Marketech Commercial API v1

> Batch update (30 Aug 2026): request up to 100 explicit project IDs with `view=detail` for extended core fields. Related collections can be fetched separately for up to 25 project IDs with `GET /projects/resources`. Existing summary and single-project endpoints remain unchanged. Both batch responses report their weighted cost in `meta.quota_units` and `X-Quota-Units`.

> Commercial account update (25 Aug 2026): subscriptions use client-specific inventory and access-point configurations. Agreement discounts and promotion codes are managed in the authenticated portals; they do not change the API contract below. Requested changes become effective only after approval and payment confirmation.

Quick start

Production base URL: `https://api.uaeprojects.com/v1`.

Send the one-time API key as `Authorization: Bearer up_live_...` (or `up_test_...`). Never put keys in query strings or browser-side code. Every response includes `X-Request-Id`; provide it when requesting support.

```
curl --fail-with-body 'https://api.uaeprojects.com/v1/projects?community=dubai-marina&per_page=25' \
  -H 'Authorization: Bearer up_live_REPLACE_ME' \
  -H 'Accept: application/json'
```

Health check

`GET /v1/health` reports service status — useful for uptime monitors or a pre-flight check before a batch job. It does not require an `Authorization` header and is not counted against your plan's quota.

```
curl https://api.uaeprojects.com/v1/health
```

Returns `200` with `{ "data": { "status": "ok", "version": "v1", "checks": { "database": true, "cache": true } } }` when healthy, or `503` with `"status": "degraded"` and the failing check(s) set to `false` otherwise.

Response and errors

Successful responses use `{ "data": ..., "meta": { "request_id": ..., "generated_at": ... } }`. Lists add `meta.pagination`. Errors use `{ "error": { "code": ..., "message": ..., "details": ... }, "meta": ... }`; validation failures are HTTP 422, authorization failures 401/403, quota failures 429, and unavailable resources 404.

Rate responses expose `RateLimit-Limit`, `RateLimit-Remaining`, `X-Quota-Daily-*`, and `X-Quota-Monthly-*` where applicable. A 429 includes `Retry-After` when a bounded retry is possible.

Rate limits and quotas

Limits are set per commercial plan and can be customized per client, so the exact numbers differ by contract —**always read the response headers rather than hard-coding a number**. A new client's default plan typically starts around 60 requests/minute with a daily/monthly allowance sized to the contract; there is no universal fixed ceiling. Build client-side throttling that backs off using `RateLimit-Remaining` (and `Retry-After` on a `429`) instead of assuming a specific limit, so your integration keeps working unchanged if your plan's limits are later adjusted. For bulk syncing many records, prefer `GET /projects?ids=...` (see below) or `GET /projects/changes?updated_since=...` over looping one-by-one, since both reduce the number of requests needed.

There are four independent limits, each returning its own `429` error code so you know exactly which one you hit:

Code	Window	Resets
`BURST_LIMIT_EXCEEDED`	a few seconds	rolling — see `Retry-After`
`PER_MINUTE_LIMIT_EXCEEDED`	1 minute	rolling — see `Retry-After`
`DAILY_LIMIT_EXCEEDED`	calendar day	midnight, server time (`Asia/Dhaka`, UTC+6)
`MONTHLY_LIMIT_EXCEEDED`	calendar month	1st of the month, midnight `Asia/Dhaka`

Every response (successful or not) that reaches quota checking includes, where a cap applies to your plan: `X-Quota-Daily-Limit`, `X-Quota-Daily-Remaining`, `X-Quota-Daily-Reset` (Unix timestamp), and the `X-Quota-Monthly-*` equivalents — so you can track how close you are to the daily/monthly ceiling well before hitting it, not just after. A `429` on any of the four codes above also includes a `details` object in the error body itself with the exact limit, current usage, and `reset_at` for both the daily and monthly windows, so you don't need to parse headers to understand why a request was rejected:

```
{
  "success": false,
  "error": {
    "code": "DAILY_LIMIT_EXCEEDED",
    "message": "Request limit exceeded.",
    "details": {
      "reset_at": "2026-08-14T02:00:00+00:00",
      "retry_after_seconds": 3600,
      "daily": { "limit": 5000, "used": 5000, "remaining": 0 },
      "monthly": { "limit": 25000, "used": 12000, "remaining": 13000 }
    }
  }
}
```

Test keys (sandbox) — free while you build

Every key belongs to one of two environments, and the key itself tells you which:

Prefix	Environment	Billed?
`up_test_…`	Sandbox	**No** — never counts toward your monthly allowance
`up_live_…`	Production	Yes

A test key returns the **same real catalogue data** as a live key, so you can build and debug against exactly what you will ship. Responses carry `X-Sandbox: true`.

Because it is free, a test key is capped on its own daily limit (1,000 calls/day by default) and has **no monthly allowance** — it is for development, not for running your site. Hitting the cap returns the usual `429` with `DAILY_LIMIT_EXCEEDED`, and it resets at midnight. Ask us if you need the cap raised while you build.

Switch to your `up_live_…` key when you go live; nothing else in your integration changes.

Conditional requests — calls that cost you nothing

Every successful `GET` returns an `ETag` header. Send that value back on the next request as `If-None-Match`, and if nothing has changed you get `304 Not Modified` with an empty body.

A `304` is not counted against your monthly allowance at all. It consumes no quota and is never billed as overage. It still counts toward the per-minute rate limit, so keep your normal throttling.

```
# First call — note the ETag
curl -i "https://api.uaeprojects.com/v1/projects?per_page=100"   -H "Authorization: Bearer $API_KEY"
# HTTP/1.1 200 OK
# ETag: "45ff9624d145a53ead58107ff4a292c1d3e270a6a1afb50893a5a2596bd55021"

# Later — ask only for changes
curl -i "https://api.uaeprojects.com/v1/projects?per_page=100"   -H "Authorization: Bearer $API_KEY"   -H 'If-None-Match:
"45ff9624d145a53ead58107ff4a292c1d3e270a6a1afb50893a5a2596bd55021"'
# HTTP/1.1 304 Not Modified   <- no body, no quota used
```

Store the `ETag` alongside whatever you cached from that response, keyed by the exact URL — the validator covers the full payload, so a different page, field selection or filter has its own ETag and will never produce a false `304`. Weak validators (`W/"…"`), comma-separated lists and `*` are all handled per RFC 9110.

Your client portal shows how many calls this has saved you, under **Technical usage**.

Overage billing (plan-dependent)

Some plans allow exceeding the monthly allowance instead of rejecting requests with `MONTHLY_LIMIT_EXCEEDED` — this is off by default and only applies if your contract enables it. When it's active and you're past your monthly allowance, requests keep succeeding and the response includes `X-Quota-Overage: true` plus `X-Quota-Overage-Price-Per-Request`; extra usage is billed on your next invoice, not charged automatically per request. Daily and per-minute/burst limits are unaffected — those still return `429` regardless of overage billing.

Endpoints and permissions

Endpoint	Scope	Additional entitlement
`GET /projects` and `/projects/{id-or-slug}`	`projects:read`	—
`GET /projects/changes?updated_since=...`	`projects:sync`	—
`GET /developers[/{id-or-slug}]`	`developers:read`	—
`GET /communities[/{id-or-slug}]`	`communities:read`	—
`GET /states/{id-or-slug}`	`states:read`	—
`GET /sub-communities` and `/sub-communities/{id-or-slug}`	`subcommunities:read`	—
`GET /catalog/metadata`	`catalog:read`	—
`GET /projects/{project}/floor-plans`	`floorplans:read`	`floorplans`
`GET /projects/{project}/payment-plans`	`paymentplans:read`	—
`GET /projects/{project}/media`	`media:read`	`images`
`GET /projects/{project}/video`	`media:read`	`images`
`GET /projects/{project}/amenities`	`amenities:read`	—
`GET /projects/{project}/nearby`	`nearby:read`	—
`GET /projects/{project}/payment-methods`	`paymentmethods:read`	—
`GET /projects/resources?ids=...&include=...`	Every requested resource's scope	`images` for media; `floorplans` for floor plans
`GET /projects/{project}/faqs`	`faqs:read`	—
`GET /projects/{project}/virtual-tours`	`virtualtours:read`	`virtual_tours`
`GET /projects/{project}/floor-plates`	`floorplates:read`	`floorplates`
`GET /projects/{project}/brochure`	`brochures:read`	`brochures`
`GET /projects/{project}/related`	`projects:read`	—
`POST /projects/{project}/pdf`	`project-pdf:generate`	`project_pdf`
`GET /agent-profiles`	`project-pdf:generate`	`project_pdf`
`GET /pdf-templates`	`project-pdf:generate`	`project_pdf`
`GET /webhooks`	`webhooks:manage`	`webhooks`
`POST /webhooks`	`webhooks:manage`	`webhooks`

Endpoint	Scope	Additional entitlement
`PUT /webhooks/{id}`	`webhooks:manage`	`webhooks`
`DELETE /webhooks/{id}`	`webhooks:manage`	`webhooks`
`POST /webhooks/{id}/rotate-secret`	`webhooks:manage`	`webhooks`

Project filters: `q`, `developer`, `community`, `state`, `nature`, `min_price`, `max_price`, `updated_since`, `ids`, `view`, `sort`, `page`, and `per_page` (maximum 100). Sort values are `updated_at`, `starting_from`, or `name`; prefix with `-` for descending. Catalog lists accept `q`, `updated_since`, `sort`, `page`, and `per_page`; communities and sub-communities also accept `state` / `community`.

`ids` fetches multiple known projects in a single call instead of one request per project, e.g. `GET /projects?ids=101,102,103` (comma-separated numeric IDs, up to 100 per request). The default `view=summary` remains backward compatible. Use `GET /projects?ids=101,102,103&view=detail` to receive the same extended representation as `GET /projects/{project}` — including `overview`, `description`, `master_plan`, `property_no`, `license_no`, `project_types`, `video_description`, `map_embed_html`, `down_payment_percent`, and `service_charges`. Detailed mode requires unique explicit IDs, is available only on `/projects`, and automatically sizes `per_page` to the supplied ID count when it is omitted.

Detailed batches consume one quota unit per 10 requested project IDs, rounded up (1–10 IDs = 1 unit; 100 IDs = 10 units). The response exposes the charge in `meta.quota_units` and `X-Quota-Units`. A rejected `429` and an ETag `304 Not Modified` response consume zero quota. Catalogue restrictions and plan field-access rules are applied exactly as they are on individual project detail calls; inaccessible or unpublished IDs are omitted and do not affect pagination totals.

Related-resource batches

Use `GET /projects/resources?ids=101,102,103&include=media,floor_plans,amenities,payment_plans,payment_methods` to retrieve selected related collections without making a separate call for every project and resource. A request may contain up to **25 unique project IDs** and any unique subset of these five resource types:

- `media` — requires `media:read` and the `images` entitlement
- `floor_plans` — requires `floorplans:read` and the `floorplans` entitlement
- `amenities` — requires `amenities:read`
- `payment_plans` — requires `paymentplans:read`
- `payment_methods` — requires `paymentmethods:read`

Hyphenated names such as `floor-plans` are accepted and normalized. The key must be authorized for **every** requested resource; otherwise the complete request returns `403` with `missing_scopes` or `missing_entitlements`, so no partially authorized payload is leaked. Catalogue restrictions are applied before loading related data. Inaccessible or unpublished IDs are omitted and listed in `meta.omitted_project_ids`.

The weighted charge is one quota unit per five requested project-resource pairs, rounded up. For example, 25 projects × 1 resource costs 5 units, and 25 projects × all 5 resources costs 25 units. The exact charge appears in `meta.quota_units` and `X-Quota-Units`; rejected `429` and cached `304` responses consume zero quota. Existing single-project related-resource endpoints remain supported and unchanged.


```
{
  "data": [
    {
      "project_id": 101,
      "resources": {
        "media": { "items": [], "master_plan_document": null },
        "amenities": { "items": [] }
      }
    }
  ],
  "meta": {
    "representation": "related_resources",
    "requested_project_count": 3,
    "returned_project_count": 1,
    "omitted_project_ids": [102, 103],
    "included_resources": ["media", "amenities"],
    "quota_units": 2
  }
}
```


Document endpoints (brochure, video file) return short-lived signed URLs (e.g. `download_url`, `file_url`). These are genuinely public, unauthenticated links — the `expires` / `signature` query parameters are the authorization, so no `Authorization` header, `X-Client-Domain`, or request signing is needed (or accepted) to fetch them. This makes them safe to use directly in an `<a href>`, `<video src>`, or `<iframe>` in a browser. They expire quickly and cannot be extended or reused once expired — request a fresh one from the generating endpoint if needed. Most signed links stream the file directly `200`);

a small number of legacy documents hosted on a third-party file host instead respond with a `302` redirect to that host — follow redirects when fetching programmatically. A `404` with code `DOCUMENT_UNAVAILABLE` / `VIDEO_NOT_AVAILABLE` means the underlying file could not be located (data gap on our side), not an access problem — please report the project ID if you hit this.

`GET /projects/{project}/video` returns `{ "title", "description", "external_url", "file_url", "file_expires_at", "thumbnail_url" }`. A project has either an uploaded file (`file_url` , a short-lived signed URL — treat like the PDF/brochure download URLs) or an external link `external_url` , e.g. a YouTube URL — embed it yourself, or extract the video ID and build `https://www.youtube.com/embed/{id}` for an iframe player). At most one of the two is populated. `404` with code `VIDEO_NOT_AVAILABLE` means the project has no video.

Project detail also includes `map_embed_html` , a ready-to-render map iframe/embed snippet you can show as a fallback whenever `location.latitude` / `location.longitude` are `null` (coordinates are not populated for every project yet). It is `null` when no fallback embed is configured either.

Idempotent PDF generation

`POST /projects/{project}/pdf` accepts an optional `Idempotency-Key` header (any unique string you generate per logical request, e.g. a UUID). If the same key is sent again — for example after a network timeout where you are unsure whether the first call succeeded — the API returns the same result (`"idempotent_replay": true` in the response) with a freshly signed, unexpired `download_url` , without generating a new PDF or consuming PDF-generation quota a second time. Keys are remembered per client for 24 hours.

Some plans include a final UAE Projects / Marketech branding page at the end of the generated PDF (contract-dependent, off by default for most plans). This is a whole extra page, not a footer line, so PDFs from these plans will have one more page than the source project's own content. The Classic branded back cover retains its UAE Projects copyright/contact footer; the duplicated website/email line is intentionally omitted from the middle of that page.

Attaching an agent profile

`POST /projects/{project}/pdf` accepts an optional `agent_profile_id` , which prints that agent's contact details on the brochure. Send it in the JSON body (a query-string parameter works too):

```
POST /v1/projects/1803/pdf
Authorization: Bearer up_live_xxx
Content-Type: application/json

{ "agent_profile_id": 1 }
```

Use `GET /v1/agent-profiles` to list the ids you can pass. It returns only profiles that are **approved** and belong to your account:

```
{ "data": [ { "id": 1, "agent_name": "First Agent", "company_name": "…",
              "email": "…", "phone": "…", "whatsapp": "…", "image_url": "…" } ] }
```

Both endpoints need the same `project-pdf:generate` scope and `project_pdf` entitlement as PDF generation. Passing an id that is not yours, not approved, or does not exist returns `422` . Omit the field entirely to generate a brochure with no agent page.

Choosing the design per request

Your account has a **default** design, set in the client portal under **PDF layout**. If you enable additional designs there, you can also pick one per request with `template` :

```
POST /v1/projects/1803/pdf
Content-Type: application/json

{ "template": "editorial", "agent_profile_id": 1 }
```

Omit `template` to use your account default. Passing a design you have not enabled returns `422` .

List what your account can request — ideal for populating a dropdown on your own site rather than hard-coding names:

```
GET /v1/pdf-templates
```

```
{ "data": [
  { "template": "classic",   "name": "Classic",   "description": "…", "is_default": true  },
  { "template": "editorial", "name": "Editorial", "description": "…", "is_default": false }
] }
```

Both `template` and `agent_profile_id` are independent, so a "Generate brochure" button on your site can offer a design dropdown and an agent dropdown side by side, and pass both in one call.

How PDF caching works

A generated PDF is cached against a fingerprint of: the project, the project's last-updated timestamp, the selected layout, the attribution setting, and the agent profile. Change any one of those and the next call generates a fresh PDF automatically.

That means **there is no cache-invalidation endpoint, and you never need to wait for a project to change at source** Switching your account default, requesting a different `template`, or attaching a different agent all produce new PDFs on the very next request. The previously cached copies simply stop being requested.

Each newly generated combination consumes one unit of PDF-generation quota. Repeat calls for a combination already cached returr `"cached": true` and consume none.

Enterprise request signing

When enabled for a client, send `X-API-Timestamp` (Unix seconds), a unique 16–128 character `X-API-Nonce`, and `X-API-Signature: v1=<hex>`. Build the canonical value using newline separators:

```
timestamp
nonce
UPPERCASE_METHOD
raw_path_and_query
sha256_hex_of_body
```

`raw_path_and_query` is the full request-target exactly as sent on the wire, **including the `/v1` prefix** — e.g. for `https://api.uaeprojects.com/v1/projects?per_page=25` the value is `/v1/projects?per_page=25`, not `/projects?per_page=25`. For GET/HEAD, the body is empty (its SHA-256 hex digest is the hash of an empty string). HMAC-SHA256 the canonical value with the API key. Timestamps have a five-minute default window and nonces are single-use.

Integration examples

PHP

```
$ch = curl_init('https://api.uaeprojects.com/v1/projects?per_page=25');
curl_setopt_array($ch, [CURLOPT_RETURNTRANSFER => true, CURLOPT_HTTPHEADER => [
    'Authorization: Bearer '.getenv('UAEPROJECTS_API_KEY'), 'Accept: application/json',
]]);
$payload = json_decode(curl_exec($ch), true, flags: JSON_THROW_ON_ERROR);
```

Laravel

```
$response = Http::withToken(config('services.uaeprojects.key'))->acceptJson()
    ->retry(2, 500, throw: false)->get('https://api.uaeprojects.com/v1/projects', ['updated_since' => $cursor]);
$response->throw();
$projects = $response->json('data');
```

Browser JavaScript

Only call from an approved server-side proxy; do not embed a live key in public JavaScript.

```
const response = await fetch('/your-secure-backend/uaeprojects?community=downtown-dubai');
const { data, meta } = await response.json();
```

Node.js

```
const response = await fetch('https://api.uaeprojects.com/v1/projects?per_page=25', {
  headers: { Authorization: `Bearer ${process.env.UAEPROJECTS_API_KEY}`, Accept: 'application/json' }
});
if (!response.ok) throw new Error(`${response.status}: ${await response.text()}`);
const payload = await response.json();
```

WordPress

```
$response = wp_remote_get('https://api.uaeprojects.com/v1/projects?per_page=25', [
    'headers' => ['Authorization' => 'Bearer '.get_option('uaeprojects_api_key'), 'Accept' => 'application/json'],
    'timeout' => 15,
]);
if (is_wp_error($response)) { throw new RuntimeException($response->get_error_message()); }
$payload = json_decode(wp_remote_retrieve_body($response), true, flags: JSON_THROW_ON_ERROR);
```

Webhooks

Supported events are `project.created`, `project.updated`, `project.status_changed`, `project.deleted`, `developer.updated`, and `community.updated`.

Every delivery is a `POST` with a JSON body shaped `{ "id": "<event id>", "type": "<event type>", "created_at": "...", "data": { ... } }` and these exact headers:

Header	Contents
`X-UAEProjects-Event`	The event type, e.g. `project.updated`
`X-UAEProjects-Event-Id`	The same value as the body's `id` field
`X-UAEProjects-Timestamp`	Unix seconds when the request was sent
`X-UAEProjects-Signature`	`v1=<hex>` — HMAC-SHA256 signature

To verify a delivery: compute `HMAC-SHA256(timestamp + "." + event_id + "." + raw_request_body, your_webhook_secret)` using the values from `X-UAEProjects-Timestamp` and `X-UAEProjects-Event-Id` (not values parsed from the body), hex-encode it, and compare against the value after `v1=` in `X-UAEProjects-Signature` using a constant-time comparison. Reject requests with an old `X-UAEProjects-Timestamp` (outside a few minutes) and store processed `X-UAEProjects-Event-Id` values idempotently, since retries reuse the same event ID. Return any 2xx response promptly; failed deliveries retry with exponential intervals and can be manually retried by an administrator.

Managing webhooks via the API

Clients with the `webhooks:manage` scope and `webhooks` entitlement can fully self-manage their webhook endpoints without contacting support:

```
# Create
curl -X POST https://api.uaeprojects.com/v1/webhooks \
  -H 'Authorization: Bearer up_live_...' -H 'Content-Type: application/json' \
  -d '{"name":"Production","url":"https://example.com/webhooks/uaeprojects","events":["project.created","project.updated"]}'

# List
curl https://api.uaeprojects.com/v1/webhooks -H 'Authorization: Bearer up_live_...'

# Update (partial)
curl -X PUT https://api.uaeprojects.com/v1/webhooks/{id} \
  -H 'Authorization: Bearer up_live_...' -H 'Content-Type: application/json' -d '{"status":"disabled"}'

# Rotate the signing secret
curl -X POST https://api.uaeprojects.com/v1/webhooks/{id}/rotate-secret -H 'Authorization: Bearer up_live_...'

# Delete
curl -X DELETE https://api.uaeprojects.com/v1/webhooks/{id} -H 'Authorization: Bearer up_live_...'
```

The signing secret (`secret`) is returned only once, on creation and on rotation — store it immediately. `{id}` is the webhook's `id` field from the create/list response, not an internal database ID.

Versioning and changelog

Breaking changes receive a new URL version. Additive fields and endpoints may be introduced within v1; integrations must ignore unknown JSON fields. Deprecations will be announced before removal. The machine-readable contract is `/openapi.yaml`.

- 2026-08-30: added opt-in detailed core batches for up to 100 IDs and a separately authorized related-resource batch for media, floor plans, amenities, payment plans and payment methods across up to 25 IDs. Both use weighted quota reporting; all existing summary and single-project responses remain unchanged.
- 2026-08-24: added account-specific project PDF design discovery (`GET /pdf-templates`), optional per-request `template`, approved Agent Profile discovery/attachment, stronger PDF cache separation by layout/agent/branding, first-party project image/icon resolution, and corrected Classic/Editorial branded back covers. The portal now also provides dependent inventory filters, itemized plan-change billing, tracked company/agent brochure generation and the Agent Card branding permission guard.
- 2026-08-15: fixed `/developers/{id}` `logo_url` returning stale local-development URLs for 34 developer records; fixed several project PDF issues — missing embedded images, a missing final page (location + contact section), and added an optional UAE Projects / Marketech branding page (plan-dependent, off by default).
- 2026-08-15: added optional overage billing — plans can now allow exceeding the monthly allowance for a per-request charge instead of rejecting with `429`, signalled via `X-Quota-Overage` / `X-Quota-Overage-Price-Per-Request`. Off by default per plan.
- 2026-08-13: fixed `X-Quota-Daily-*` headers not being sent (previously documented but missing from actual responses) and added a `details` object to `429` quota errors with the exact limit, usage, and reset time for the daily/monthly window that was hit. Documented `GET /v1/health` and the four distinct rate/quota limit codes with their reset schedule.
- 2026-08-12: fixed signed document/PDF/video download links so they no longer require an API key or request signature — they are now truly public, unauthenticated URLs as intended, usable directly in `<a href>` / `<video src>`. Also added `states`, `sub-communities`, project `amenities` / `nearby` / `payment-methods` / `faqs` / `virtual-tours` / `floor-plates` / `video`, `map_embed_html` on project detail, bulk fetch via `?ids=`, `Idempotency-Key` support on PDF generation, and self-service webhook management (`/webhooks`).
- 2026-07-20: v1 foundation, commercial authentication, quotas, published catalogs, premium resources, signed requests, usage reporting, and signed webhooks.

Enterprise catalogue controls and project PDFs

Each client environment has one authorized normalized domain. Configure the exact host (not a URL path or wildcard) and, where required, an IP allow/deny policy. Server-to-server integrations must send `X-Client-Domain`; this header supplements API-key, IP, and optional request-signature controls and is not a replacement for them.

Restricted catalogue clients receive only their assigned states. A state with no selected community grants that client all published communities in the state; selecting communities narrows that state to those communities. Inaccessible projects, communities, premium project resources, related projects, delta records, and PDF generation return `404` and do not affect pagination totals.

`POST /projects/{project}/pdf` requires `project-pdf:generate` and the `project_pdf` entitlement. It returns a short-lived signed `download_url`; callers must retain their bearer key for the download. New PDF generation is governed by editable PDF minute/day/month limits and returns `429` with `Retry-After` when denied. Reusing an identical cached approved PDF does not consume PDF-generation quota, although it remains a normal authenticated API request. Keys belong only in private server-side configuration; use staging `up_test_` keys outside production and never expose `up_live_` keys in browser code.

Commercial portal lifecycle

The Client Portal is the self-service control plane for the same contract: inventory coverage is selected by state and optional community, communities are limited to the chosen states, and individually priced access points are itemized in the live plan calculator. A submitted plan change does not alter API access immediately. Staff reviews its current-versus-requested difference, approval issues an itemized invoice, and the approved policy becomes effective after payment confirmation (unless staff deliberately authorizes an active-but-unpaid subscription).

Approved Agent Profiles can sign in to the Agent Portal and generate tracked brochures only from the company's effective catalogue. Company users can also generate tracked project brochures from PDF Layout. The portal permits an Agent Card only when the account has **PDF exports without branding**; otherwise portal downloads retain Marketech branding. These portal rules do not change the API scopes in the endpoint table above—API callers must still use the entitlements and approved IDs returned for their own account.

> Property data source: uaeprojects.com. The API platform, client access, quotas and support console are operated under the Marketech API brand.