Crudcast Documentation

Release 0.1.0a8

Christopher Davies

Dec 18, 2018

Contents:

CHAPTER 1

Getting started with Crudcast

Crudcast is a REST API generator that makes it extremely simple to create a fully-functioning RESTful API

1.1 Installing Crudcast

Crudcast can be installed with *pip*

```
pip install crudcast
```

1.2 Installing MongoDB

Crudcast requires MongoDB - it can be installed and run as follows

```
brew install mongodb
brew service run mongodb
```

1.3 Creating an app

To create an app, simply create a file called *config.yml* and add the following content

```
models:
  publisher:
    fields:
      name:
      authors:
        type: manytomany
        to: author
```

(continues on next page)

(continued from previous page)

```
  author:
    fields:
      name:

  book:
    fields:
      name:
      author:
        type: foreignkey
        to: author
```

1.4 Running your app

In the same path as your *config.yml*, open a terminal and run the following command

```
crudcast
```

Your API is now running on port 5000

Note: To see the full set of available arguments for the *crudcast* command, see here

1.5 Using your app

As well as creating a RESTful API, Crudcast also documents it automatically. Once you've started running your app, you'll be able to see how it works, and test it, using the Swagger view created at this location:

```
http://localhost:5000/api/docs
```

1.6 Adding users and authentication

Crudcast also provides an out of the box solution for managing users and basic authentication. To turn on the user module, simply add the line *users:* to your *config.yml* as a root element

```
models:
    # model definitions go here
    ...

users: <= this line
```

Adding this option will create routes for creating, viewing, updating and deleting users. These routes will be documented in the swagger view. However, keep in mind that in order to user these routes, you need to be authenticated. You therefore need to create a user using the command line, before you can use them:

```
crudcast --create-user

Enter a username: chris
Enter password:
Confirm password:
```

You can now use the above user to create other users via the API.

To require users to authenticate in order to access other routes, simply annotate your models as follows:

```
models:
  thing:
    fields:
      name:

    auth_type: basic
```

The above will ensure that you must be authenticated using Basic authorization in order to access all *thing* routes

Note: More auth types will be added in future versions of crudcast

1.7 Adding custom methods

It's also possible to create API methods with your own custom code - methods can be defined as follows:

```
methods:
- path: say-hello/<string:arg1>
  resource: HelloResource
  file: hello.py
```

hello.py:

```
from crudcast.resources import Resource

class HelloResource(Resource):
    def get(self, arg1):
        return {'hello': arg1}
```

The above will create an additional route as *basePath/say-hello/{arg1}* which returns a simple response. For more help on this function, see **'extra_methods'_**

CHAPTER 2

Running your Crudcast app

Once you've created your Crudcast *config.yml* file, you can start your app with the following command (from the same folder as your config file):

```
crudcast
```

2.1 Futher options

2.1.1 Config file

If your *config.yml* is in a different location, you can point Crudcast at it like this:

```
crudcast --config-file /path/to/custom/config.yml
```

2.1.2 Host

By default, Crudcast sets the host name to *0.0.0.0*. This can be overwritten as follows:

```
crudcast --host 127.0.0.1
```

2.1.3 Port

Similary, the default port (*5000*) can also be modified

```
crudcast --port 9000
```

2.1.4 Import name

It's also possible to specify Flask's import_name parameter

```
crudcast --import-name myapplication
```

2.1.5 Debug mode

By default, Debug mode is set to False. You can enable it like this

```
crudcast --debug
```

2.1.6 Disable dotenv loading

By default, Flask looks for environmental variables in the nearest `.env` or `.flaskenv` file. To disable this behaviour, use the following command

```
crudcast --no-load-dotenv
```

2.1.7 Creating users

If you have enable the *users* model, you can create new users via the command line, as follows:

```
crudcast --create-user

Enter a username: chris
Enter password:
Confirm password:
```

Note: Using this switch doesn't start the server - it will just exit once the user has been created

CHAPTER 3

The YAML config file

The config file contains the entire definition for your application. The config file should be in YAML format. By default, Crudcast looks for a file called *config.yml* in the current folder on startup. For example:

```
$ ls -l
-rw-r--r--   1 chris.davies  staff    229 Dec 10 08:42 config.yml

$ crudcast
* Serving Flask app "Crudcast" (lazy loading)
* Environment: production
WARNING: Do not use the development server in a production environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

You can also specify a custom config file location

```
crudcast --config-file /path/to/config.yml
```

The full set of config options are documented below

3.1 Models

Here is a very simple configuration that generates two different objects - *person* and *address*:

```
models:
  person:
    fields:
      first_name:
      last_name:
        required: true
      email_address:
        unique: true
```

(continues on next page)

(continued from previous page)

```
        required: true
      age:
        type: number

  address:
    fields:
      house_number:
        type: number
      street_name:
      city:
      post_code:
      person:
        type: foreignkey
        to: person
        required: true
```

Each object has a *fields* attribute - this is an array of attributes to be applied to your object. In the case of the above objects, the *person* model has *first_name*, *last_name*, *email_address* and *age* fields. Note that:

- *first_name* has no options supplied, and therefore inherits the default field config - it is treated as a string field, which can be left blank.
- *last_name* also has no *type* specified, which means it too defaults to a string field. However, as we have set *required: true*, a validation error will get generated if we attempt to create or update a *person* object without providing this value
- *email_address* is another *required* text field. However, this field also adds the *unique: true* property - this means that on creating/updating a *person*, this field is validated to make sure that no other *person* objects with the same *email_address* already exist
- *age* specifies a different type - *number*. This means that the field will be validated to ensure that the input on creating/updating an object is numeric

The *address* object stores the simple text/numeric properties *house_number*, *street_name*, *city*, and *post_code*. However, this model also has a *person* field which uses the *foreignkey* field type. This field type allows you to create a relationship between one object and another. Note that when using the *foreignkey* field type, you must also provide a *to* value which corresponds to the name of another model. This allows our app to link *person* objects with *address* objects

Note: There are a number of other different field types not listed here. For full details, refer to the fields page

3.2 Database configuration

Crudcast requires MongoDB, and the config file is used to tell Crudcast where to look for the MongoDB, and what to call the database. It can be set like as follows:

```
mongo_url: mongodb://localhost:27017/
db_name: database
```

Note: The above example shows the default values. If you do not set these values, the above MongoDB configuration would still apply

3.3 Users

To enable the user model, add *users:* to the top level of your *config.yml*:

```
users:
```

This parameter does not currently support any additional options

3.4 Authentication

You can require users to authenticate using basic auth to access the routes of a specific model by adding *auth_type: basic* to your model config

```
models:
  thing:
    fields:
      name:

    auth_type: basic
```

This will ensure that all users must be logged in to access the *thing* routes.

> **Warning:** Setting an *auth_type* while not enabling the *user* model (see above) will cause your routes to be completely inaccessible

3.5 Documenting your API

Crudcast supports the OpenAPI specification, and automatically generates a Swagger view showing your fully documented API, and allowing you to test it. You can update the default Swagger documentation generated by Crudcast by adding a *swagger:* parameter to your config file.

```
swagger:
    swagger: 2.0
    basePath: /api
    url: /api/docs
    info:
        description: This is an API automatically generated by crudcast
        version: 1.0.0
        title: My Crudcast app
```

Note that you can also add a *description* value at a model level:

```
models:
  person:
    description: personal details <==
    fields:
      first_name:
      ...
```

Warning: The *basePath* is very important here - if you change this, all your routes will also change (not just in the documentation). Note that you must not include slashes in the middle of this value, e.g. */api/v1* is invalid

Note: The *url* value refers to the location at which your swagger view is hosted

CHAPTER 4

Field types

This page documents all the different field types available in Crudcast

4.1 Manual fields

Manual fields refers to any model field that can be updated using the REST API.

Note: the *required* and *unique* options

All manual fields support the *required* and *unique* options. *required* fields are fields that cannot be null or blank. *unique* fields are fields which must have a unique value for the parent model. Note that if a field is *unique*, but not required, the uniqueness validation is not performed on null/blank inputs

4.1.1 String fields

String fields are the simplest field type, and accept string inputs only

Usage:

```
models:
    my_model:
        my_string_field:
            type: string
```

Note: The **String field** is the default field type. If you do not specify the *type* option, the field will be treated as a string field. Adding the *type: string* parameter, as shown above, is not necessary

String fields do not support any additional options beyond the *required* or *unique* properties

4.1.2 DateTime fields

Date time fields are used to store date and time information against a model

Usage:

```
models:
    my_model:
        my_date_time_field:
            type: datetime
```

By default, DateTime fields accept date time strings in the format *2018-12-29 09:55:00.123*. You may change the default format using the *format_string* option:

```
my_date_time_field:
    type: datetime
    format_string: '%d/%m/%y %H:%M'
```

The above *format_string* would accept the input *21/11/06 16:30*. The *format_string* parameter accepts any valid formatting string supported by Python's *datetime* library - see here to see how to build a valid value for this option

4.1.3 Number fields

Number fields are just like string fields, but they only accept numeric input

Usage:

```
models:
    my_model:
        my_number_field:
            type: number
```

Note: Number fields can accept both integers, e.g. *1*, *2*, *3* or decimal/float values, e,g. *1.23*, *3.45*

Number fields do not support any additional options beyond the *required* or *unique* properties

4.1.4 Boolean fields

Boolean fields can store a *true* or *false* input

Usage:

```
models:
    my_model:
        my_number_field:
            type: boolean
```

Warning: Although it will theoretically support it, its generally a bad idea to use the *unique* parameter on boolean fields

Boolean fields do not support any additional options beyond the *required* or *unique* properties

4.1.5 Foreign key fields

Foreign key fields can be used to store relationships between objects of different types

Usage:

```
models:
    my_model_one:
        name:

    my_model_two:
        foreignkey_field:
            type: foreignkey
            to: my_model_one
```

When specifying this field type, you must provide the *to* parameter, which must be the name of another defined model in the same config file

Foreign key fields are just like text fields, but they store the validated *_id* of another object.

4.2 Many to Many fields

Many to many fields are like Foreign key fields, but for multiple objects. You can associate an object with multiple others by using a many to many field

```
models:
    my_model_one:
        name:

    my_model_two:
        many_to_many_field:
            type: manytomany
            to: my_model_one
```

As with Foreign key fields, the *to* field must be supplied, and the IDs are validated when the object is saved. The main difference is that the many to many field stores an array of validated IDs

4.3 Auto fields

As well as manual fields, Crudcast also supports a number of auto fields - fields that are automatically populated when an object is saved. This type of field does not accept end user input

4.3.1 _id field

Crudcast uses MongoDB, which allocates a unique Object ID to every document stored in the database. This ID is set automatically when the document is created, and is used to reference documents in subsequent lookups. It is automatically addedd to all models and cannot be remove

4.3.2 Auto fields

Auto fields attach an incremental number to each object in your collection

Usage:

```
models:
    my_model:
        my_auto_field:
            type: autofield
```

Sending a GET request to the above model would return a response that looked something like this:

```
[{
    "id": "",
    "my_auto_field": 1
}, {
    "id": "",
    "my_auto_field": 2
}, {
    "id": "",
    "my_auto_field": 3
}]
```

Auto fields do not support any additional options

4.3.3 Auto date time field

This field stores a date/time automatically - the value is set when the object is saved

Usage:

```
models:
    my_model:
        my_auto_date_time_field:
            type: auto_datetime
```

The auto date time field also supports the *create_only* option, which is set to *False* by default. If this is set to *True*, then the value is only set when the object is created - Any further updates to the object would not change the value. Let's look at this example:

```
models:
    my_model:
        name:
        date_created:
            type: auto_datetime
            create_only: true
        date_changed:
            type: auto_datetime
```

In the case of the above example, when you create a new instance of the *my_model* object, the *date_created* and *date_changed* dates would be set to the current date and time. However, on saving the same instance at a later date, the *date_changed* field would be updated, but the *date_created* field would remain unchanged

CHAPTER 5

Running Crudcast in production

We recommend using Docker to run Crudcast in production. Here's a very simple example using docker-compose:

5.1 Folder structure

Create the following three files in the same folder:

```
- docker-compose.yml
- Dockerfile
- config.yml
```

5.2 Docker-compose.yml

```yml
# docker-compose.yml
version: '3'

services:
  crudcast:
    build: ./

    ports:
      - "5000:5000"

    links:
     - mongodb

  mongodb:
    image: mongo
```

5.3 Dockerfile

```
FROM python:3.6-alpine
RUN pip install
COPY config.yml /
WORKDIR /
CMD crudcast --host 0.0.0.0
```

5.4 config.yml

5.5 How to run it

CHAPTER 6

Crudcast API reference

6.1 Crudcast App

class crudcast.app.**CrudcastApp**(*import_name*, *config_file*, ***kwargs*)

The main Crudcast app object, which extends the Flask app

Parameters

- **import_name** – The *import_name* parameter for Flask
- **config_file** – Local path to a valid *config.yml*
- **kwargs** – additional arguments to be passed to Flask

client = None

crudcast_config = {'db_name': 'database', 'mongo_url': 'mongodb://localhost:27017/',

db = None

get_definition(*model*)

Returns the model definition based on the model's fields

Return type dict

get_instance_path(*model*)

Returns the path entries for instance level calls such as PUT, DELETE, RETRIEVE

Return type dict

get_model_path(*model*)

Generates a path entry for the swagger view for each model

Return type dict

get_security_definitions()

get_swagger_ui_view()

Creates a swagger view using *flask_swagger_ui*

Returns a swagger view

get_tag(*model*)
Returns the tag name/description to be used in the swagger view for each model

Return type dict

methods = {}

models = {}

set_crudcast_config(*config_file*)
Overwrites *self.crudcast_config* with the content of the config file

Parameters **config_file** – a local file path to a valid config file

swagger_config
Retrieves the complete swagger configuration

Return type dict

user_config = {'fields': {}, 'options': {'auth_type': 'basic'}}

user_manager = None

6.2 Manual Fields

6.2.1 BaseField

class crudcast.fields.**BaseField**(*name*, *model*, ***options*)
Base class for field objects. All fields must extend this class

Parameters

- **name** – Field name.
- **model** (*models.Model*) – the Model object to which the field belongs
- **options** – field options. The available options vary based on field type. Only the *required* and *unique* options are implemented at this level

auto = False

type = ''

validate(*data*, *_id=None*)
When creating or updating an object. This method validates the inputs

Parameters

- **data** (*dict*) – data to validate
- **_id** (*str or None*) – ID as a string, or None in case of new objects

Return type dict

6.2.2 StringField

class crudcast.fields.**StringField**(*name*, ***options*)
String input field

Parameters

- **name** – field name
- **unique** (*Boolean*) – if True, no new documents can be created if there is another document in the collection has the same value for this field as the one being provided

type = 'string'

validate(*data*, *_id=None*)
Validates that the input value is a string

6.2.3 DateTimeField

class crudcast.fields.**DateTimeField**(*name*, ***options*)
A field for storing datetime objects

Parameters

- **name** – field name
- **format_string** – format string to be used for validating input. Defaults to '%Y-%m-%d %H:%M:%S.%f' (ISO format)

validate(*data*, *_id=None*)
Validates the input string by asserting that the time format is valid

6.2.4 NumberField

class crudcast.fields.**NumberField**(*name*, *model*, ***options*)
Numeric input field.

type = 'number'

validate(*data*, *_id=None*)
Asserts that the input is numeric

6.2.5 BooleanField

class crudcast.fields.**BooleanField**(*name*, *model*, ***options*)
A simple true/false field

type = 'boolean'

validate(*data*, *_id=None*)
Checks that the input value is a Boolean

6.2.6 ForeignKeyField

class crudcast.fields.**ForeignKeyField**(*name*, *model*, ***options*)
A field that points at another collection in the database

Parameters

- **name** – field name
- **model** – parent model
- **to** – the related model's name

get_related(*related_model_name*)
Returns the related model

Parameters **related_model_name** – the name of another model in the app

Return type crudcast.models.Model

type = 'object'

validate(*data*, *_id=None*)
Checks to see if a document in the related model's collection matches the ID provided as *data*

6.2.7 ManyToManyField

class crudcast.fields.**ManyToManyField**(*name*, *model*, ***options*)

type = 'array'

validate(*data*, *_id=None*)
Checks to see if a document in the related model's collection matches the ID provided as *data*

6.3 Auto fields

6.3.1 AutoBaseField

class crudcast.fields.**AutoBaseField**(*name*, *model*, ***options*)
An automatically generated field. When *Model.create()* is called, the fields will be set automatically - the *.set()* method must be implemented in order for this to happen. Any attempt to set the value of this field manually will generate a ValidationError

auto = True

get_original(*_id*)
Helper function that returns the original value of the object, before it is updated. If you want to leave a value unchanged on update, the *.set()* method should call and return this function (otherwise the field value will be set to *None*)

Parameters **_id** – the object id as a string

set(*_id=None*)
This method must be extended by all child classes. If it returns None, the value will not be changed

Parameters **_id** – If an object is being updated, the _id will be provided

validate(*data*, *_id=None*)
Any attempt to set the value of an auto field will raise a validation error

6.3.2 AutoField

class crudcast.fields.**AutoField**(*name*, *model*, ***options*)
An auto field, e.g. one that creates a sequential number, e.g. 1, 2, 3, etc. Cannot be set manually - fields that extend this must implement the *.set()* method

set(*_id=None*)
Returns the collection length + 1 when the object is created

6.3.3 AutoDateTime

class crudcast.fields.**AutoDateTimeField**(*name*, *model*, ***options*)

set(*_id=None*)
Sets the field value to the current date/time when the object is created. If the object is being updated, and the value of *self.create_only* is *True*, then the date/time will be updated to the current time

6.4 Exceptions

6.4.1 ValidationError

class crudcast.exceptions.**ValidationError**(*message*, *field=None*, *status_code=None*)
When invalid data is sent to via POST or PUT, this exception gets raised

Parameters

- **message** – The error message
- **field** – The name of the field that raised the exception, if applicable
- **status_code** – allows overwriting of the default bad request status code, 400

status_code = 400

to_dict()
Returns a REST-friendly API response

6.4.2 ConfigException

class crudcast.exceptions.**ConfigException**
This exception gets raised if the config file is invalid

6.5 Resources

6.5.1 Resource

class crudcast.resources.**Resource**(*api=None*, **args*, ***kwargs*)
Extension to the default *flask_restplus.BaseResource* which sets the *app* paramter, which is used for model creation

app = None

check_auth()

model = None

classmethod set_app(*app*)
Sets the value of __class__.app

6.5.2 ModelResource

class crudcast.resources.**ModelResource**(*api=None*, **args*, ***kwargs*)

Model level resources - implements all REST methods for paths with no ID

get(*model_name*)

Lists all instances of a given model

Parameters **model_name** – the name of the model, as it appears in the config file

Returns a list of instances of the model object

methods = {'POST', 'GET'}

post(*model_name*)

Create an instance of a model

Parameters **model_name** – the name of the model, as it appears in the config file

Returns details of the created instance

6.5.3 InstanceResource

class crudcast.resources.**InstanceResource**(*api=None*, **args*, ***kwargs*)

Instance level resources - implements all REST methods for paths with an ID

delete(*model_name*, *_id*)

Delete a single instance by its ID

Parameters

- **model_name** – the name of the model, as it appears in the config file
- **_id** – MongoDB _id string

get(*model_name*, *_id*)

Retrieve a single instance by its ID

Parameters

- **model_name** – the name of the model, as it appears in the config file
- **_id** – MongoDB _id string

Returns the MongoDB document

methods = {'DELETE', 'PUT', 'GET'}

put(*model_name*, *_id*)

Update a single instance by its ID

Parameters

- **model_name** – the name of the model, as it appears in the config file
- **_id** – MongoDB _id string

Returns the MongoDB document

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

T

U

V