Customisable Django Admin Documentation

Release in development

Leander Hach

May 27, 2018

Contents:

CHAPTER 1

Models

1.1 Basic Model Setup

Because models that are passed to CDA might already exist, the only addition that is initially needed is a managed field to allow CDA to use the field

```
managed = models.BooleanField(default=True)
```

1.2 Model Fields

Once a model is managed, the fields that can be used to geneerate front-end content need to be specified. This is done primarily because not every field is relevant when generating content. for example, a User model might display a name field, and an Id, but not a password field (if explicitly defined).

```
def compatible_fields(self):
    l = [self.field1, self.field2, self.field3]
    return l
```

1.3 Model Types

The final step needed to complete the model setup is telling the CDA what kind of data to expect. For example a a Product model that only contains a name and stock data would not be suitable for a pie chart, while a model that tracks sales data over time would be well-suited to a line chart.

```
def type_flags():
    flags = ["table", "pieChart"]
    return flags
```

There are several different model types avaliable, listed below:

1.3.1 Table

The table display type is very simple. It takes the entire list of models from comptabile_fields, in the order that they are passed, and then adds them to a table with columns named after each field.

1.3.2 pieChart

1.3.3 lineGraph

1.3.4 number

1.3.5 numberMinMax

It is important to note that for the first 3 model types listed above, whatever the fields or types, the process will be applied to every instance of a given model, so if a Product model with 3 instances is made into a table that table will display 3 rows, one per model instance.

CHAPTER 2

Customisation

TODO: Make it customisable ;)

CHAPTER 3

Limitations

3.1 Model Types

At this time, there is no support for adding more model types although this functionality is planeed. Because of this, the current methods of displaying data re limited to the initally provided six different types.

3.2 Data Parsing

some model types will only display a single point of data, even if there is more than one field provided in the compatible_fields method. In this case, the model will use the first appropriate field that matches the field type expected. For example, if a number model type is given the following model:

```
class DailySummary(models.Model):
    managed = models.BooleanField(default=True)
    name = models.CharField() #Daily Summary
    total = models.FloatField() #$1234.56
    data = models.DateTimeField()

    compatible_fields(self):
        l = [self.name, self.total, self.date]
        return l
```

The number model type would grab the first avaliable field that matches a number format, in this case a FloatField.

CHAPTER 4

Important Considerations

4.1 Model Names

The CDA uses the table names of the managed models when allowing users to select which model to pull data from. because of this, it is advised that you specifiy a custom table name for the model if it contains more than one word.

This is due to the automatic case change that occurs when the default table names are generated, ie:

```
dailySummary becomes dailysummary
```

simply camel-case the intended table name as usual:

```
class Meta:
    db_table = "dailySummary"
```

This will return the following : Daily Summary.

If the model is a single word this is not required.

4.2 Usability

Because the front-end user is presented with the exact table name that was initally given to the model, it is best to avoid complex table names, or, if they are required, to provide a more easily legible alternative in the db_table option.

CHAPTER 5

Introduction

This project was concieved as a pontential alternative to the kinds of admin panels commonly found on sites like Wordpress or Amazon, which allow site admins to control large portions of their site without getting into the code too much. At the same time, it was recognised that CDA might be integrated into existing django projects, which nessecitates the use of a much more flexible strucuture so that it can be moulded to fit as may use cases as possible.

CHAPTER 6

Requirements

As is included in the name itself, this project can only be integrated or built into a Django app. It alsoe requires Python 3.XX and a Django version <2.

CHAPTER 7

Credit and Notes

This project uses Chart.js for all graph represenations

All examples in these docs are based on a fictional eCommerce site, which should help draw a link between the backend and frontend representations.