- Recipe Model:

```
from django.db import models
from recipe_ingredients.models import RecipeIngredient
from ingredients.models import Ingredient
from django.shortcuts import reverse


TYPE_OF_RECIPE= (
    ('breakfast', 'Breakfast'),
    ('lunch', 'Lunch'),
    ('dinner', 'Dinner'),
)


def process_recipe_ingredients(ingredients_string):
    # Split the string into a list based on commas and spaces
    ingredients_list = [ingredient.strip() for ingredient in
ingredients_string.split(',')]

    # Loop through the list and add each ingredient to the master
ingredient list
    for ingredient_name in ingredients_list:
        # Check if the ingredient already exists in the database
        ingredient, created =
Ingredient.objects.get_or_create(name=ingredient_name)

    print("Ingredients List:", ingredients_list)
    # Return the processed list of ingredients
    return ingredients_list


class Recipe(models.Model):
    name = models.CharField(max_length=120)
```

```
    cooking_time = models.PositiveIntegerField(help_text="In minutes")
    difficulty = models.CharField(max_length=50)
    min_serving_size = models.PositiveIntegerField(help_text="Enter the minimum number of people this would serve.")
    max_serving_size = models.PositiveIntegerField(help_text="Enter the maximum number of people this would serve.")
    type_of_recipe = models.CharField(max_length=30, choices=TYPE_OF_RECIPE)
    ingredients = models.TextField("ingredients.Ingredient", blank=True, help_text="Enter the ingredients for the recipe, separated by commas.", default="")
    directions = models.TextField(help_text="Enter the directions for preparing the recipe.")

    pic = models.ImageField(upload_to="recipes", default="no_picture.jpg")

    def calculate_difficulty(self):
        ingredients_list = [ingredient.strip() for ingredient in self.ingredients.split(',')]
        num_ingredients = len(ingredients_list)

        print("Cooking Time:", self.cooking_time)
        print("Number of Ingredients:", num_ingredients)

        if self.cooking_time < 10 and num_ingredients < 4:
            print("Easy")
            return "Easy"
        elif self.cooking_time < 10 and num_ingredients >= 4:
            print("Medium")
            return "Medium"
        elif self.cooking_time >= 10 and num_ingredients < 4:
            print("Intermediate")
            return "Intermediate"
        else:
            print("Hard")
```

```
            return "Hard"
```

```
    def save(self, *args, **kwargs):
        self.difficulty = self.calculate_difficulty()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.name

    def add_to_master_list(self):
        # Split the ingredients string by commas and strip spaces
        ingredient_list = [ingredient.strip() for ingredient in
self.ingredients.split(',')]

        # Remove duplicates from the ingredient list
        unique_ingredients = list(set(ingredient_list))

        # Get the current master ingredient list (you need to define it
somewhere)
        current_master_list = get_master_ingredient_list()

        # Add only the unique ingredients not already in the master list
        new_ingredients = [ingredient for ingredient in unique_ingredients
if ingredient not in current_master_list]
        current_master_list.extend(new_ingredients)

        # Update the master ingredient list (you need to define the
method)
        update_master_ingredient_list(current_master_list)
```

```
        # Save the changes
        self.save()


    def get_absolute_url(self):
        return reverse("recipes:recipe_detail", kwargs={"pk": self.pk})
```

- Changes:
  - I moved cooking time above ingredients because that seemed like it would make more sense to the user to be laid out that way—amount of time is a quick glance display item vs the blob of text the directions and ingredients would be after—semed more user-friendly
  - Added “help_text="In minutes"” param to the cooking time
  - Added pic
  - Added Ingredient model—I saw it in one of the examples, but I am now wondering if I need it... It now feels very unnecessary to me... I might delete it when working on 2.6.
  - I added the app “recipe_ingredients” as an intermediary to manage the many-to-many relationship—the example was set up this way, but I don’t think this is necessary in my application, so I might delete—would appreciate feedback on this. I am getting a little confused...

        - Added appropriate help text with all text inputs in the recipe model
        - I included the calculate difficulty function in the recipe model
- Users Model

```
from recipes.models import Recipe




class User(models.Model):
    username = models.CharField(max_length=120)
    saved_recipes = models.ManyToManyField(Recipe,
related_name='saved_by_users', blank=True)

    def __str__(self):
        return self.username
```

    - Changes:
        - I think we learn this in 2.6, so I am not sure if this looks right…
- Recipe_ingredients:
    - I got this idea from an example, but I'm not positive if I'm understanding/executing it right. I made this app to manage the relationship between the user model and the recipe model. To my understanding, it takes ingredients submitted by the user via the create_recipe template, and adds those ingredients to the ingredients list in the ingredients app. Please provide feedback if I am understanding that correctly, and thus, have it set up correctly.

```python
from django.db import models
from ingredients.models import Ingredient

# Create your models here.

class RecipeIngredient(models.Model):
    recipe = models.ForeignKey(
        "recipes.Recipe", on_delete=models.CASCADE,
related_name="recipe_ingredients"
    )
    ingredient = models.ForeignKey("ingredients.Ingredient",
on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.ingredient} - {self.recipe}"
```

- Ingredients:

```python
from django.db import models

# Create your models here.


class Ingredient(models.Model):
    name = models.CharField(max_length=225)

    def __str__(self):
        return str(self.name)
```