Vittles Documentation

Release 0.1

Eric Pierce

November 27, 2012

CONTENTS

Vittles is a Django web application for entering and viewing recipes. Get the source code from Github.

It is still in the early experimental phases, but has basic functionality for entering foods, units, ingredients, recipes, and nutritional information through the Django admin interface, as well as for viewing recipes in an easy-to-read format.

Contents:

CHAPTER

ONE

VITTLES USER MANUAL

Contents:

1.1 Using the Admin Interface

This page gives a brief overview of the features provided by Vittles' administrative interface. You can access it by visiting http://yoursite/admin and logging in with the admin account you set up when you ran `manage.py syncdb`.

The *Auth* and *Sites* sections can safely be ignored; Vittles does not currently do anything with those. The other sections are the major application areas of Vittles:

Core Stores a list of foods, ways of preparing food, categorizations for food, units of measurement for food, and how to convert between units of measurement

Cookbook Keeps track of recipes and the ingredients used in them, as well as recipe and ingredient categories and portion sizes

Nutrition Nutritional information for each food

In the administration site, you can create, edit, and delete anything from the above apps. Most of the features should be self-explanatory.

1.2 Core

Vittles provides several models in the **Core** app for working with the basic elements of creating recipes. These include:

Foods Edible stuff that you cook with

Categories Used for grouping foods

Units Ways of measuring food by volume, weight, or discrete quantities, such as cups, liters, ounces or grams

Equivalences Conversions between units

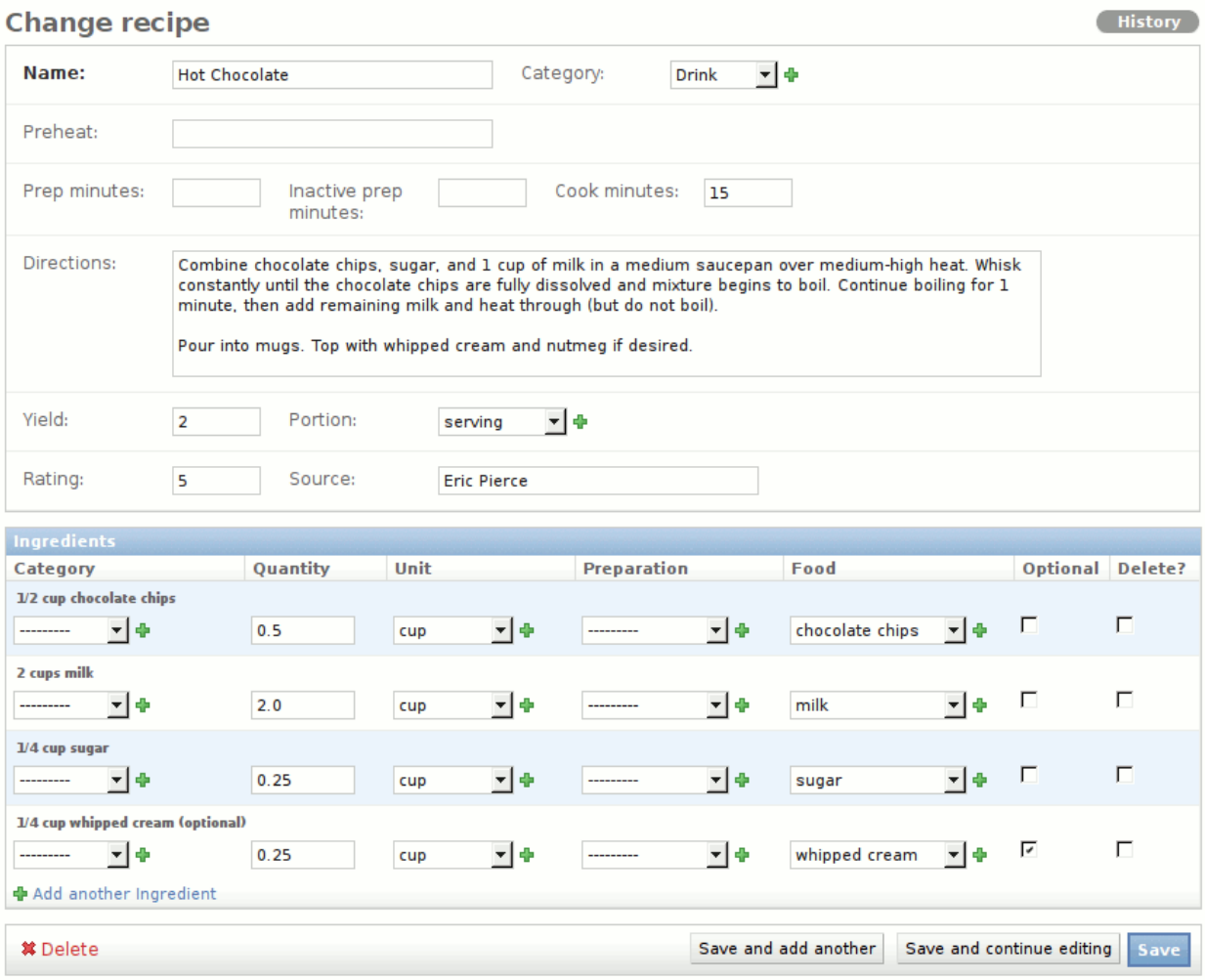
Preparations Ways of preparing food such as chopping, slicing, or mashing

The initial database load populates these tables with some basic starter data, but you may find that you need to add more as you create new recipes.

1.3 Cookbook

A big part of what Vittles does is manage recipes, and the **Cookbook** app is what handles this.

What you're likely to work with most often is **Recipes**. Each recipe has a name, and includes a list of ingredients, some paragraphs of instruction, and several optional fields such as category, rating, preparation time, and yield.



The only required field is **Name**; all others are optional. Some dropdown fields include a green **+** button, which you can use to add new values if the one you want isn't present in the dropdown already.

1.3.1 Prep Time

These optional fields indicate how long your recipe takes to prepare and cook; you can use them in whatever way makes sense to you, but the intended usage is something like this:

Prep minutes Time spent gathering, preparing, and mixing the ingredients

Inactive prep minutes Time spent waiting, for example when meat is marinating, or bread dough is rising

Cook minutes Time spent actually cooking or baking

Again, you can leave these blank if you don't feel like estimating them, but many published recipes already include this information, so it's easy to enter if you're so inclined.

1.3.2 Servings

Each recipe is presumed to make some number of servings, but it's up to you how you would like to define your servings. Obviously, some people eat bigger servings than others, so you may want to specify what exactly you mean

by “Makes 4 servings”.

For example, a muffin recipe may make 12 muffins. If that’s the case, you can enter a **Yield** of “12”, and a **Portion** of “muffin”. When you’re making pancakes, you can specify how many pancakes the recipe makes, and how big the pancakes are. If you like big pancakes, your recipe may have Yield = “4”, Portion = “6-inch pancake”, but if you’re going for silver-dollar-size, you might enter Yield = “24”, Portion = “2-inch pancake”.

Note that the **Portion** field is singular, and will be automatically pluralized when the context calls for it. So don’t use a portion of “2-inch pancakes”, or else your servings will be Gollumized into “24 2-inch pancakeses.”

1.3.3 Ingredients

When creating a new recipe, you’ll have several empty **Ingredient** rows; the most effective way to navigate these is with the keyboard, since many of the fields are dropdowns. When you’re in a dropdown, as soon as you start typing a few letters, the first match will be selected. You can then use the arrow keys to select the value you’re looking for. Many times, it only takes two or three keystrokes to enter the desired value.

The optional **Category** field for each ingredient allows you to divide up the ingredients into logical groups. For example, muffins call for combining the *wet* ingredients and *dry* ingredients separately. You could also use it to separate the *cake* ingredients from the *icing* ingredients. Add new categories if the existing ones don’t suit your needs. For simple recipes, you can omit this field entirely.

The **Quantity** field accepts both decimals and fractions. For example, if your recipe calls for two and one-half cups of flour, you could enter “2.5” or “2-1/2”, then tab over to the **Unit** field and type “cu” to get cups. If the ingredient has no unit (for example, 2 eggs), just leave the “Unit” field empty.

Preparation is where you can specify how the ingredient is to be prepared. Flour may need to be sifted, eggs may need to be beaten, etc. If no preparation is needed, you can leave this unfilled also.

Next, and most importantly, the **Food** field is where you choose the food for this ingredient. It’s at the end to allow a more natural reading across the fields; maybe if Jean-Luc Picard had designed it, the food may have come first, so he could say “Tea, Earl Grey, hot.”

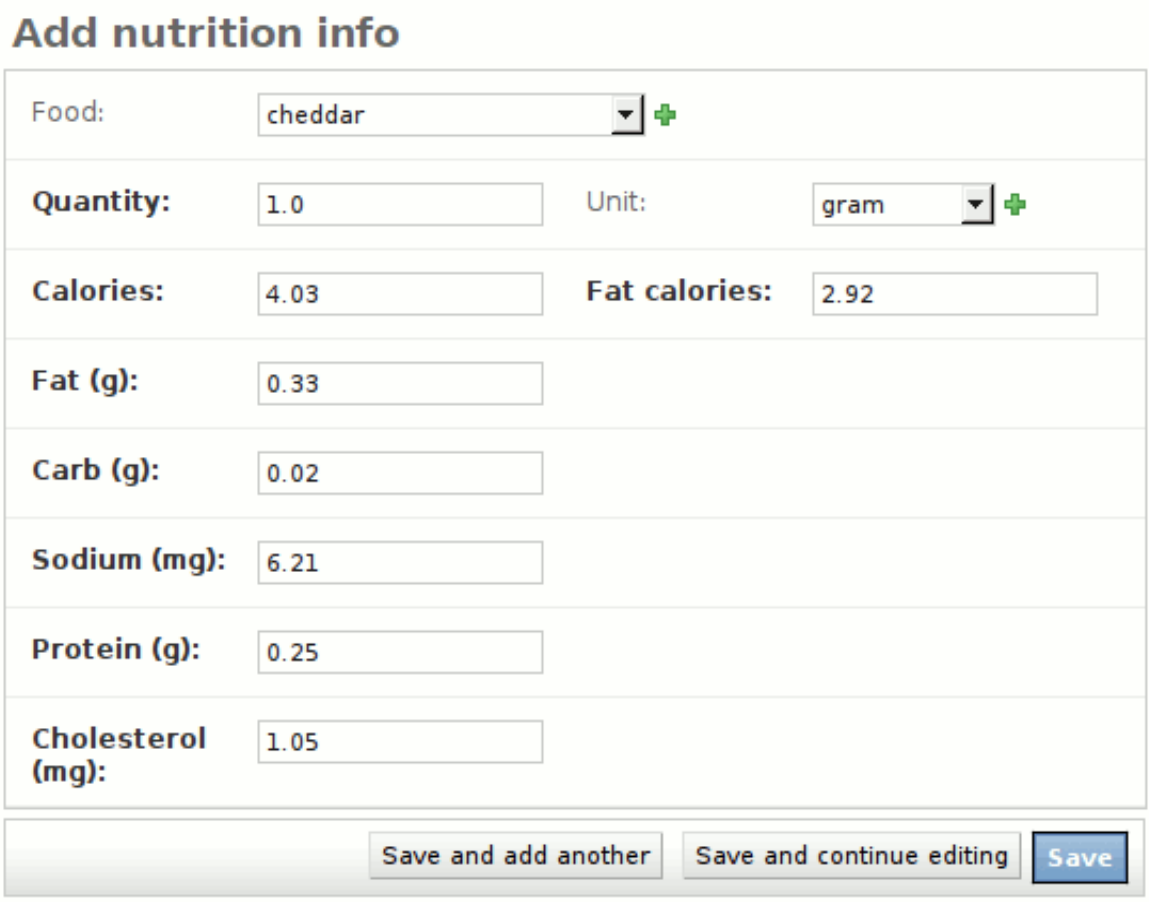
At the end of each food row is an **Optional** checkbox, for ingredients that can be safely omitted from the recipe with no ill effects. When you’re editing an existing recipe, you’ll also see a **Delete** checkbox here, which you can use (in conjunction with the red “Delete” link in the lower-left corner) to remove ingredients.

Finally, click the **Save** button when you’re done editing.

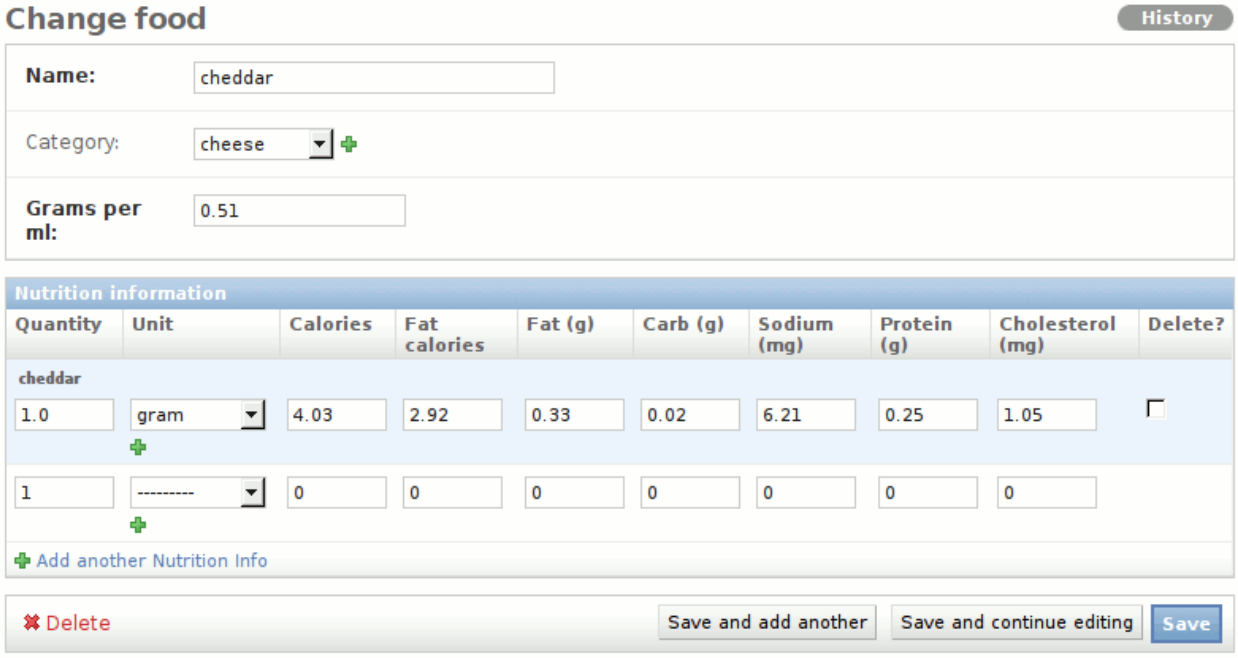
1.4 Nutrition

Each food may have its nutritional value specified in multiple ways; for example, you may want to list the nutritional information for a single large potato, or for a specific weight or volume of potatoes. This nutritional information is managed by the **Nutrition** app.

You can click the **Add** link to add nutritional information for any food, or **Change** to view the list of existing nutritional information. There are two ways to add new nutritional information; you can do it through via the **Nutrition** app, where you’ll get a form like this:



An easier way is to add nutritional information directly to the **Food** form:



Using a standard nutrition label from the food you're entering, simply enter the numeric values for each nutrient. It's best to use a quantity with a **Unit** of **gram** whenever possible, though you may want to omit the unit for certain foods

if they are not normally measured by weight or volume. For example, the nutritional info for one egg could have a **Quantity** of 1, with the **Unit** omitted.

1.4.1 Density

The most accurate way to measure nutritional information is by weight, and Vittles uses grams as a basline for all calculations related to nutritional information. But it's quite common for recipes to specify volumetric measurements–1 cup of flour, 2 tablespoons of honey, etc.–and you can't convert volume to weight unless you know the **density** of whatever you're measuring.

Water has a density of 1.0 grams per milliliter, but most foods have a higher or lower density. For example, all-purpose flour weighs about 0.42 g/ml, while honey weighs about 1.44 g/ml. If you have 1 cup of something, Vittles doesn't know what it weighs unless it knows what's in the cup. If it's flour, it weighs 100 grams, but if it's honey, it weighs 340 grams. If it's material from the core of a white dwarf star, it weighs 230 metric tons.

For this reason, correct calculation of nutritional value is very dependent on having correct densities for your foods. The default value of 1.0 g/ml may be a reasonable approximation in most cases, but if you notice that your recipes containing "1 cup chopped nuts" have thousands of extra calories, better check the density of your nuts.

CHAPTER

TWO

API

This is the Vittles API documentation, autogenerated from the source code.

2.1 core

2.1.1 core.admin

2.1.2 core.models

2.1.3 core.utils

core.utils.**float_to_fraction**(*quantity*, *denominator=16*)

Convert the given quantity into a fraction string, rounded to the nearest `1 / denominator` increment. For example, if `denominator` is 16, round the result to the nearest 1/16th.

If `quantity` is less than 1.0, a simple fraction is returned:

```
>>> float_to_fraction(0.5)
u"1/2"

>>> float_to_fraction(0.33)
u"1/3"

>>> float_to_fraction(0.25)
u"1/4"

>>> float_to_fraction(0.125)
u"1/8"

>>> float_to_fraction(0.1)
u"1/10"
```

If `quantity` is 1.0 or greater, a mixed fraction is returned, with a hyphen separating the integer part from the fractional part.

```
>>> float_to_fraction(1.25)
u"1-1/4"

>>> float_to_fraction(2.75)
u"2-3/4"
```

```
>>> float_to_fraction(9.33)
u"9-1/3"
```

All results are rounded to the nearest `1 / denominator` increment, so you can decide how precise you need the result to be:

```
>>> float_to_fraction(0.1875, 16)
u"3/16"

>>> float_to_fraction(0.1875, 8)
u"1/5"

>>> float_to_fraction(0.1875, 4)
u"1/4"
```

core.utils.**fraction_to_float**(*fraction_string*)

Convert a fraction string into a floating-point value.

Simple fractions take the form of "n/d":

```
>>> fraction_to_float("1/2")
0.5
>>> fraction_to_float("3/8")
0.375
```

Mixed fractions may be separated by one or more spaces or hyphens.

```
>>> fraction_to_float("1 1/4")
1.25
>>> fraction_to_float("1-1/4")
1.25
>>> fraction_to_float("1 - 1/4")
1.25
```

Any string that is already a decimal expression is just converted to its numeric form:

```
>>> fraction_to_float("5.75")
5.75
>>> fraction_to_float(".5")
0.5
```

core.utils.**pluralize**(*noun*)

Pluralize the given noun, using a simple heuristic. Will pluralize some nouns incorrectly because English is beastly complicated.

Consonant + 'o' gets '-es':

```
>>> pluralize('potato')
'potatoes'
```

Sibilants get '-es':

```
>>> pluralize('batch')
'batches'
>>> pluralize('box')
'boxes'
```

Consonant + 'y' becomes '-ies':

```
>>> pluralize('cherry')
'cherries'
```

Endings of 'f' or 'fe' become '-ves':

```
>>> pluralize('loaf')
'loaves'
>>> pluralize('bay leaf')
'bay leaves'
>>> pluralize('knife')
'knives'
```

Simple 's' ending:

```
>>> pluralize('ounce')
'ounces'
>>> pluralize('egg')
'eggs'
```

core.utils.**format_food_unit**(*quantity*, *unit*, *food*)

Return a unicode string describing the given quantity of food. `quantity` may be an actual number (int or float), or a string containing a decimal or fraction as understood by `fraction_to_float()`.

If a unit is given, the unit is pluralized when appropriate:

```
>>> format_food_unit(2, 'cup', 'flour')
u"2 cups flour"
>>> format_food_unit(1.5, 'teaspoon', 'salt')
u"1-1/2 teaspoons salt"
>>> format_food_unit('1-3/4', 'ounce', 'butter')
u"1-3/4 ounces butter"
```

If no unit is given, the food is pluralized:

```
>>> format_food_unit(3, None, 'egg')
u"3 eggs"
>>> format_food_unit(2, None, 'potato')
u"2 potatoes"
>>> format_food_unit('4-1/2', None, 'bell pepper')
u"4-1/2 bell peppers"
```

In all cases, if the quantity is <= 1, no pluralization is done:

```
>>> format_food_unit(1, None, 'egg')
u"1 egg"
>>> format_food_unit(0.75, 'cup', 'flour')
u"3/4 cup flour"
>>> format_food_unit(0.5, 'cup', 'oil')
u"1/2 cup oil"
>>> format_food_unit('1/4', 'teaspoon', 'baking powder')
u"1/4 teaspoon baking powder"
```

2.1.4 core.helpers

core.helpers.**convert_unit**(*unit*, *to_unit*)

Convert `unit` to the equivalent quantity `to_unit`. Requires that an `Equivalence` be defined for the relevant units; if no `Equivalence` is found, raise a `NoEquivalence` exception.

core.helpers.**to_grams**(*unit*, *food=None*)

Return the given unit in terms of grams. If `unit` is a volume, attempt to convert based on the given food's density (g/ml). If `food` is not given, assume a density of 1.0 g/ml.

core.helpers.**to_ml**(*unit*, *food=None*)
Return the given unit in terms of milliliters. If `unit` is a weight, attempt to convert based on the given `food`'s density (g/ml). If `food` is not given, assume a density of 1.0 g/ml.

core.helpers.**convert_amount**(*quantity*, *unit*, *to_unit*)
Convert a quantity in a given unit to the equivalent quantity in another unit. Requires that an `Equivalence` be defined for the relevant units; if no `Equivalence` is found, raise a `NoEquivalence` exception.

core.helpers.**add_amount**(*quantity*, *unit*, *to_quantity*, *to_unit*)
Add two amounts together (possibly using different units), and return the resulting quantity in terms of the first unit.

core.helpers.**subtract_amount**(*quantity*, *unit*, *to_quantity*, *to_unit*)
Subtract one amount from another (possibly using different units), and return the resulting quantity in terms of the first unit.

core.helpers.**group_by_category**(*queryset*)
Group objects in `queryset` by category name, and return a list of `(category_name, [matching_objects])` for each category.

Objects in `queryset` must have a `category` attribute, where `category` itself has a `name` attribute.

2.2 cookbook

2.2.1 cookbook.admin

2.2.2 cookbook.models

2.3 nutrition

2.3.1 nutrition.admin

2.3.2 nutrition.models

class nutrition.models.**NutritionInfo**(**args*, ***kwargs*)
Generic nutritional information.

full_string()
Return a string with full details of this NutritionInfo.

is_empty()
Return True if this NutritionInfo is empty.

is_equal(*other*)
Return True if this `NutritionInfo` is equal to another, False otherwise.

set_equal(*other*)
Set this `NutritionInfo` equal to another, and save.

2.4 inventory

2.4.1 inventory.admin

2.4.2 inventory.models

CHAPTER

THREE

SETUP

You'll probably want to set up a virtual environment for Vittles using virtualenv or virtualenvwrapper. Refer to those projects' docs for instructions.

Once you have a virtual environment activated, install Vittles' dependencies:

```
$ cd /path/to/vittles
$ pip install -r reqs.txt
```

Edit `settings.py` and define where you want the database to live:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME: '/some/path/vittles.sqlite3',
    }
}
```

Save, then sync:

```
$ ./manage.py syncdb
```

This will load some basic data from `core/fixtures/initial_data.yaml` to get you started, including a bunch of foods, units, and equivalences.

Start the development webserver:

```
$ ./manage.py runserver
```

Then visit http://127.0.0.1:8000/admin to start entering recipes, and http://127.0.0.1:8000/cookbook to view the recipes you've entered.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

c

n