; never poll

Prices and trades

The book channel drives the top of book; the trade channel drives the tape.

Market state

Session and halt events arrive automatically — use them to grey out a halted instrument.

Charts

Draw the history from recorded daily bars and snapshots, then extend it live from the stream.

Recipe 3 — a research script

COOKBOOK

```
GET /api/v1/history/daily
    ?symbol=AAPL&from=2026-06-01&to=2026-06-30
Authorization: Bearer key-readonly-demo
```

One call, a whole month

Ask for a date range and you receive every day in it, oldest first. Without a range you get only the most recent day — a month would then cost a request per day.

Daily bars

Open, high, low, close and volume, per day.

Intraday snapshots

Mid, bid and ask at a fixed interval.

The public tape

Every printed trade, filterable and paged.

Index levels

Index history plus the structural log.

Loop the cursor

Keep asking while more is reported.

Mind the freshness

Today's close moves until the session ends.

A minimal order from the command line

WORKED EXAMPLE

```
cd docs/examples/REST/python
python3 submit_market_order.py --side BUY  --symbol AAPL --qty 100
python3 submit_market_order.py --side SELL --symbol MSFT --qty 50 --wait-ack
```

Without waiting

- An identifier comes back immediately
- The status reads pending
- The outcome follows on the events stream

With waiting

- The same identifier, plus the engine's verdict
- Accepted, with a reason when it is not
- The complete acknowledgement is printed

A market order must not carry a price — including one is refused as invalid before it leaves the gateway.

Habits of a well-behaved client

GOOD PRACTICE

1 Snapshot, then stream

Establish the picture over REST, keep it current from events.

2 Reconnect deliberately

After any drop, re-authenticate, re-subscribe and re-snapshot.

3 Reference every order

Your own identifier makes reconciliation and duplicate detection possible.

4 Respect the limits

Pace writes; treat a rate refusal as a signal, not a failure.

5 Handle every code

A default branch that logs the unknown code beats a silent crash.

6 Shut down cleanly

Cancel your exposure before your process exits.

Before you call it done

OPERATIONAL CHECKLIST

1. Every key maps to an identity the exchange actually allows
2. Each trading identity is used by exactly one gateway process
3. Engine, then recorder, then gateway — started in that order
4. The interactive documentation opens, if it is enabled
5. The health endpoint answers without a key
6. An authenticated status call succeeds with your key
7. The events stream is connected before the first order is sent
8. History is only used while the recorder is running

09

Finding out more

Where the detail lives when this deck runs out

Four places to look

DOCUMENTATION

The interactive docs

The exact contract for the running build: every endpoint, every field, every permitted value — and a button to try it.

The written guide

Configuration, start-up, authentication, endpoint reference, troubleshooting and design notes.

The example folders

Runnable Python and C clients you can copy as the starting point for your own.

The protocol appendices

Formal wire references for the sibling protocols, when you outgrow HTTP.

When a document and the running system disagree, believe the running system — then report the difference.

What to take away

SUMMARY

1 One door

The gateway turns the exchange into an ordinary web service you already know how to call.

2 Two styles

Ask over REST; listen over WebSocket. Never poll for something that can be pushed.

3 One key

A bearer token in a header, or a first message on a stream — and it decides what you may do.

4 Six categories

Order entry, order management, reference data, market data, history and administration.

5 Five shapes

Request, acceptance, page, event and error. Learn them once, reuse them everywhere.

6 Errors that teach

A stable code with every failure tells your client exactly how to respond.

10

Technical appendix

Complete listings for reference — endpoints, channels, fields and codes

Where API keys actually come from

APPENDIX · 1 OF 16

There is no sign-up page and no self-service portal. A key exists because whoever runs the exchange wrote it into the gateway's configuration — so getting one means asking that person, or editing the file yourself in your own lab.

```
api_gateways:
  desk:
    credentials:
      - api_key: key-trader-demo
        gateway_id: TRADER01
        description: Demo trading client
      - api_key: key-dashboard-demo
        gateway_id: null
        description: Read-only dashboard
```

Anatomy of one credential

api_key — the literal string your client sends. Any text you choose; the demo ones are simply named to be readable.

gateway_id — the exchange identity the key acts as, or empty for a viewer.

description — a human label, for your own benefit only.

The key is created, not generated: adding those three lines and restarting the gateway is the entire issuing process. Nothing is stored in a database and nothing is registered anywhere else.

Why there are no passwords

APPENDIX · 2 OF 16

A usual login system

Several moving parts

- A user name and a secret password
- A login call that exchanges them for a session token
- Tokens that expire and must be refreshed
- Server-side session state to keep track of

```
Authorization: Bearer key-trader-demo
```

This gateway

One string, and nothing else

- The API key is the whole credential — it names you and proves it
- No login call, no session, no refresh, nothing to expire
- Every request carries the key; the gateway checks its list
- The same string works for REST and for streams

```
{ "api_key": "key-trader-demo" }
```

Two lists must agree

APPENDIX · 3 OF 16

A key that looks perfectly correct still fails if the identity behind it is unknown to the engine. Authentication happens twice, in two different files.

1 · The key list

The gateway's credentials decide whether the string is recognised at all

2 · The identity list

The engine's own allow-list decides whether that identity may connect, and with which role

3 · The verdict

Both must agree before a single order can be sent

Unknown string

Refused by the gateway with an authentication error before anything else happens.

Known string, unknown identity

Refused with an engine authentication error — the key is fine, the identity is not allowed.

Known both, wrong power

Accepted, then refused per call: read-only keys cannot trade, non-admins cannot administer.

Each trading identity may be used by only one gateway process; viewer credentials may be shared freely.

Getting a key that works, step by step

APPENDIX · 4 OF 16

1 Decide what the key is for

Displaying prices only, trading, or running the exchange. This single choice fixes everything that follows.

2 Choose the identity

For trading, pick an exchange identity that exists in the engine's allow-list with the role you need. For a viewer, leave it empty.

3 Add the credential

Write the key string, the identity and a description into the gateway's configuration block.

4 Restart the gateway

The credential list is read at start-up, so a new key only exists after a restart.

5 Prove the gateway is up

Call the health endpoint. It needs no key, so a failure here is never a key problem.

6 Prove the key works

Call the status endpoint with the key. A successful reply confirms both lists agree — you now have a working key.

```
curl -s -H "Authorization: Bearer YOUR-KEY" http://HOST:PORT/api/v1/status
```

Reading the refusal

APPENDIX · 5 OF 16

Each failure names exactly which of the two lists rejected you, so a key problem never needs guesswork.

What you see	What it means	What to change
`AUTH, 401`	The gateway does not know this string	Check for a typo, or that the gateway was restarted after the key was added
`ENGINE_AUTH, 403`	The string is known, the identity is not allowed	Add the identity to the engine's allow-list, or point the key at one that exists
`READ_ONLY, 403`	A viewer key called a trading endpoint	Use a key that carries an identity
`ROLE_DENIED, 403`	The identity lacks the administrative role	Grant that role at the engine, or use an admin identity
`Health fails as well`	Not a key problem at all	Start the gateway, or correct the address and port

Rotating a key is the same operation as creating one: edit the string in the configuration, restart, and hand the new value to the client. Nothing else has to be revoked.

Handing the key to your client

APPENDIX · 6 OF 16

```
# never in source control
EDUMATCHER_API_URL=http://127.0.0.1:8080 \
EDUMATCHER_API_KEY=key-trader-demo \
python3 submit_market_order.py --side BUY ...
```

```
# in the interactive documentation
Authorize →
Bearer key-trader-demo
```

From the environment

The supplied example clients read the address and the key from environment variables, so no key ever has to be typed into a file.

Through the docs page

Paste the key once behind the Authorize button and every trial call on the page carries it.

Never in the browser

A key shipped to a web page is public. Keep it on a small server of your own and let the page talk to that.

Because a key is the whole credential, anyone holding the string has exactly the power that key was given — treat it as you would a password.

REST endpoints — order entry

APPENDIX · 7 OF 16

Method	Path	Purpose
POST	/orders	Submit one order
DELETE	/orders/{order_id}	Cancel one order
PATCH	/orders/{order_id}	Amend price and/or quantity
POST	/orders/{order_id}/replace	Cancel, then submit a replacement
POST	/oco	Submit a one-cancels-other pair
DELETE	/oco/{oco_id}	Cancel an OCO pair
POST	/combos	Submit a multi-leg combo order
DELETE	/combos/{combo_id}	Cancel a combo
POST	/quotes	Submit a two-sided quote
DELETE	/quotes/{symbol}	Cancel the quote for one instrument
POST	/mass-cancel	Cancel all, or one instrument's, exposure
POST	/kill-switch	Alias of mass cancel

All paths are relative to the base path /api/v1 and require a trading credential.

REST endpoints — state and reference

APPENDIX · 8 OF 16

Method	Path	Returns
GET	/orders	Live orders for the caller's identity
GET	/orders/{order_id}	One order, read from the gateway cache
GET	/symbols	Instrument metadata, from the engine
GET	/session	Current session state and schedule
GET	/quotes/bootstrap	Active quote bootstrap state
GET	/quotes/legs	Quote leg state, cached or from the engine
GET	/positions	Net position per instrument, from cached fills
GET	/status	Cache summary, resolved role, and gateway count for admins
GET	/healthz	Liveness probe — no authentication, not in the docs page

Endpoints that round-trip to the engine report a timeout error if it does not answer within the configured reply window. Reading one order returns a plain not-found body rather than the usual error envelope.

REST endpoints — history

APPENDIX · 9 OF 16

Path	Access	Contents
`/history/orders`	trading	Your order lifecycle events
`/history/orders/{order_id}`	trading	One order's full lifecycle, unpaginated
`/history/fills`	trading	Your fills
`/history/trades`	any key	Public trade tape
`/history/daily`	any key	Daily open, high, low, close and volume
`/history/price-snapshots`	any key	Intraday mid, bid and ask series
`/history/index-daily`	any key	Daily index open, high, low and close
`/history/index-snapshots`	any key	Intraday index level series
`/history/index-ids`	any key	Index identifiers with recorded data
`/history/index-events`	any key	Index structural and audit log, live from the index process

Private history is always scoped to the calling identity. All of these read the recorded database, except the index audit log, which is answered live by the index process.

History query parameters

APPENDIX · 10 OF 16

Endpoint	Parameters
`/history/orders`	symbol, event_type, date, from, to, limit (1–5000, default 500), after
`/history/orders/{order_id}`	none — path parameter only
`/history/fills`	symbol, date, from, to, limit, after
`/history/trades`	symbol, date, from, to, limit, after
`/history/daily`	symbol, date, from, to, limit, after
`/history/price-snapshots`	symbol (required), date, from, to, limit, after
`/history/index-daily`	index_id, date, from, to, limit, after
`/history/index-snapshots`	index_id (required), date, from, to, limit, after
`/history/index-ids`	date
`/history/index-events`	index_id (required), from, to, types, max_records (max 10,000)

Ranges on the daily tables are calendar dates; ranges on the tape and snapshot tables are timestamps; ranges on the index audit log are seconds since the epoch. Omitting every filter on a daily table returns the latest date only.

Admin endpoints

APPENDIX · 11 OF 16

Method	Path	Body	Result
POST	/admin/session/transition	target state	Accepted, pending — applied by the engine
GET	/admin/session/schedule	none	Whether sessions are scheduled, and the timetable
GET	/admin/gateways	none	Identity, role, description, connection state
POST	/admin/gateways/{gid}/disconnect	none	That participant is disconnected
POST	/admin/circuit-breaker/trigger	instrument	Halt acknowledged by the engine
POST	/admin/circuit-breaker/resume	instrument	Resume acknowledged by the engine
GET	/admin/halts	none	Currently halted instruments
POST	/admin/kill-switch/symbol	instrument	All exposure in that instrument cancelled

These require the administrative role, resolved from the exchange rather than from the key. Session transitions do not wait for confirmation; halts and kill switches do. Writes are rate limited like any other write.

WebSocket streams and channels

APPENDIX · 12 OF 16

Path	Carries	Who may connect
`/api/v1/events`	Private order, quote and risk lifecycle events	Trading credential
`/api/v1/market-data`	Public book, trades, depth, auction, session and halts	Any valid key
`/api/v1/admin/monitor`	Every event across every participant	Administrative role only

book

Best prices on each side

trades

Executions as they print

depth

Volume by price level

auction

Uncross results per instrument

The first message on any stream must carry the key. The monitor stream confirms authentication, then streams everything; a non-administrative key is refused and disconnected. Session and circuit-breaker events are always delivered.

Order fields in full

APPENDIX · 13 OF 16

Field	Required	Notes
`symbol`	yes	Instrument symbol
`side`	yes	BUY or SELL
`order_type`	yes	MARKET, LIMIT, STOP, STOP_LIMIT, FOK, ICEBERG, IOC, TRAILING_STOP
`quantity`	yes	Positive whole number
`tif`	no	DAY, GTC, ATO, ATC — defaults to DAY
`price`	conditional	Required for LIMIT, FOK, IOC, ICEBERG and STOP_LIMIT
`stop_price`	conditional	Required for STOP and STOP_LIMIT
`visible_qty`	conditional	Required for ICEBERG; smaller than the quantity
`trail_offset`	conditional	Required for TRAILING_STOP
`smp_action`	no	Self-match prevention action
`client_order_id`	no	Your own reference; must be unique while active

Market orders must carry neither a price nor a stop price; including either is refused as invalid.

Payload examples

APPENDIX · 14 OF 16

POST /oco

```
{ "oco_id":"tp-sl-1", "symbol":"AAPL", "quantity":100,
  "leg1":{"side":"SELL","order_type":"LIMIT","price":152.0},
  "leg2":{"side":"SELL","order_type":"STOP","stop_price":147.0} }
```

POST /combos

```
{ "combo_id":"spread-1", "legs":[ {"symbol":"AAPL","side":"BUY","quantity":100,"price":150.0},
                                  {"symbol":"MSFT","side":"SELL","quantity":100,"price":410.0} ] }
```

POST /quotes

```
{ "symbol":"AAPL", "bid_price":150.0,
  "bid_qty":500, "ask_price":150.1, "ask_qty":500 }
```

POST /mass-cancel

```
{ "symbol":"AAPL" }   or   {}  for everything
```

Design decisions worth knowing

APPENDIX · 15 OF 16

One configuration file

Gateway settings live with the engine's configuration, following the same pattern as the other gateways.

Identities are exclusive

A trading identity belongs to one gateway process, so private session state is never split across two.

Waiting is targeted

A waiting request resolves on the event matching its own order, so parallel calls never take each other's answer.

Replace is cancel-then-submit

The cancel is confirmed before the replacement is sent, and the replacement carries a full order body.

Events stay close to the engine

Outbound events are wrapped in a consistent envelope but otherwise keep the engine's own payloads.

History is optional at start-up

A missing history database does not stop the gateway; only the history endpoints are affected.

Known limits and edges

APPENDIX · 16 OF 16

Not exposed here

- Adding or editing instruments while the engine runs
- Index administration: corporate actions and constituent changes
- These live with dedicated tools, not with this gateway

Behavioural edges

- Reading one order by identifier returns a plain not-found body, unlike every other error
- A halt level may be supplied but is not currently forwarded
- Today's index close keeps moving until the session is marked closed
- Snapshots are recorded periodically, so they are not a substitute for a live feed

For teaching and local laboratories the keys sit in plain text. Any real deployment belongs behind encryption, with secrets managed by the surrounding platform.

EDUMATCHER API GATEWAY

Start here

Open the interactive documentation, authorise with a demo key, and call the status endpoint. Everything in this deck follows from that first successful request.

`http://127.0.0.1:8080/docs`