Model Generator Service (MGS)

Generate a fully scaffolded **Model** (with `apiSchema`, validation rules, relationships, casts) and a **view config JSON** from either a Laravel migration file or a raw SQL dump.

Quick Start

```
# From a migration
php artisan uiapi:generate Person --
migration=database/migrations/2024_01_01_create_people_table.php

# From a SQL dump
php artisan uiapi:generate Person --sql=storage/dumps/people.sql

# Interactive (prompts for source and path)
php artisan uiapi:generate
```

Command Signature

```
uiapi:generate {name?} {--migration=} {--sql=}
```

Argument / Option	Description
`name`	Model name in StudlyCase (e.g. `Person`, `CourtCase`). Prompted if omitted.
`--migration`	Path to a Laravel migration file (absolute or relative to project root).
`--sql`	Path to a raw SQL dump file containing a `CREATE TABLE` statement.

If neither `--migration` nor `--sql` is provided, the command presents an interactive choice. If both are provided, `--sql` takes precedence.

Generated Output

Two files are created (with overwrite confirmation if they already exist):

File	Path	Contents
Model	`app/Models/{Name}.php`	Extends `BaseModel`, includes `$table`, `$searchable`, `casts()`, `apiSchema()`, `baseRules()`, `rulesForCreate()`, `rulesForUpdate()`, `validationMessages()`, and relationship methods.

File	Path	Contents
View Config	`app/Services/viewConfigs/{name}.json`	`listView` block with `table`, `form`, `toolbar`, `filterSection`, and `meta` components pre-configured from the parsed schema.

Source: Migration File

Standard Laravel migrations are supported. The parser extracts:

- **Table name** from `Schema::create('table_name', ...)`
- **Columns** from `$table->type('name')` chains — recognizes `id`, `uuid`, `string`, `text`, `longText`, `integer`, `bigInteger`, `unsignedBigInteger`, `unsignedInteger`, `tinyInteger`, `smallInteger`, `boolean`, `date`, `dateTime`, `timestamp`, `decimal`, `float`, `double`, `json`, `foreignId`
- **Modifiers** — `->nullable()`, `->unique()`, `->default(...)`, `->constrained('table')`
- **Foreign keys** — explicit `->foreign(...)->references(...)->on(...)` and `->constrained()`
- **Timestamps** — detects `$table->timestamps()`

Example

```
php artisan uiapi:generate Invoice --
migration=database/migrations/2025_03_10_create_invoices_table.php
```

Source: SQL Dump

For legacy tables without migrations, export the schema and point the generator at it.

Obtaining a dump

```
# MySQL — structure only, single table
mysqldump -u root -p --no-data your_database people > people.sql

# Or export via phpMyAdmin → Export → Structure only
```

The parser handles:

- `CREATE TABLE` statements (with or without backticks, `IF NOT EXISTS`)
- Column types: `int`, `bigint`, `varchar`, `char`, `text`, `longtext`, `enum`, `set`, `decimal`, `float`, `double`, `date`, `datetime`, `timestamp`, `json`, `tinyint`, `boolean`, `blob`, `binary`, `uuid`, etc.
- `tinyint(1)` is treated as **boolean**
- `NOT NULL` / `NULL`, `DEFAULT` values, `UNIQUE` constraints
- `FOREIGN KEY ... REFERENCES` constraints (inline or via `ALTER TABLE`)

- Columns ending in `_id` are auto-inferred as foreign keys (e.g. `country_id` → `countries.id`) when no explicit constraint exists
- `created_at` + `updated_at` presence enables `$timestamps = true`

Example

```
php artisan uiapi:generate Person --sql=storage/dumps/people.sql
```

What to Review After Generation

The generator produces a solid starting point. Review and adjust:

1. **Dhivehi labels** — marked as `"TODO"` in both `apiSchema()` and the view config JSON.
2. **Relationship model references** — the generator infers `belongsTo` from foreign keys but uses a guessed class name. Verify imports and class names.
3. **Validation rules** — business-specific constraints (conditional rules, custom rules) need manual additions.
4. **View config components** — tune `columnCustomizations`, `filters`, form `groups` and `fields`, `per_page`, and component-level overrides as needed.
5. **Casts** — `json` columns cast to `array`, `date`/`datetime` to `datetime`, `boolean` to `boolean`. Add custom casts if needed.

Type Mapping Reference

Migration Types → Normalized

Migration Type	Normalized
`id`, `integer`, `bigInteger`, `unsignedBigInteger`, `unsignedInteger`, `tinyInteger`, `smallInteger`, `foreignId`	`integer`
`boolean`	`boolean`
`decimal`, `float`, `double`	`number`
`date`, `dateTime`, `timestamp`	`date`
`json`	`json`
`string`, `text`, `longText`, `uuid`	`string`

SQL Types → Normalized

SQL Type	Normalized
`int`, `bigint`, `smallint`, `mediumint`, `tinyint` (except `tinyint(1)`)	`integer`
`tinyint(1)`, `bool`, `boolean`	`boolean`
`decimal`, `float`, `double`, `numeric`, `real`	`number`

SQL Type	Normalized
`date`, `datetime`, `timestamp`	`date`
`json`, `jsonb`	`json`
`varchar`, `char`, `text`, `longtext`, `mediumtext`, `enum`, `set`, `blob`, `binary`, `uuid`	`string`