Logger

Component Management System API

A custom logger is implemented to log the events and errors in the system. The logger is implemented using the logging module in python. The logger is implemented in the component_management_system package and is used to log the events and errors in the system. The logger is implemented in the logger.py file in the component_management_system package.

Loggers:-

1. **Root Logger**: The root logger is the default logger in the system. It is used to log the events and errors in the system. The root logger is implemented in the logger.py file in the component_management_system package.

2. **Elasitcsearch Logger**: The elasticsearch logger is used to log the events and errors in the system. The elasticsearch logger is implemented in the logger.py file in the component_management_system package.

3. **Flask Logger**: The flask logger is used to log the events and errors in the system. The flask logger is implemented in the logger.py file in the component_management_system package.

Custom handlers are implemented to log the events and errors in the system. The custom handlers are implemented in the handlers.py file in the component_management_system package. The custom handlers are implemented to log the events and errors in the system. The custom handlers are implemented in the handlers.py file in the component_management_system package.

Handlers:-

1. **Stream Handler**: The stream handler is used to log the events and errors in the system. The stream handler is implemented in the handlers.py file in the component_management_system package.

2. **Rotating File Handler**: The rotating file handler is used to log the events and errors in the system. The rotating file handler is implemented in the handlers.py file in the component_management_system package.

3. **Flask Handler**: The flask handler is used to log the events and errors in the system. The flask handler is implemented in the handlers.py file in the component_management_system package.

Components Library Addon(Client Application)

Logging for the Client Application used FreeCAD logger.

ElasticSearch

The ElasticSearchBase class is an abstract base class for Elasticsearch models. It is used to create a simple Object Relational Mapper (ORM) for Elasticsearch. The class is implemented in the base.py file in the component_management_system package.

The ElasticSearchBase class has the following attributes: 1. __abstract__ (bool): Indicates if the class is abstract 2. __es: The Elasticsearch client 3. __schema (SQLAlchemyAutoSchema): The schema for a single instance 4. __schema_many (SQLAlchemyAutoSchema): The schema for multiple instances

The ElasticSearchBase class has the following methods: 1. __init__(*args, **kwargs): Initializes the ElasticsearchBase instance 2. create(): Creates a new instance in the database and indexes it in Elasticsearch 3. update(field_name, value): Updates the instance in the database and Elasticsearch based on the specified field and value 4. delete(field_name, value): Deletes the instance from the database and Elasticsearch based on the specified field and value 5. elasticsearch(search_key): Performs an Elasticsearch search and returns a set of matching names 6. set_schemas(schema, schema_many): Sets the schemas for the ElasticsearchBase class

The ElasticSearchBase class is an abstract base class, so it cannot be instantiated directly. It is designed to be subclassed by other classes that represent specific Elasticsearch models.

Dynamic Attributes

This project revolves around making components in a system more flexible. Imagine you have different types of components like *screws* and *LEDs*. These components have various details like *material*, *size*, *color*, *voltage*, and more. The catch is, each type of component has different details unique to them.

Traditionally, systems force all components to fit into the same set of details, which doesn't work well. So, we're introducing a system where users can add and adjust details for each component as they need.

Advantages:

1. Get Specific with Component Details:

Users can now be very precise about what makes each component special. This means better and more accurate descriptions for each one.

1. Keep Component Names Simple:

Instead of complicated names, we can use simpler names for components. The extra details are taken care of by our system, making everything clearer.

1. Find Components Easily:

Searching for components becomes a breeze. You can use all sorts of details to find exactly what you need.

Dynamic Attributes in API:

Dynamic attributes is implemented as **key, value pair** which uses elasticsearch. It has **one-to-many** mapping with the Metadata Model.

Uses:

New searching system is backwards compatible. User searching for a component need to enter the name of the component in the search bar. For improving the searching results for more accurate results, user can make use of attributes searching feature by following a simple **syntax** :

```
component_name key:value
```

- Each search query must contain **component_name**
- **component_name** and the attributes must be separated by a single *(space)* character.
- An **attribute** consists of a key and a value connected with a single :*(colon)* character.
- **Key** can be used stand alone but must be followed by a single :*(colon)* character.

- **Value** can be used stand alone but must be preceded by a single :*(colon)* character.

- ### Using key:value pair

  User can search with any number of **key:value** pairs.

```
component_name key1:value1 key2:value2 key3:value3...
```

1. ### Using only key

   If user wants to search only those components which has a specific **key** defined in it, it can be done in the following way:

```
component_name key1: key2: key3:...
```

1. ### Using only value

   If user wants to search only those components which has a specific **value** defined in it, it can be done in the following way:

```
component_name :value1 :value2 :value3...
```

1. ### Using combination

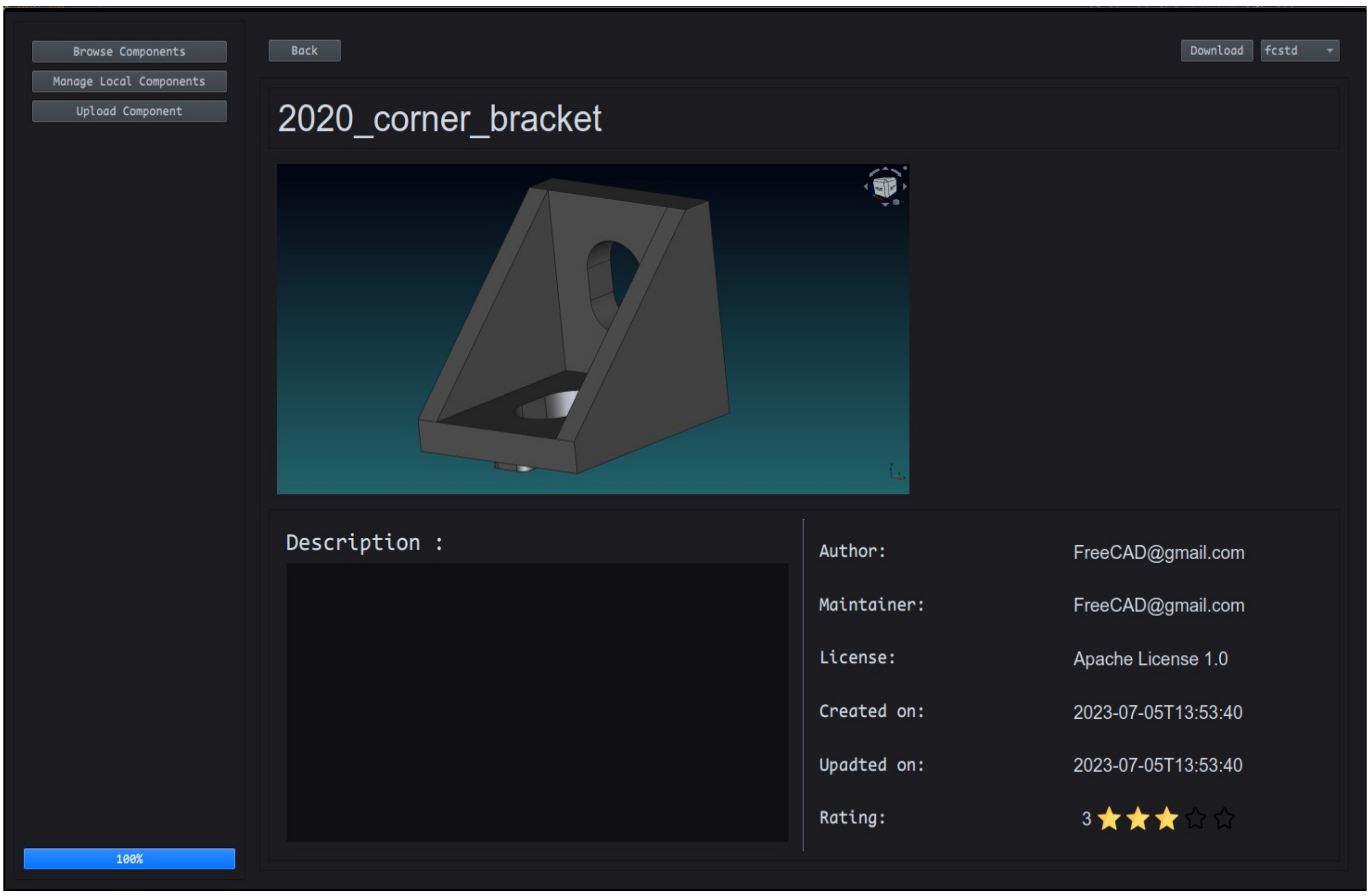
   A combination of **key:value** pairs and **keys** and **values** alone can be used simply by combining the syntax.

```
component_name key1:value1 key2:value2 key3:value3 :value4 :value5 key4: key5: value6 key:6...
```

!!! note The order of attributes **key:value, key:** & **:value** does not matter and has no effect on the results.

UI changes:

Before:

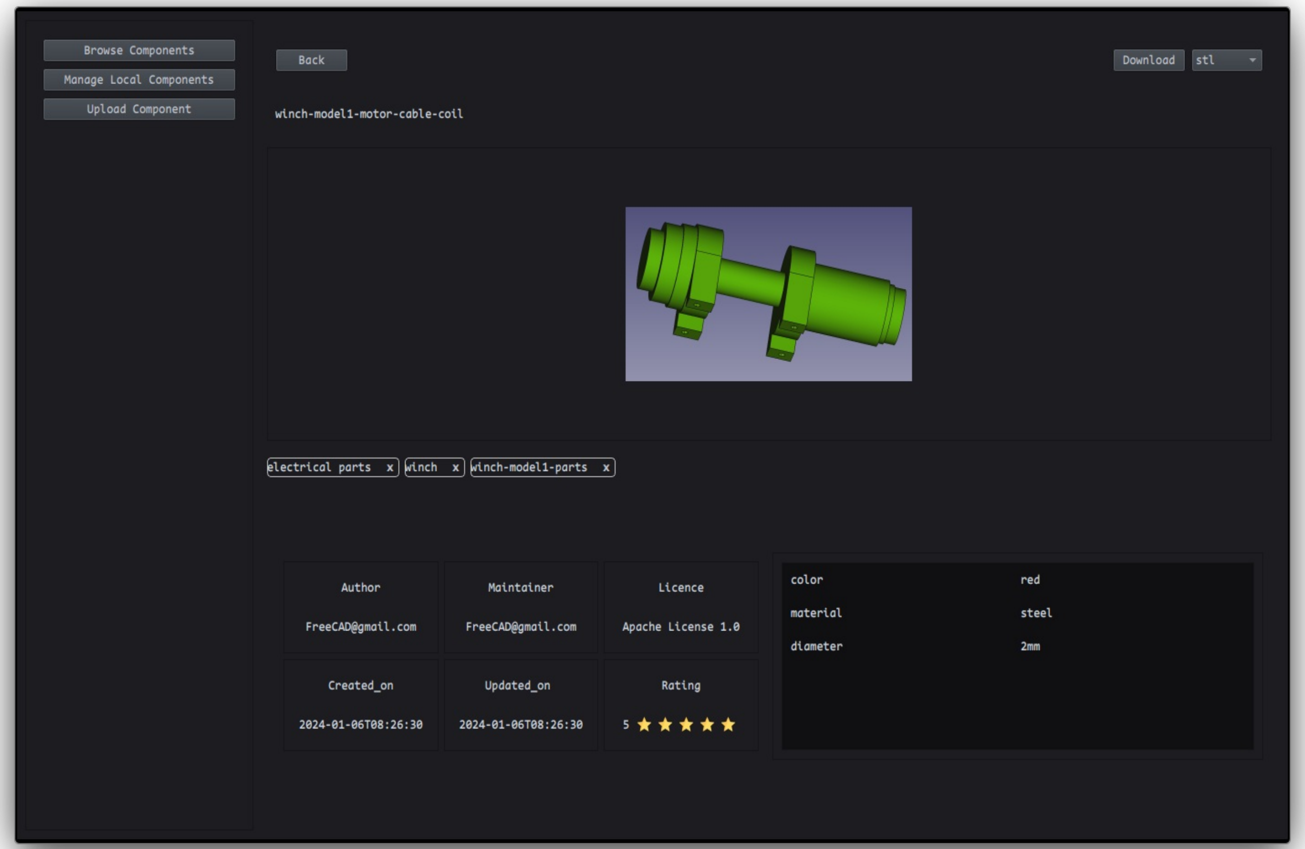


2020_corner_bracket

Description :

Author:	FreeCAD@gmail.com
Maintainer:	FreeCAD@gmail.com
License:	Apache License 1.0
Created on:	2023-07-05T13:53:40
Updated on:	2023-07-05T13:53:40
Rating:	3 ★★★☆☆

After:



winch-model1-motor-cable-coil

Author	Maintainer	Licence
FreeCAD@gmail.com	FreeCAD@gmail.com	Apache License 1.0
Created_on	Updated_on	Rating
2024-01-06T08:26:30	2024-01-06T08:26:30	5 ★★★★★

color	red
material	steel
diameter	2mm

Old User Interface VS New User Interface

1. The thumbnail is small, centre aligned and is with a scrollable widget so that more thumbnail can be accommodated

easily.
2. The size of the description box of thumbnail is resized and made small to save necessary space.
3. The format of the meta data is changed in a manner that it is more space efficient with better visuals.
4. A scrollable area is added for the dynamic attributes.
5. The tags of the component are now listed in the detailed UI.

Validation

API Defination Level

pattern

The pattern keyword is used to restrict the value of a string to a specific regular expression.

lenght limit

maxLenght and minLenght are used to restrict the length of the string.

data validation

validations of urls, emails, phone numbers, etc. are done using regex and other libraries at the model level.