# Lab: Advanced Django Model Techniques

This document defines the problems for the in-class lab for the [Python ORM course @ Software University](https://softuni.bg/trainings/4253/python-orm-october-2023).

Submit your solutions in the SoftUni [Judge system](https://judge.softuni.org/Contests/4331/Advanced-Django-Model-Techniques-Lab).

For this lab, you are given an empty **ORM project skeleton** (you can download it from [here](https://softuni.bg/downloads/svn/python-db/trunk/Sep-2023/Python-ORM/08-Advanced-Django-Model-Techniques/08-lab-orm-skeleton.zip)). Your task for today is to **create a Restaurant Review System** with **restaurants**, **menus**, **customers**, and **reviews**.

## Restaurant

In the **main\_app** create a model called **"Restaurant"** with the following fields:

* **name**
  + A **character** field.
  + Validations:
    - **Minimum length** of **2** characters with a **custom error message** **"Name must be at least 2 characters long."**
    - **Maximum length** of **100** characters with a **custom error message** **"Name cannot exceed 100 characters."**
  + Represents the name of the restaurant.
* **location**
  + A **character** field.
  + Validations:
    - **Minimum length** of **2** characters with a **custom error message** **"Location must be at least 2 characters long."**
    - **Maximum length** of **200** characters with a **custom error message** **"Location** **cannot exceed 200 characters."**
  + Represents the location of the restaurant.
* **description**
  + A **text** field.
  + **Optional** field.
  + Represents the description of the restaurant.
* **rating**
  + A **decimal** field.
  + It stores a maximum of **3 digits** and **2 decimal places**.
  + Validations:
    - Has a **minimum value** of **0** with a **custom error message** **"Rating must be at least 0.00."**
    - Has a **maximum value** of **5** with a **custom error message** **"Rating cannot exceed 5.00."**
  + Represents the rating of the restaurant.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Restaurant  from django.core.exceptions import ValidationError  valid\_restaurant = Restaurant(  name="Delicious Bistro",  location="123 Main Street",  description="A cozy restaurant with a variety of dishes.",  rating=5.00,  )  try:  valid\_restaurant.full\_clean()  valid\_restaurant.save()  print("Valid Restaurant saved successfully!")  except ValidationError as e:  print(f"Validation Error: {e}")  invalid\_restaurant = Restaurant(  name="A",  location="A" \* 201,  description="A restaurant with a long name and invalid rating.",  rating=5.01,  )  try:  invalid\_restaurant.full\_clean()  invalid\_restaurant.save()  print("Invalid Restaurant saved successfully!")  except Exception as e:  print(f"Validation Error: {e}") |
| **Output** |
| Valid Restaurant saved successfully!  Validation Error: {'name': ['Name must be at least 2 characters long.'], 'location': ['Location cannot exceed 200 characters.'], 'rating': ['Rating cannot exceed 5.00.']} |

## Menu

In the **main\_app** create an **additional** **model** called **"Menu"** with the following fields:

* **name**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the name of the menu.
* **description**
  + A **text** field.
  + A custom validator called **"validate\_menu\_categories"** that checks if the description field includes **each of the categories:** **"Appetizers"**, **"Main** **Course"**, and **"Desserts"**. If any of these categories is **missing**, a **ValidationError** should be raised with the message **"The menu must include each of the categories "Appetizers", "Main Course", "Desserts"."**
  + Represents the content of the menu.
* **restaurant**
  + A **foreign key** field.
  + Establishes a **many-to-one relationship** with the **"Restaurant" model**, associating each menu with a restaurant.
  + When a **restaurant is deleted**,all **menus** that reference the deleted restaurantshould be **deleted too**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Restaurant, Menu  *# keep the data from the previous exercise, so you can reuse it*  valid\_menu = Menu(  name="Menu at The Delicious Bistro",  description="\*\* Appetizers: \*\*\nSpinach and Artichoke Dip\n\*\* Main Course: \*\*\nGrilled Salmon\n\*\* Desserts: \*\*\nChocolate Fondue",  restaurant=Restaurant.objects.first(),  )  try:  valid\_menu.full\_clean()  valid\_menu.save()  print("Valid Menu saved successfully!")  except ValidationError as e:  print(f"Validation Error: {e}")  invalid\_menu = Menu(  name="Incomplete Menu",  description="\*\* Appetizers: \*\*\nSpinach and Artichoke Dip",  restaurant=Restaurant.objects.first(),  )  try:  invalid\_menu.full\_clean()  invalid\_menu.save()  print("Invalid Menu saved successfully!")  except ValidationError as e:  print(f"Validation Error: {e}") |
| **Output** |
| Valid Menu saved successfully!  Validation Error: {'description': ['The menu must include each of the categories "Appetizers", "Main Course", "Desserts".']} |

## Restaurant Review

In the **main\_app** create an **additional** **model** called **"RestaurantReview"** with the following fields:

* **reviewer\_name**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the name of the author of the review.
* **restaurant**
  + A **foreign key** field.
  + Establishes a **many-to-one relationship** with the **"Restaurant" model**, associating each review with a restaurant.
  + When a **restaurant is deleted**,all **reviews** that reference the deleted restaurantshould be **deleted too**.
* **review\_content**
  + A **text** field.
  + Represents the content of the review.
* **rating**
  + A **positive integer** field.
  + Has a **maximum value** of **5**.

For this model, also define:

* A **default ordering** **in descending order** for the model's database queries on the field **"rating"**.
* A **human-readable name** for the model in the admin interface - **"Restaurant Review"**.
* A **human-readable plural name** for the model in the admin interface - **"Restaurant Reviews"**.
* That **NO two records** for the fields **"reviewer\_name"** and **"restaurant"** should **share the same combination of values**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Restaurant, RestaurantReview  from django.core.exceptions import ValidationError  restaurant1 = Restaurant.objects.create(name="Restaurant A", location="123 Main St.", description="A cozy restaurant", rating=4.88)  restaurant2 = Restaurant.objects.create(name="Restaurant B", location="456 Elm St.", description="Charming restaurant", rating=3.59)  RestaurantReview.objects.create(reviewer\_name="Bob", restaurant=restaurant1, review\_content="Good experience overall.", rating=4)  RestaurantReview.objects.create(reviewer\_name="Aleks", restaurant=restaurant1, review\_content="Great food and service!", rating=5)  RestaurantReview.objects.create(reviewer\_name="Charlie", restaurant=restaurant2, review\_content="It was ok!", rating=2)  duplicate\_review = RestaurantReview(reviewer\_name="Aleks", restaurant=restaurant1, review\_content="Another great meal!", rating=5)  try:  duplicate\_review.full\_clean()  duplicate\_review.save()  except ValidationError as e:  print(f"Validation Error: {e}")  print("All Restaurant Reviews:")  for review in RestaurantReview.objects.all():  print(f"Reviewer: {review.reviewer\_name}, Rating: {review.rating}, Restaurant: {review.restaurant.name}") |
| **Output** |
| Validation Error: {'\_\_all\_\_': ['Restaurant Review with this Reviewer name and Restaurant already exists.']}  All Restaurant Reviews:  Reviewer: Aleks, Rating: 5, Restaurant: Restaurant A  Reviewer: Bob, Rating: 4, Restaurant: Restaurant A  Reviewer: Charlie, Rating: 2, Restaurant: Restaurant B |

## Restaurant Review Types

We decided to differentiate **two types of restaurant reviews** - regular reviews and food critic reviews. In this case, we do not want the base **"RestaurantReview"** model to have a database table and save data on its own. Also, in the **main\_app** create **two** **additional** **models** called **"RegularRestaurantReview"** and **"FoodCriticRestaurantReview"**.

The **"RegularRestaurantReview"** inherits all fields and options from the base model. **No additional fields or options** are **defined** in this subclass.

The **"FoodCriticRestaurantReview"** includes an **additional field**:

* **food\_critic\_cuisine\_area**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the cuisine area/specialization of the food critic.

For this model, also define:

* A **default ordering in descending order** for the model's database queries on the field **"rating"**.
* A **human-readable name** for the model in the admin interface - **"Food Critic Review"**.
* A **human-readable plural name** for the model in the admin interface - **"Food Critics Reviews"**.
* That **NO two records** for the fields **"reviewer\_name"** and **"restaurant"** should **share the same combination of values**.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import Restaurant, RegularRestaurantReview, FoodCriticRestaurantReview  restaurant1 = Restaurant.objects.create(name="Restaurant A", location="123 Main St.", description="A cozy restaurant", rating=4.88)  RegularRestaurantReview.objects.create(reviewer\_name="Bob", restaurant=restaurant1, review\_content="Good experience overall.", rating=4)  RegularRestaurantReview.objects.create(reviewer\_name="Aleks", restaurant=restaurant1, review\_content="Great food and service!", rating=5)  duplicate\_review = RegularRestaurantReview(reviewer\_name="Aleks", restaurant=restaurant1, review\_content="Another great meal!", rating=5)  try:  duplicate\_review.full\_clean()  duplicate\_review.save()  except ValidationError as e:  print(f"Validation Error: {e}")  print("Regular Restaurant Review:")  print(f"Model Name: {RegularRestaurantReview.\_meta.verbose\_name}")  print(f"Model Plural Name: {RegularRestaurantReview.\_meta.verbose\_name\_plural}")  print("Food Critic Restaurant Review:")  print(f"Model Name: {FoodCriticRestaurantReview.\_meta.verbose\_name}")  print(f"Model Plural Name: {FoodCriticRestaurantReview.\_meta.verbose\_name\_plural}") |
| **Output** |
| Validation Error: {'\_\_all\_\_': ['Restaurant Review with this Reviewer name and Restaurant already exists.']}  Regular Restaurant Review:  Model Name: Restaurant Review  Model Plural Name: Restaurant Reviews  Food Critic Restaurant Review:  Model Name: Food Critic Review  Model Plural Name: Food Critic Reviews |

## Menu Review

In the **main\_app** create an **additional** **model** called **"MenuReview"** with the following fields:

* **reviewer\_name**
  + A **character** field.
  + It has a **maximum length** of **100** characters.
  + Represents the name of the author of the review.
* **menu**
  + A **foreign key** field.
  + Establishes a **many-to-one relationship** with the **"Menu" model**, associating each review with a menu.
  + When a **menu is deleted**,all **reviews** that reference the deleted menushould be **deleted too**.
* **review\_content**
  + A **text** field.
  + Represents the content of the review.
* **rating**
  + A **positive integer** field.
  + Has a **maximum value** of **5**.

For this model, also define:

* A **default ordering** **in descending order** for the model's database queries on the field **"rating"**.
* A **human-readable name** for the model in the admin interface - **"Menu Review"**.
* A **human-readable plural name** for the model in the admin interface - **"Menu Reviews"**.
* That **NO two records** for the fields **"reviewer\_name"** and **"menu"** should **share the same combination of values**.
* An option to **improve database query performance** when filtering or sorting by **"menu"**. The option must be set to have a custom index name called **"main\_app\_menu\_review\_menu\_id"**

## Rating and Review Content

As you can see the rating and the review content are part of each type of review. Your task is to **create a** **reusable component** that can be **mixed** into the models **"RestaurantReview"** and **"MenuReview"** to add the review-related **fields** **"rating"** and **"review\_content"** and their **validation rules**. Do not forget to **add the common meta options** of the models related to the review parts.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from main\_app.models import RegularRestaurantReview, Restaurant, MenuReview, Menu  Restaurant.objects.create(name="Savory Delight", location="456 Elm Avenue", rating=4.2,)  restaurant\_from\_db = Restaurant.objects.get(name="Savory Delight")  RegularRestaurantReview.objects.create(reviewer\_name="Alice", restaurant=restaurant\_from\_db, rating=4, review\_content="Good experience overall.")  review\_from\_db = RegularRestaurantReview.objects.get(reviewer\_name="Alice", restaurant=restaurant\_from\_db)  print(  f"Reviewer name: {review\_from\_db.reviewer\_name}\n"  f"Restaurant: {review\_from\_db.restaurant.name}\n"  f"Rating: {review\_from\_db.rating}\n"  f"Review content: {review\_from\_db.review\_content}"  )  Menu.objects.create(name="Delightful Food Menu", description="Appetizers:\nSpinach and Artichoke Dip\nMain Course:\nGrilled Salmon\nDesserts:\nChocolate Fondue", restaurant=restaurant\_from\_db)  menu\_from\_db = Menu.objects.get(name="Delightful Food Menu")  MenuReview.objects.create(reviewer\_name="Lilly", menu=menu\_from\_db, rating=5, review\_content="Delicious food")  menu\_review\_from\_db = MenuReview.objects.get(reviewer\_name="Lilly", menu=menu\_from\_db)  print(  f"Reviewer name: {menu\_review\_from\_db.reviewer\_name}\n"  f"Menu: {menu\_review\_from\_db.menu.name}\n"  f"Rating: {menu\_review\_from\_db.rating}\n"  f"Review content: {menu\_review\_from\_db.review\_content}"  ) |
| **Output** |
| Reviewer name: Alice  Restaurant: Savory Delight  Rating: 4  Review content: Good experience overall.  Reviewer name: Lilly  Menu: Delightful Food Menu  Rating: 5  Review content: Delicious food |