Templating engine for Python and Angular CRUD interfaces.

This tool generates the Python backend and Angular with Angular Material frontend code for CRUD interfaces.

Table of Contents

1 General

1.1 Licencing

2. Installation

Download the zip file from the version control or use git to obtain the repository.

2.1 Install via .zip

```
pip3 install pytemplate-master.zip
```

```
git clone https://gitlab.pe2mbs.nl/angular/pytemplate.git
cd pytemplate
pip3 install .
```

Note: if you're using Python 2.7.x use pip2

3. Usage

Standard usage:

```
gencrud [ options ] { <template-file> [ <template-file-n> ] }
                    { [ <template-folder> ] * }
```

`template-file` is the template filename that you want to be processed.

`template-folder` is the template folder that contains the template .yaml files, the

3.1. Actual usage

```
gencrud examples\role-table.yaml
```

This generates for the role table the frontend and backend code.

```
gencrud examples\role-table.yaml examples\user-table.yaml
```

This generates for the role and user tables the frontend and backend code.

```
gencrud examples/*
```

This generates for all the template files in the folder examples the frontend and backend code.

3.2. Options

The following options

> -h / --help This help information.
>
> -v Verbose option, prints what the tool is doing.
>
> -V / --version Print the version of the tool.

> -b / --backup Make backup of the orginal project files files.

When used every file that belongs to the orignal project will be backuped every time its altered. This is override by the `options.backup` in the template file.

> -o / --overwrite Force overwriting the files.

If this option is omitted the program will exit on encountering a module name that already exists. This is override by the `options.overwite` in the template file.

> -s / --sslverify Disable the verification of ssl certificate when retrieving some external profile data.

When you are behind a proxy that uses it own certificates, you may need to enable this option ones, to retrieve to extra data files from the nltk package.

> -c / --ignore-case-db-ids Set the database ids in lower case.

This option is for databases that are installed on a Windows server system. As sqlalchemy detectes the changes in the database model, when using full uppercase columns names, the detection fails.
This is override by the `options.ignore-case-db-ids` in the template file.

> -M / --use-module Creates a module per object that in included into app.module.ts and injects a route into app-routing.module.ts when it exists.

Enables the creation of an Angular module per template. Therefore only the one module is imported in app.module.ts, instead of at least 4 components. And in app-routing.module.ts the Route is injected as a single variable instead of all the routes needed to handle the template components.
This is override by the `options.use-module` in the template file.

4. Requirements

For the default templates there are requirements to the Python and Angular project setup.

4.1. Python

4.1.1. Packages

The following packages are a minimal requirement;

- Flask, version 1.0.2 or higher
- SQLAlchemy, version 1.2.12 or higher
- marshmallow, version 2.15.6 or higher
- flask-marshmallow, version 0.10.1 or higher
- Flask-SQLAlchemy, version 2.3.2, or higher
- marshmallow-sqlalchemy, version 0.19.0 or higher

4.1.2. modules

In the root of the project the following modules and variables must be present;

- from applic.database import db
- from applic.extensions import mm
- import common

4.2. Angular

4.2.1. Packages

The following packages are a minimal requirement;

- @angular, version 6 or higher (tested with 6 and 7).

- @angular/material, version 6 or higher (tested with 6 and 7).
- @angular/flex-layout, version 7.0.0-beta.24
- ngx-material-timepicker, version 2.13.0
- html2canvas, version 1.0.0-rc.1
- core-js, version 2.4.1
- zone.js, version 0.8.26

4.2.2. modules

4.3. Example files

The folowing example files use the templates that where installed with the package.

- examples\role-table.yaml
- examples\user-table.yaml
- examples\screens-base.yaml

The folowing example file use private templates that where created by you. examples\screens.yaml

5. YAML Template

YAML (YAML Ain't Markup Language) is a human-readable data serialization language. It is commonly used for configuration files. See for more information https://yaml.org/spec/1.2/spec.html.

Below an example of a simple template for demonstration purposes.

```
templates:
  python: ./templates/python
  angular: ./templates/angular
  common:
    python: ./templates/common/python
    angular: ./templates/common/angular
source:
  base: ./output
  python: backend
  angular: src/app
objects:
  - name: role
    class: Role
    application: testapp
    uri: /api/role
    menu:
      displayName: Database
      iconName: database
      index: 0
      menu:
        displayName: Roles
        iconName: roles
        index: 0
        route: /roles
    table:
      name: WA_ROLES
      columns:
        - field: D_ROLE_ID       INT         AUTO NUMBER  PRIMARY KEY
          label: Identification
          index: 0
          ui:
            type: label
        - field: D_ROLE          CHAR(20)    NOT NULL
          label: Role
          index: 1
          ui:
            type: textbox
```

In the descriptions of the attributes all atttributes are mandatory, unless otherwise stated.

5.1. source and templates section

At root level in the file.

```
source:
  python: ./output/backend
  angular: ./output/src/app
templates:
  python: ./gencrud/python
  angular: ./gencrud/angular
  common:
    python: ./gencrud/common/python
    angular: ./gencrud/common/angular
```

When using a template on more than one machine type, ie. Linux and Windows. The `source` tag needs to be placed in a group of the machine name (all lowercase). Currently the following system types are supported: `linux`, `windows` and `osx` (as `osx` was nor tested, but assumed that its unix-like machine, it should work without any problems)

```
windows:
  source:
    base: .\output
    python: backend
    angular: frontend/src/app
  templates:
    python: .\gencrud\python
    angular: .\gencrud\angular
    common:
      python: .\gencrud\common\python
      angular: .\gencrud\common\angular
linux:
  source:
    base: ./output
    python: backend
    angular: frontend/src/app
  templates:
    python: ./gencrud/python
    angular: ./gencrud/angular
    common:
      python: ./gencrud/common/python
      angular: ./gencrud/common/angular
osx:
  source:
    base: ./output
    python: backend
    angular: frontend/src/app
  templates:
    python: ./gencrud/python
    angular: ./gencrud/angular
    common:
      python: ./gencrud/common/python
      angular: ./gencrud/common/angular
```

In the groups `source`, `templates` and `common` the `base` is optional but then the `python` and `angular` both need to contain the complete absolute path. Both `python` and `angular` are mandatory. For the `source` group the `python` points to the root

folder of the Python backend project. And the `angular` points to the root folder of the Angular project.

The `templates` group is optional, when omitted the templates that come with the package shall be used. `python` and `angular` as well as `common` are mandatory. Whenever you need to customize the templates, take a look at the templates folder in the `gencrud` package (site-packages\gencrud\templates on your platform). The templates are generated using the Mako template engine, for the syntax of this see https://www.makotemplates.org/

For the folders starting with `~` the user home folder is resolved. Whenever a folder needs to point at the current folder just use a single dot `.`

In the `source.python` folder there must be the **config.yaml** or **config.json** with the Flask configuration. The Flask configuration must contain the following keys `COMMON.API_MODULE` with the module name where the code will be generated.

In the `source.angular` folder there must be **angular.json** with the Angular configuration of the project.

5.2. Application name (module name)

At root level in the file.

```
application: testapp
```

The `application` contains the same name as in the Flask configuration key `COMMON.API_MODULE` configured.

5.2. Version

At root level in the file.

```
version: 1
```

This is an optional element and only version `1` is supported at this time.

5.3. Objects

At root level in the file. `objects` describes the components it shall generate. It contains information aboutsts out how it is presented, the table defintion, the menu entry, and its actions.

```
objects:
  - name: role
    title: User role
    class: Role
    uri: /api/role
    actionWidth: 10%
    actions: ...
    menu: ...
    extra: ...
    table: ...
```

The `name` contains the name of the module that is generated within the namespace of the `application`.

The `class` contains the name of the class name prefix/suffix used within the Python and Angular modules.

The `uri` contains REST API of the backend and frontend used for this module.

The `actionWidth` is the css defintion of the column where the action buttons are placed. The default is `10%`, for most cases this ok, but when using more than 3 buttons, or on a small screen this may not be large enough. This is an optional element.

The `actions` defines the buttons in the list view module, there are a number of default actions defined; add, edit and delete. This is an optional element. For more details see section **5.4 Actions**.

The `menu` describes the menu entry in the frontend of the module. Multiple menu levels are allowed. See for more details section **5.6 Menu**.

The `table` describes the data table with fields and their presentation properties. See for more details section **5.7 Table**.

5.4 Actions

`actions` defines the actions that should be executed when the user presses an button, icon, row, or cell.

```
actions:
  - name: add
    type: screen
    position: header
    on: click
    color: primary
    label: Add
    icon: add
    route: ...
  - name: edit
    type: screen
    position: row
    on: dblclick
    label: Edit
    route: ...
  - name: delete
    type: dialog
    on: click
    position: cell
    color: primary
    label: Delete
    icon: delete
    function: deleteRecord( i, row, 'D_ROLE' )
  - name: assign
    type: api
    position: cell
    label: Assign
    color: alert
    css: alternate-theme
    on: click
    icon: connect
    uri: /api/role/assign_something
    param:
      id: D_ID
      name: D_ROLE
```

name

The `name` is the name of the button, this is internally used and must be unique for the view. There are three predefined buttons, which may be overriden: add, edit and delete

type

The `type` is the result type where the action points to, there are the following options;

- dialog creates a dialog for the action
- screen creates a screen for the action
- list (inoperative at this time)
- api issues an rest API call to the backend
- none disabled the action, this is used to disable the default actions.

position

The `position` is the position where a button or action link is placed in the list view. The position may be one of he following;

- cell Places the button in the last cell of the table row.
- header Places the button in the header of the table view (for add).
- footer Places the button in the footer of the table view (for add).
- row Triggers on the double-click of the row.
- none Disables the action, this is used to disable the default actions.

label

The `label` is the caption of the button or an icon-button. It is used as the tooltip of the icon-button.

on

The `on` can be used to alter the behaviour of a action, normally it selects the default action based on the position parameter; for `row` it is `dblclick` and for the others it is `click`.

color

The `color` can be used to alter color scheme of the button. The color scheme that are available are the following;

- primary
- accent
- warn

When another color scheme is needed a `mat-palette` and an `angular-material-theme` must be defined in *styles.scss*. And define the alternate theme there and set theme in the css attribute of the module.

This is an optional element. When omitted the default is set to *primary*

css

The `css` can be used to set a extra css class to the action. This is an optional element.

function

The `function` defines the function that shall be executed in the Angular class. Current supported function are:

- addRecord()
- editRecord(i, row)
- deleteRecord(i, row [, ''])

Note that for `editRecord` and `deleteRecord` the parameters `i` and `row` are always as defined above.

For `deleteRecord` the column_name parameter is optional, when omitted the first column after the primary key will be used.

To use custom functions a mixin class must be be defined. And with the current version the mixin class must be added manually to the component. This shall be changed in future version.

`function` is mutually exclusive with `route` or `uri`, therefore optional.

route

The `route` describes to navigate to another component, see **5.5 Route** for more details.

`route` is mutually exclusive with `function` or `uri`, therefore optional.

uri

The `uri` is used to call the REST API of the backend.

`uri` is mutually exclusive with `route` or `function`, therefore optional.

param

The `param` is used to define parameters that need to be passed through the REST API call.

It is a dictionary with a key defining the name of the parameter, and as a value a variable or a function on the Angular class as used in the .html part of the Angular component.

This is an optional element. And only effective when `uri` is defined.

5.5 Route

name

`name` is used to construct the route, it consists out of the object's name and this `name`, therefore you should make sure that the combination is unique. Otherwise the user land on a different component as you intended.

label

This is an optional element.

class

`class` is the class name of the Angular component that needs to be activated.

module

`module` is that name of the module where the Angular component resides in.

This is an optional element. When omitted it assumes the same module as defined by `object.name`, it automaticlly selects the source file for screen or table.

params

`params` is used to define parameters that need to be passed with the Angular component.

It is a dictionary with a key defining the name of the parameter, and as a value a variable or function on the Angular class as used in the .html part of the Angular component.

This is an optional element.

5.6 Menu

```
menu:
  caption: Administration
  icon: cogs
  index: 0
  menu:
    caption: User roles
    icon: roles
    index: 0
    route: /roles
```

The `caption` or `displayName` is the caption being used in the frontend presentation.

The `icon` or `iconName` is the icon shown on the left side of the caption. This is an optional element.

The `index` is a number where in the list, where the menu entry initial should be placed. This is an optional element. When ommitted the menu entry will be added to the end of the menu.

The `menu` is a child of the menu item its placed in. The number of levels is unlimited, but its recommended not to exceed 3 levels. This is an optional element, but when ommitted the `route` element should be present.

The `route` defines the route for the frontend, each route must be unique. It always starts with a forward slash `/`. This is an optional element, but when ommitted the `menu` element should be present.

5.7 Table

```
table:
  name: WA_ROLES
  dialogtabs: ...
  screentabs: ...
  order-by:
    - D_ROLE
  viewSort:
    field: D_ROLE
    direction: desc
  columns: ...
```

name

`name` defines the name of the table in the database. This should be an unique name for the database.

dialogtabs & screentabs

`dialogtabs` and `screentabb` define the tabs for each component (dialog or screen). For more information see section **5.9 dialogtabs & screentabs**. This is are optional elements.

order-by

`order-by` defines one or more indexes, by the field name. The index name is automatically generated. This is superseeded by `column.uniqueKey`. This is an optional element.

viewSort

'viewSort' defines the default sorting of the table view, it needs two attributes;

- field
- direction 'viewSort' is an optional element.

`field` defines the column name (by the database column name) where the default sorting will be based on.

`direction` defines the sorting order, i.e. asc (ascending) or desc (decending).

columns

`columns` defines all the columns in the table with thier attributes for the handling the screen or dialog. For more inforation see section **5.8 columns**.

5.8 columns

```
 - field: D_ROLE_ID       INT         AUTO NUMBER  PRIMARY KEY
   readonly: true
   label: Identification
   ui: ...
   listview:
     index: 0
     width: 10%
 - field: D_ROLE          CHAR(20)    NOT NULL
   uniqueKey: D_ROLE_IDX
   initValue: New user role
   label: Role
   ui: ...
   listview:
     index: 1
     width: 80%
 - field: D_ROLE_COMMENT   CLOB       NULL
   label: Comment
   ui: ...
   tab:
     label: Remark
     index: 0
```

field

`field` defines in pseudo SQL the column. See **6.1 Pseudo SQL** for more information

readonly

`readonly` defines wheather a field shall be readonly in the user interface. This is an optional element.

label

`label` defines the field caption in the user interface. This is an optional element. When omitted the field shall not be shown in the user interface.

uniqueKey

`uniqueKey` defines an unique index for the column in the database. This is an optional element.

initValue

`initValue` defines the default value on the action **new**. This is an optional element.

ui

`ui` defines the properties of the field in the user interface. For more information see section **5.9 ui**. This is an optional element, but when `label` is defined its mandatory.

listview

`lastview` defines the position of the field in the table view and the width of the field. It has two attributes.

`index` defined the positon in the table view. This is an optional element. When omitted the position on the table view shall follow the order they are defined in the table.

`width` defines the width of the field in the table view. It uses CSS width defintions. Practical advise is to use percentage, so that the scaling works correcly.

`lastview` is an optional element. When omitted the field shall not be present in the table view.

tab

`tab` defines the field on a tab therefore the `screentabs` and/or `dialogtabs` must be defined earlier. `tab` has two attributes;

- `label` is the tab page label name.
- `index` is this index at what position the field should apprear on the tab page.

`tab` is an optional element. When omitted the field should be present in the edit view when `label` is defined.

5.9 ui

```
ui:
  type: choice
  debug: true
  pipe: datetime
  format: nl-NL;YYYY-MM-DD HH:mm:ss
  service: ...
  resolve-list: ...
  constant-format: "'C_{0:48} = {1}'.format( '_'.join( [ table, field, label.upper() ] ), value )"
  rows: 20
  cols: 80
```

type

`type` defines the control type in the dialog or screen. There are the following types;

- label: just the text which is readonly.
- textbox or text: text control with required and max-length validators.
- checkbox: checkbox control.
- password: password control with a unmask button.
- textarea: memo text area control.
- number: number control with up and down buttons. (not tested).
- email: an email control with email address validation.
- choice: selection list that pulls down. (not fully tested).
- combobox or combo: text control with a selection list that pulls down.
- slider: slider from minimal value to maximal value.
- slidertoggle: an on/off slider control, it has same functionality as checkbox.

- date: date control with validation. (not tested).
- time: time control with validation. (not tested).
- datetime: datetime control with validation. (not tested).
- datepicker: date picker control with validation (not fully tested).
- timepicker: time picker control with validation (not fully tested).
- datetimepicker: datetime picker control with validation (not fully tested).

Depending on the `type` other attributes may be required.

debug

`debug` is just a flag to set the online debug on, in the developer view of the browser the logging wheather the ui controls can be seen or not. This is an optional element. When omitted the default value is false.

prefix

`prefix` defines an icon or text part used as prefix for the field. The prefix is put in front of the actual field. This is an optional element.

prefix-type

`prefix-type` defines the prefix type, it can contain only two values:

- text
- icon

When omitted the default value is text. This is an optional element.

suffix

`suffix` defines an icon or text part used as suffix for the field. The suffix is put at end of the actual field. This is an optional element.

suffix-type

`suffix-type` defines the suffix type, it can contain only two values:

- text
- icon When omitted the default value is text. This is an optional element.

min

`min` is used for the **slider** control. Defines the minimal value for the slider field, when omitted the value `0` is used as default. This is an optional element.

max

`max` is used for the **slider** control. Defines defines the maximal value for the slider field, when omitted the value `100` is used as default. This is an optional element.

interval

`interval` is used for the **slider** control. It defines the interval value for the slider field, when omitted the value `1` is used as default. This is an optional element.

displayWidth

`displayWidth` is used for the **slider** control. It defines the displayWidth value for the slider field, when omitted the value `100%` is used as default. This is an optional element.

pipe

`pipe` is only effective for the **label** control. And defines the function used to format the data, it is used in conjuntion with `format`. The following values are supported at this time;

- date
- time
- datetime

This is an optional element.

format

`format` is used to format the `pipe`, `date`, `time` or `datetime` controls. `format` may consist out of two element seperated by a semicolon `;` first part (optional) the locale for the field. The locale is used for setting the correct timezone of the field. The second part is the formatting of the repersentation of the field.

	Token	Output
Month	M	1 2 ... 11 12
	Mo	1st 2nd ... 11th 12th
	MM	01 02 ... 11 12
	MMM	Jan Feb ... Nov Dec
	MMMM	January February ... November December
Quarter	Q	1 2 3 4
	Qo	1st 2nd 3rd 4th
Day of Month	D	1 2 ... 30 31
	Do	1st 2nd ... 30th 31st
	DD	01 02 ... 30 31
Day of Year	DDD	1 2 ... 364 365
	DDDo	1st 2nd ... 364th 365th
	DDDD	001 002 ... 364 365
Day of Week	d	0 1 ... 5 6
	do	0th 1st ... 5th 6th
	dd	Su Mo ... Fr Sa
	ddd	Sun Mon ... Fri Sat
	dddd	Sunday Monday ... Friday Saturday

Day of Week (Locale)	e	0 1 ... 5 6
Day of Week (ISO)	E	1 2 ... 6 7
Week of Year	w	1 2 ... 52 53
	wo	1st 2nd ... 52nd 53rd
	ww	01 02 ... 52 53
Week of Year (ISO)	W	1 2 ... 52 53
	Wo	1st 2nd ... 52nd 53rd
	WW	01 02 ... 52 53
Year	YY	70 71 ... 29 30
	YYYY	1970 1971 ... 2029 2030
	Y	1970 1971 ... 9999 +10000 +10001 **Note:** This complies with the ISO 8601 standard for dates past the year 9999
Week Year	gg	70 71 ... 29 30
	gggg	1970 1971 ... 2029 2030
Week Year (ISO)	GG	70 71 ... 29 30
	GGGG	1970 1971 ... 2029 2030
AM/PM	A	AM PM
	a	am pm
Hour	H	0 1 ... 22 23
	HH	00 01 ... 22 23
	h	1 2 ... 11 12
	hh	01 02 ... 11 12
	k	1 2 ... 23 24
	kk	01 02 ... 23 24
Minute	m	0 1 ... 58 59

	mm	00 01 ... 58 59
Second	s	0 1 ... 58 59
	ss	00 01 ... 58 59
Fractional Second	S	0 1 ... 8 9
	SS	00 01 ... 98 99
	SSS	000 001 ... 998 999
	SSSS ... SSSSSSSSS	000[0..] 001[0..] ... 998[0..] 999[0..]
Time Zone	z or zz	EST CST ... MST PST **Note:** as of 1.6.0, the z/zz format tokens have been deprecated from plain moment objects. Read more about it here. However, they *do* work if you are using a specific time zone with the moment-timezone addon.
	Z	-07:00 -06:00 ... +06:00 +07:00
	ZZ	-0700 -0600 ... +0600 +0700
Unix Timestamp	X	1360013296
Unix Millisecond Timestamp	x	1360013296123

See for more information https://momentjs.com/docs/#/displaying/

This is an optional element.

service

`service` defines a service call to another component to retrieve a list of values and labels that are used for the **choice** and **combobox** controls.

```
service:
  name: tr
  class: TestRun
  path: ../tr/service
  value: TR_ID
  label: TR_FULL_NAME
```

`service` has the following attributes;

- name: defines the module name where the service resides in
- class: defines the base name of the class
- path: defines path and name where the Angular service is defined
- value: defines the field name from the table which is used as the value that is stored in the current table
- label: defined the field name from the table which is shown in the **choice** or **combobox** control

`service` and `resolve-list` are mutually exclusive. This is an optional element.

resolve-list

`resolve-list` defines a list of labels and values that are used for the **choice** and **combobox** controls.

```
resolve-list:
  - label: "Pending"
    value: 0
  - label: "Warning"
    value: 1
  - label: "Manual"
    value: 2
  - label: "Passed"
    value: 3
```

`resolve-list` is a list with the following attributes;

- value: defines the field name from the table which is used as the value that is stored in the current table
- label: defines the field name from the table which is shown in the **choice** or **combobox** control

`service` and `resolve-list` are mutually exclusive. This is an optional element.

`resolve-list` has a simpler format where the value is the key of the `resolve-list` and the value of that key is the value.

```
resolve-list:
  0: "Pending"
  1: "Warning"
  2: "Manual"
  3: "Passed"
```

constant-format

`constant-format` is an expression to format the constant variable with the values in the *constant.py* file. By default the label from the `resolve-list` is used as the constant variable name and the value from the `resolve-list` is used without any formating. This means that in cases where the same label is defined, only the first occurrence shall be in the *constant.py* file. To avoid this the template writer must make the constant names unique. This can be done with `constant-format` expression, there are the following variables available;

- table: table name as defined object -> table -> name
- field: field name as defined object -> column -> field (only the name)
- label: label from the resolve-list
- value: value from the resolve-list

The following simple example creates a constant name preceeded by the field name.

```
constant-format: "'{0} = {1}'.format( '_'.join( [ field, label.upper() ] ), value )"
```

Note: The `constant-format` is an actual Python expression. When an error is detected the traceback is shown to resolve the error.

Note: The internal of gencrud norimaizes the attributes (table, field, label), this means that when it starts with a digit, an underscore `_` is prepended to the attribute, where other characters are used than letters and digits, then those are replaced with a single underscore `_`.

rows

`rows` defines the number of rows that are shown for the **textarea** control. This is an optional element.

cols

`cols` defines the number of columns that are shown for the **textarea** control. This is an optional element.

5.10 dialogtabs & screentabs

```
dialogtabs:
  labels:
    - Remark
    - Users
  grouptag: mat-tab-group
  tabtag: mat-tab
  contenttag: mat-card
  tab:
    - label: Users
      component: app-user-table
      params:
        id: "D_ROLE_ID"
        value: D_ROLE_ID
        mode: "filter"
```

`dialogtabs` and `screentabs` have the same attributes, when the action defines a dialog the `dialogtabs` shall be used for contructing the tabs or for screen the `screentabs` shall be used.

labels

`labels` defines a list of tab captions, the caption of a tab is also used to link a field to a specific tab.
The order of the list is the order of the tabs on the screen or dialog.

grouptag

`grouptag` defines the HTML selector of the Material design tab-group component with supports for basic tab pairs (label + content) This is an optional element. When omitted the default used is 'mat-tab-group'

tabtag

`tabtag` defines the HTML selector of the Material tab component, where the content of the tab is placed in. This is an optional element. When omitted the default used is 'mat-tab'

contenttag

`contenttag` defines an extra HTML selector panel within the tab. This is an optional element. When omitted none is used.

tab

`tab` defines an actual tab with an component, on this tab only the defined component shall be available, no fields from the current table are allowed. This is an optional element.

`tab` have de following attributes;

- label: the tab label name to insert the component into
- component: the HTML selector of the component
- params: this is a optioal dictionary containing the attributes with values to be injected into the component

5.11 Options

At root level in the file. This is available from gencrud version 1.6.366.

```
options:
  use-module: false
  overwrite: false
  backup: false
  ignore-case-db-ids: false
  lazy-loading: false
```

This is an optional element. When omitted the default for the options is *false*
When an option is included it supersedes the command line option.

- `use-module` enables the creation of an Angular module per template.
- `overwrite` enables the overwriting of the files in the target folder. When overwrite is *false*, no files shall be written if they exist, even when they are changed.
- `backup` makes backups of all the files it overwrites.
- `case-insensitive-db-ids` all database names shall be in lower case.
- `lazy-loading` this option is only effective when `use-module` is enabled. When enabled the module is added as a lazy loaded module instead of direct loaded.

5.12 Extra

At the root level in the file. This is available from gencrud version 1.7.367. This is only required when `use-module` is set to **true**. With `extra` imports to the TypeScript components can be made. As the TypeScript compiler cannot find components that are defined in other shared modules, it is nessarry to import those components or modules for a specific directive, service, or component.

```
extra:
  imports: ...
```

`imports` see for more details **5.12.1 Imports**.

5.12.1 Imports

`imports` is a list of modules or components that needs to be imported within the module.

```
 - module: RoleModule
   path: ./role.module
   where: module
   type: typescript
 - component: StandardComponent
   path: ./role.module
   where: module
   type: typescript
 - module: SomeModule
   path: ./modules/some.module
   where: app
   type: typescript
```

- `module` is the module class name to be imported.
- `component` is the component class name to be imported.
- `path` is the path including the filename where the module or component is defined in.
- `where` is where the import must take place, there are two options:
- *module*: the current module that is being generated
- *app*: the main application module **app.module.ts**

When `where` is omitted the default is *module*

`type` type if import, there are two type of imports;

- *typescript*: the import is intended for the typescript module
- *python*: the import is intended for the python module

When `where` is omitted the default is *typescript*

references

- references:
- app-module:
- filename: app.module.ts
- class: AppModule
- module: app.module
- app-routing:
- filename: app.routingmodule.ts
- class: AppRoutingModule
- module: app.routingmodule