Generic API Guide (Frontend) V1.01

Base URL: https://uiapi.pgo.mv/api

This API provides generic CRUD endpoints for any Eloquent model by using the model name in the path. It also supports powerful query parameters for filtering, searching, sorting, selecting columns, and eager-loading relations.

Endpoints

- List: `GET /api/{model}`
- Show: `GET /api/{model}/{id}`
- Create: `POST /api/{model}`
- Delete: `DELETE /api/{model}/{id}`

Model Names

Use the model class basename in lowercase. Examples from this app:

- `person` → `App\Models\Person`
- `country` → `App\Models\Country`
- `entrytype` → `App\Models\EntryType`
- `user` → `App\Models\User`

Note: Use letters only (no dashes/underscores). The API maps `/{model}` by lowercasing and then capitalizing.

Listing: Query Parameters

Attach these to `GET /api/{model}`:

- `columns`: comma list of columns to return.
  - Example: `columns=id,name,email`
- `search`: advanced AND/OR search with `column:value` pairs.
  - AND within a group: comma-separated
  - OR across groups: pipe (`|`) separated
- `filter`: exact-match filters with AND across pairs and OR within a field.
  - OR within a field: use `|` (pipe), e.g., `gender:M|F` → gender IN [M,F].
  - Special values:
    - `!null`: matches NOT NULL, e.g., `closed_at:!null`.
    - empty string: use an empty value to match empty string OR NULL, e.g., `status:`.
  - No wildcards: all comparisons are exact (no LIKE).
    - `filter=gender:M|F,country.code:MV`
- `wrap`: when set to `data`, and `pagination=off`, wraps the array response in a top-level `data` key.
  - Useful for option lists and components expecting `{ data: [...] }`.
  - Example: `wrap=data`
  - `filter=closed_at:null`

    - `filter=status:` (matches empty string or NULL)
- `searchAll`: single term searched in model-defined `searchable` columns (if the model defines them).
    - Example: `searchAll=john`
- `withoutRow`: exclude rows matching exact values using the same AND/OR grammar as `search`.
    - Example: `withoutRow=status:archived|role:guest`
- `sort`: comma list; prefix `-` for descending.
    - Example: `sort=-created_at,name`
- `with`: eager load relations; optionally restrict related columns.
    - Example: `with=entries:id,title` (primary key is auto-included for related models)
- `pivot`: eager load belongsToMany relations including pivot data (if the relation exists and is many-to-many).
    - Example: `pivot=tags,categories`
- Without pagination + wrapped (`pagination=off&wrap=data`):

```
{ "data": [ { /* record */ }, ... ] }
```

- `pagination`: set to `off` to return all rows; otherwise results are paginated (default per-page).
    - You can pass standard `page=2` for subsequent pages.

Listing Examples

- `GET /api/person?pagination=off`
- `GET /api/person?columns=id,name,email&search=name:Ali|email:gmail&sort=-created_at&with=entries:id,title`
- People using the new filters (AND pairs, OR within a field):
    - `GET /api/person?filter=gender:M|F,country.code:MV&sort=name`
- Option list style (wrapped, no pagination):
    - `GET /api/country?columns=id,name_div&sort=name_div&pagination=off&wrap=data`
    - `GET /api/entry?filter=closed_at:!null`
    - `GET /api/user?filter=active:true,status:`

Response Shapes

- Paginated: Laravel paginator JSON

```
{
  "data": [ { /* record */ }, ... ],
  "links": { /* first, last, next, prev */ },
  "meta": { /* current_page, from, to, per_page, total */ }
}
```

- Without pagination (`pagination=off`): plain array of records

```
[ { /* record */ }, ... ]
```

Show

- `GET /api/{model}/{id}`
- Supports `columns`, `with`, and `pivot` like listing.
- Example: `GET /api/person/123?with=entries:id,title`

Create

- `POST /api/{model}` with a JSON body of the model fields.
- Example:

```
POST /api/person
Content-Type: application/json

{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "secret"
}
```

- Notes:
  - `password` is hashed server-side if provided.
  - Validation rules depend on the model; if validation fails, you'll get `422` with details.
  - Some models may define file fields; those are handled by the backend's file upload service during create (send as multipart/form-data when applicable).

Update

- `PUT /api/{model}/{id}` with JSON body of fields to update.
- Example:

```
PUT /api/person/123
Content-Type: application/json

{
  "email": "johnny@example.com",
  "password": "newsecret"
}
```

- Notes:
  - `password` is hashed if provided.
  - Returns the updated record.

Delete

- `DELETE /api/{model}/{id}`
- Returns `204 No Content` on success.

Errors & Status Codes

- `400`: model not found (invalid `{model}` path)
- `404`: record not found (invalid `{id}`)
- `422`: validation failed (on create/update)
- `500`: server error

Authentication

If your environment requires authentication, include a bearer token:

```
Authorization: Bearer <token>
```

Quick cURL Examples

```
# List people, with filters and relations
curl -s "https://uiapi.pgo.mv/api/person?
columns=id,name,email&search=name:Ali|email:gmail&sort=-
created_at&with=entries:id,title"

# Show one person
curl -s "https://uiapi.pgo.mv/api/person/123?with=entries:id,title"

# Create a person
curl -s -X POST "https://uiapi.pgo.mv/api/person" \
  -H "Content-Type: application/json" \
  -d '{"name":"John Doe","email":"john@example.com","password":"secret"}'

# Update a person
curl -s -X PUT "https://uiapi.pgo.mv/api/person/123" \
  -H "Content-Type: application/json" \
  -d '{"email":"johnny@example.com"}'

# Delete a person
curl -s -X DELETE "https://uiapi.pgo.mv/api/person/123"
```