Cait.Rocks Documentation

Release 0.8

Edwin Lee

Jan 15, 2019

Contents:

This documentation captures code information, tips, tricks, and other discussions, based around the code for Cait.Rocks.

CHAPTER 1

Database Models

1.1 Recipes

The recipe model defines a recipe by defining new fields for title, directions, etc., defining relationships to User instances for tracking who owns the recipe, and relying on Ingredient instances to be back referenced to recipes through foreign keys. The recipe model has a function called get_absolute_url which is used to retrieve the url of the current recipe's recipe_detail page.

class recipes.models.recipe.**Recipe**(**args*, ***kwargs*)

Bases: django.db.models.base.Model

The recipe model captures the heart of this project, defining what a recipe is, including a title, directions, and ingredients. Title, directions, recipe type, and other scalar fields are defined here, while Ingredients are defined on the Ingredient model itself, to allow a variable number of them, via foreign key.

exception DoesNotExist

Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned

Bases: django.core.exceptions.MultipleObjectsReturned

RECIPE_TYPE_CHOICES = ((u'10', u'Unknown'), (u'30', u'Entree'), (u'35', u'Salad-Entree

This tuple of tuples allows referencing display values for each recipe type via enum string values

created_date

The created and modified date fields allow us to automatically track timestamps, internally the field is updated on the field's pre_save() method.

creator

The creator field is a foreign key, allowing us to link this recipe to a specific User instance

creator_id

A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day01recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day01recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day02recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day02recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day03recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day03recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day04recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day04recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day05recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day05recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day06recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day06recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day07recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day07recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day08recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day08recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day09recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day09recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day10recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day10recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day11recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day11recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day12recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day12recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day13recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day13recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day14recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day14recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day15recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day15recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day16recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day16recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day17recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day17recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day18recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day18recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day19recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day19recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day20recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day20recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day21recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day21recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day22recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day22recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day23recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day23recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day24recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day24recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day25recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day25recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day26recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day26recipe1

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day27recipe0

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day27recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day28recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day28recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day29recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day29recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day30recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day30recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day31recipe0
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

day31recipe1
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

directions
The directions field is a free form text field defining the directions for making this

get_absolute_url()
Gets the URL path to this recipe's nice view page, not the API url!

Returns string

get_next_by_created_date(***morekwargs*)

get_next_by_modified_date(***morekwargs*)

static get_poor_recipes(*request_user_id=None*)

get_previous_by_created_date(***morekwargs*)

get_previous_by_modified_date(***morekwargs*)

get_recipe_type_display(***morekwargs*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

image
The image field contains image data to allow showing off the beauty of this recipe

ingredients
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

is_poor()

modified_date
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

recipe_type
The recipe type choice field captures the type of recipe

title
The title field is required to be a unique string across all Recipe instances

class recipes.models.recipe.**RecipeTypes**
Bases: object

This class is a list of strings used to internally reference and store different recipe types

DESSERT = u'70'

DRINK = u'60'

ENTREE = u'30'

SALAD = u'50'

SALAD_ENTREE = u'35'

SAUCE_DRESSING = u'90'

SEASONING = u'100'

SIDE_DISH = u'80'

SOUP = u'40'

UNKNOWN = u'10'

1.2 Ingredients

The ingredient model is a simple model containing member fields that are used to describe the ingredient class (amount, description, etc.). The members are generally given defaults to allow flexible capabilities. The only relationship on the ingredient class is a ForeignKey to the Recipe model to allow connecting the ingredient to a single Recipe instance. (See notes for more detail on that connection)

class recipes.models.ingredient.**Ingredient**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This class describes a single ingredient, including amount, measurement, and item description. The class includes model fields to describe the ingredient, plus one ForeignKey to a Recipe model instance. The only methods that are added to this model class are the __unicode__ and __str__ methods for representation.

AMOUNT_TYPE_CHOICES = (('0', u''), ('10', u'\u215b'), ('30', u'\t\xbc'), ('40', u'\u21

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

MEASUREMENT_TYPE_CHOICES = (('0', u''), ('5', u'Pinch'), ('10', u'tsp'), ('20', u'Tbsp

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

amount
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_amount_display(***morekwargs*)

get_measurement_display(***morekwargs*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

item_description
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

measurement
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

objects = <django.db.models.manager.Manager object>

recipe
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

recipe_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

1.3 Planning

The planning model defines the monthly recipe plan. Each day of the month has two recipe placeholder spots. Initially this was left more general, but the complexity felt through the roof for what I was trying to accomplish. I've now got it down to this relatively small list of fields. I understand the field list is still pretty lengthy, and that adding more fields to do another recipe each day is a major burden at this point, but I have at least reduced the code required to reference each recipe field down using some getattr and setattr magic.

class recipes.models.planning.**Calendar**(**args*, ***kwargs*)
Bases: django.db.models.base.Model

This class describes a full month of data, including the calendar year/month as well as two recipes per day each day Each recipe field is a ForeignKey to a Recipe and the creator field is a ForeignKey to a User Contains fields, a __str__ method, and two worker methods for getting full monthly data and recipes for one day

exception DoesNotExist
Bases: django.core.exceptions.ObjectDoesNotExist

exception MultipleObjectsReturned
Bases: django.core.exceptions.MultipleObjectsReturned

YEAR_NUMBER_CHOICES = ((u'2018', u'2018'), (u'2019', u'2019'), (u'2020', u'2020'), (u'
This tuple of tuples is a map between Year strings to meaningful display options

creator
The creator field is a ForeignKey to a User model instance to keep track of who created this recipe

creator_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day01recipe0
The dayXYrecipeA fields are optional pointers to recipe instances, two per day for 31 days

day01recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day01recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

child.parent is a ForwardManyToOneDescriptor instance.

day01recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day02recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day02recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day02recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day02recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day03recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day03recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day03recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day03recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day04recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day04recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day04recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day04recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day05recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day05recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day05recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day05recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day06recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day06recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day06recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day06recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day07recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day07recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day07recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day07recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day08recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day08recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day08recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day08recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day09recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day09recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day09recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day09recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day10recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day10recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day10recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day10recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day11recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day11recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day11recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day11recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day12recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day12recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day12recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day12recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day13recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day13recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day13recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day13recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day14recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day14recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day14recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day14recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day15recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day15recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day15recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day15recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day16recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day16recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day16recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day16recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day17recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day17recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day17recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day17recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day18recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day18recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day18recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day18recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day19recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day19recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day19recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day19recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day20recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day20recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day20recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day20recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day21recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day21recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day21recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day21recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day22recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day22recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day22recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day22recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day23recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day23recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day23recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day23recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day24recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day24recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day24recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day24recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day25recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day25recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day25recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day25recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day26recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day26recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day26recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day26recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day27recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day27recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day27recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day27recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day28recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day28recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day28recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day28recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day29recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day29recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day29recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day29recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day30recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day30recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day30recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day30recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day31recipe0
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day31recipe0_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

day31recipe1
Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

day31recipe1_id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

get_month_display(***morekwargs*)

get_monthly_data()
Returns monthly data, including date numbers and recipe ids for each day, if applicable

Returns An array of 4, 5, or 6 weeks, each week is an array of 7 days, each day is a dictionary that has a date_number key that is the actual date or 0, and recipe0 and recipe1 ids, which point to the recipe objects in the database (recipe ids may be None if no recipes are selected)

get_recipes_for_day_of_month(*date_num*)
Returns the recipe ids for the given date num on this calendar instance, if applicable

Parameters **date_num** – The date of the month, 1-31

Returns List of two ids: [recipe01_id, recipe02_id]; these may be None if no recipes are selected or if date_num is out of range

get_year_display(***morekwargs*)

id
A wrapper for a deferred-loading field. When the value is read from this object the first time, the query is executed.

month
The month field stores an integer month, and the list of options is conveniently generated from a library

nickname
The nickname field is a personalized string representation of this month

objects = <django.db.models.manager.Manager object>

year
The year field stores a string reference that is matched to a descriptive year for display

class recipes.models.planning.**YearNumber**
Bases: object

This class is an enum list of years which was created to conveniently create the drop-down for the user when creating a new monthly plan. Using a choice list based on these strings allows for easier validation.

A = u'2018'

B = u'2019'

C = u'2020'

D = u'2021'

E = u'2022'

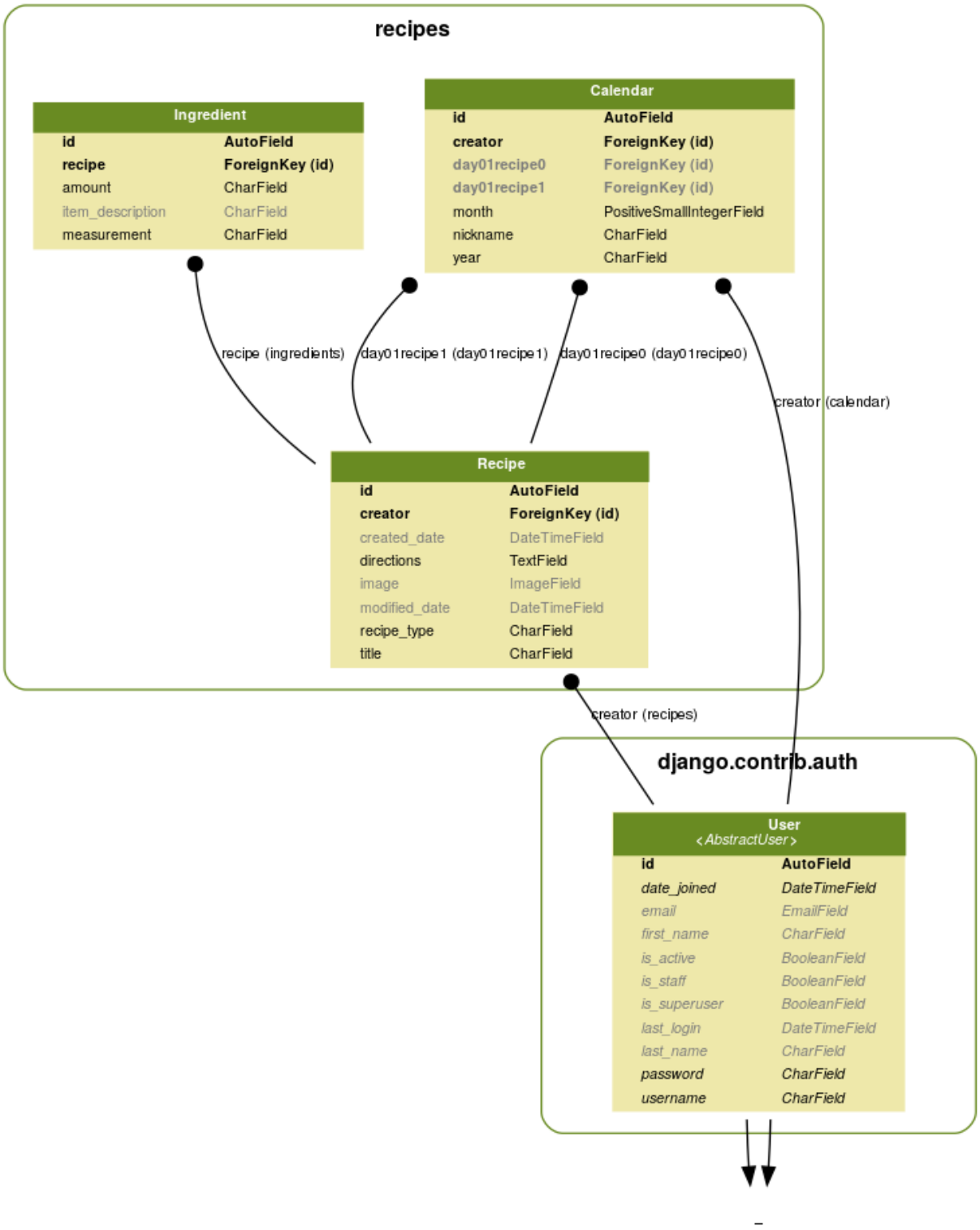
F = u'2023'

G = u'2024'

H = u'2025'

Here is an overview of the model relationships. This view is simplified because in reality there are many more day??recipe? foreign keys, but they are all the same. Repeating them all here overly complicated that portion of the graph. The relationships shown here reveal how rather simple the relationships are:

- Regarding the inter-application relationships, both the calendar and recipe objects have a creator key to User
- Calendar models have foreign keys to Recipe for each day
- Ingredient has a foreign key to a recipe for defining that the ingredient is on that recipe


recipes
Ingredient
id AutoField
recipe ForeignKey (id)
amount CharField
item_description CharField
measurement CharField
Calendar
id AutoField
creator ForeignKey (id)
day01recipe0 ForeignKey (id)
day01recipe1 ForeignKey (id)
month PositiveSmallIntegerField
nickname CharField
year CharField
recipe (ingredients)
day01recipe1 (day01recipe1)
day01recipe0 (day01recipe0)
creator (calendar)
Recipe
id AutoField
creator ForeignKey (id)
created_date DateTimeField
directions TextField
image ImageField
modified_date DateTimeField
recipe_type CharField
title CharField
creator (recipes)
django.contrib.auth
User
<AbstractUser>
id AutoField
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField

CHAPTER 2

Model Serializers

2.1 Serializer Base Classes

This module holds base classes that are derived off of the rest framework serializer base serializers, but with some additional capabilities added on that are useful in aspects here.

class recipes.serializers.base.**CreatorBaseSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

This base class provides a method for getting a creator object serialized properly. This includes creating a serializer method field named creator that should match up with the underlying model being serialized (i.e. it should also have a creator model field). This also includes writing a get_creator method that will be used to extract the creator from the model instance.

creator = None

The creator field is a read-only serializer method field. Internally the base class's to_representation method will call the get_creator method when serializing the object for output

get_creator(*created_instance*)

This worker function will get a creator from a class that has that field, or else return a blank string

Parameters **created_instance** – A Python model object, or really any object that has a creator member variable - not a dict

Returns A string, either the creator name or a blank string

2.2 Recipe Serializers

The recipe serializer is the most complex of the serializers, which isn't saying a lot, as it's not really that complex. The recipe serializer is based on the Recipe model, so most operations flow easily without additional code, but there are a few additions:

- The serializer has an explicit recipe_type field which is used to get the formatted representation of the recipe_type
- There are serializer method fields for getting read-only versions of the creator and absolute_url
- There is an ingredients StringRelatedField, which is a read-only field used to show the ingredients connected with this recipe. This works easily because the related_name on the ForeignKey defined on the Ingredient model is "ingredients".

class recipes.serializers.recipe.**RecipeSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: recipes.serializers.base.CreatorBaseSerializer

This serializer allows direct serialization for recipe objects, with additional keys as needed

class Meta

fields = ('id', 'title', 'recipe_type', 'creator', 'absolute_url', 'ingredients', '

model
alias of recipes.models.recipe.Recipe

get_absolute_url(*recipe_instance*)
This worker function enables us to append the absolute_url field to requests for Recipe model objects :param recipe_instance: A full recipe instance, we really just need the ID :return: The fully formed URL for this recipe instance

2.3 Ingredient Serializers

The ingredient serializer is a basic model serializer that relies on much of the rest framework magic to minimize the amount of extra code required. The only additional fields on the serializer are `amount` and `measurement`, which each contain a `source` argument pointing to the `get_FIELD_display` autogenerated function. This allows the serializer to look up the formatted version of the choice fields.

class recipes.serializers.ingredient.**IngredientSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: rest_framework.serializers.ModelSerializer

This serializer allows direct serialization for ingredient objects, with additional keys for choice fields

class Meta

fields = '__all__'

model
alias of recipes.models.ingredient.Ingredient

2.4 Planning Serializers

The planning, or calendar, serializer is a very simple model based serializer, with only one additional field. The creator serializer method field is a read only field used to get the creator represented on the output.

class recipes.serializers.planning.**CalendarSerializer**(*instance=None*, *data=<class rest_framework.fields.empty>*, ***kwargs*)

Bases: recipes.serializers.base.CreatorBaseSerializer

This serializer allows direct serialization for calendar objects, with additional keys as needed

class Meta

fields = '__all__'

model
alias of recipes.models.planning.Calendar

CHAPTER 3

Application Programming Interface (API) Endpoints

3.1 Recipes API

class recipes.api.recipe.**RecipeViewSet**(***kwargs*)

Bases: rest_framework.mixins.CreateModelMixin, rest_framework.viewsets.ReadOnlyModelViewSet

This class provides the API get and retrieve views for the recipe objects. The class also uses the CreateModelMixin to provide the POST hook, and the create method is overridden to customize the creation step. I think this should probably be moved to the serializer at some point

create(*request*, **args*, ***kwargs*)

The create method is overridden here to allow us to do two things: 1) verify the current user is logged in before allowing creation of a new calendar, and 2) assigning the creator attribute on the request to the currently logged in user before passing the data to the serializer. The serializer should then handle grabbing the creator instance and assigning it to the newly created calendar. I think there is something in there that would make this easier, like overriding the pre-save on the serializer or something.

Parameters

- **request** – An http request
- **args** – Ordered arguments
- **kwargs** – Keyword arguments

Returns JSONResponse with the created object or a failure message

poor_recipes(*request*)

queryset

serializer_class

alias of recipes.serializers.recipe.RecipeSerializer

3.2 Ingredient API

class recipes.api.ingredient.**IngredientViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ReadOnlyModelViewSet

This class provides the API get and retrieve views for the ingredient objects

queryset

serializer_class
alias of recipes.serializers.ingredient.IngredientSerializer

3.3 Planning API

class recipes.api.planning.**CalendarViewSet**(***kwargs*)
Bases: rest_framework.mixins.CreateModelMixin, rest_framework.mixins.DestroyModelMixin, rest_framework.viewsets.ReadOnlyModelViewSet

This class provides the API get and retrieve views for the calendar month objects, plus three workers: - mine, which is simply a GET call that filters by the current user, to list those available for editing - monthly_data, which is used to get the bulk data for a whole month - recipe_id, which is used to put the id onto a

create(*request*, **args*, ***kwargs*)
This function is a custom handler for the POST call into this endpoint This function adds in a creator field to the request data before passing the data to the serializer

Parameters

- **request** – An HTTP request
- **args** – Ordered arguments, none are required here
- **kwargs** – Keyword arguments, none are required here

Returns A JSON response

get_queryset()
This function allows Django to call this model but only retrieve a subset of the database. In this case, it is down-selecting to only the currently logged-in user

Returns A queryset of Calendar objects belonging to the current user

mine(*request*)
This function provides an alternate GET endpoint to get a list of the planning months that are owned by the current user. This is used to list the months available for editing in the planner page. For the months page, we want to expose them all, read-only of course, so that endpoint should just use the regular list method.

Returns A JSON response with all the calendar objects available for this user

monthly_data(*request*, *pk*)
Creates a custom response on the API for getting monthly data, including date numbers, recipes, etc.

Parameters

- **request** – A full django request object
- **pk** – The primary key for this particular calendar instance

Returns

A JSONResponse with two keys: data and num_weeks.

- num_weeks returns the number of weeks in this month for convenience, either 4, 5 or 6
- data is an array of 4, 5, or 6 items, with each item being weekly data. Each weekly data item is an array of 7 items, with each item being daily data. Each daily data item is a dictionary containing keys date_number, recipe0, recipe0title, recipe1, and recipe1title. The date_number key can be "-" to represent this day does not belong in the current month. The recipe0 and recipe1 keys are ids to recipe objects in the database. The recipe0title and recipe1title keys are simply the recipe titles for convenience.

recipe_id(*request*, *pk*)

Sets the recipe for this particular calendar date and recipe id Expects three parameters on the request body: date_num (1-31), daily_recipe_id (0 or 1), and recipe_pk If recipe_pk is 0, that indicates this recipe item should be cleared

Parameters

- **request** – A full django request object
- **pk** – The primary key of the calendar to modify

Returns A JSONResponse object with keys success and message. The status code will also be set accordingly

serializer_class

alias of `recipes.serializers.planning.CalendarSerializer`

CHAPTER 4

Views

For these views, I tried to wrap them in ViewSets where it made sense, but really, that sorta pushed things in the wrong direction. These are not views into the API, these are simply page views. In the future, when we have a single-page Angular-routed view, several of these will not be separate page views, so I won't be updating them now.

4.1 Planning Pages

class recipes.views.planning_pages.**MonthViewSet**(***kwargs*)

Bases: rest_framework.viewsets.ViewSet

This view set provides two page views, a "list" view which shows all the months and a "detail" view which shows just a single month

list(*request*)

Retrieves a list of monthly plans and renders them in a template :param request: An http request object :return: Rendered HTML

retrieve(*request*, *pk=None*)

Retrieves a single monthly plan and renders it in a template :param request: An http request object :param pk: The pk for the month of interest :return: Rendered HTML

class recipes.views.planning_pages.**PlannerViewSet**(***kwargs*)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, rest_framework.viewsets.ViewSet

This view set provides a single view into the planner page. That page is fully dynamic through Angular XHR calls so no context needs to be rendered.

list(*request*)

4.2 Recipe Pages

class recipes.views.recipe_pages.**RecipeViewSet**(***kwargs*)
Bases: rest_framework.viewsets.ViewSet

This view set provides two page views, a "list" view which shows all the recipes and a "detail" view which shows just a single recipe

list(*request*)
Retrieves a list of recipes and renders them in a template

Parameters **request** – An http request object

Returns Rendered HTML

retrieve(*request*, *pk=None*)
Retrieves a single recipe and renders it in a template

Parameters

- **request** – An http request object
- **pk** – The pk for the recipe of interest

Returns Rendered HTML

4.3 Utility Pages

These views are simple standalone pages that are not inside classes, which is appropriate for them.

recipes.views.utility_pages.**handle404**(*request*)
This view is called by Django when the a GET request is made to a page that does not exist. It renders a static page but isn't amenable to making into a simple TemplateView because it must return a 404

Parameters **request** – An HTTP request

Returns A rendered HTML response

recipes.views.utility_pages.**server_version_data**(*request*)
This function provides the "About" page response, including server information

Parameters **request** – An HTTP request

Returns A rendered HTML response

CHAPTER 5

Notes and Discussions

I will use this as a place to drop interesting things I have found during development, or use it as a place to post information for discussions or whatever.

5.1 Serializing Thoughts and Concerns and Etc

I would really like to wrap my head around the best use of serializers, but I'm not quite there yet. One thing I have noted from the inter webs is that people say we should have "light" views, and "heavy" serializers. I know this means that when we need to start customizing the output/processing, we should focus on adding to the serializers, not the views, as much as possible. But sometimes, it seems way easier to just override the create method on the view, for example, because we need access to the request.user. I'm sure there is a better way to do it through overriding something in the serializer, but I'm not sure of it yet.

5.2 ForeignKey to Recipe on Ingredient model

The ingredient model class has a ForeignKey member to a Recipe model instance. This means the ingredient is assigned to a single recipe class forever. This is a bit silly, as an ingredient should just be standalone and potentially applied to multiple recipes. Right? Wait no. That could be bad. If an ingredient gets altered in one recipe, you don't want it being altered in all the recipes that reference it. OK, it's good. Good to talk things through. Thanks Mr. Duck.

5.3 Missing ForeignKey to User on Ingredient model

The ingredient model does not have a foreign key to a User. This seems like a problem given all the other ownership stuff we've put in. I think we need to add it. It will probably be a super light addition once I figure out the serializer stuff.

5.4 Year DropDown Select Box?

Right now we have the year field on the planning class, which is a choice field based on a specific list of years. This is good. However, in the HTML, we still hardwire the same number of years. I think I need to figure out how to properly generate a form out of the month/year/title fields on the planning class. Then the form will automatically use the list created in Python. My concern is that it will be difficult to integrate Angular stuff in there if we start going down that path, so that's something to consider.

5.5 Permissions

I would like to take some time to dig deep into the permissions around my models and views. I want to make sure that only authors of recipes can modify their own (except super users of course). Some of this will come from information on the rest framework tutorial here: http://www.django-rest-framework.org/tutorial/4-authentication-and-permissions/.

- search