# Exercises: Advanced Django Model Techniques

This document defines the **exercise assignments** for the [**Python ORM course @ Software University**](https://softuni.bg/modules/137/python-db).

Submit your solutions in the SoftUni [**Judge system**](https://judge.softuni.org/Contests/4332/Advanced-Django-Model-Techniques-Exercise).

## Customer

Write a Django model called "**Customer"** with the provided **fields**:

* **name** - character field, **consisting of a maximum of 100 characters**. The name must contain only **letters** and **spaces**, otherwise **raise** a **ValidationError** with the message: "**Name can only contain letters and spaces"**.
* **age** - positive integer field. If the **age** is under 18, **raise** a **ValidationError** with the message: "**Age must be greater than or equal to 18"**.
* **email** - email field. If the email is **invalid**, raise a **ValidationError** with the message: "**Enter a valid email address"**.
* **phone\_number** - character field, **consisting of a maximum of 13 characters**. The phone number must start with **"+359"** followed by 9 more digits, otherwise raise a **ValidationError** with the message: **"Phone number must start with '+359' followed by 9 digits"**.
* **website\_url** - URL field. If the URL is **invalid**, raise a **ValidationError** with the message "**Enter a valid URL**".

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| customer = Customer(  name="Svetlin Nakov1",  age=1,  email="nakov@example",  phone\_number="+35912345678",  website\_url="htsatps://nakov.com/" )  try:  customer.full\_clean()  customer.save()  except ValidationError as e:  print('\n'.join(e.messages)) |
| **Output** |
| Name can only contain letters and spaces  Age must be greater than or equal to 18  Enter a valid email address  Phone number must start with '+359' followed by 9 digits  Enter a valid URL |

## Media

You'll build a **media** **management** **system** that handles various **media** types, including **books**, **movies**, and **music**.

### Model BaseMedia

It is a base model and **is NOT meant to create a database table on its own**. The model has the following fields:

* **title** - character field, **consisting of a maximum of 100 characters.**
* **description** - text field.
* **genre** - character field, **consisting of a maximum of 50 characters.**
* **created\_at** - date time field. Every time a **new** **record** is created it should save the **current time** of the creation of the **record**.

### Model Book

It is a model of type **media**. The model has the following **fields**:

* **author** - character field, **consisting of a maximum of 100 characters.** The title must have at least 5 characters, otherwise, **raise** a **ValidationError** with the message: "**Author must be at least 5 characters long**".
* **isbn** - character field, **consisting of a maximum of 20 characters, unique.** The isbn must have at least 6 characters, otherwise, **raise** a **ValidationError** with the message: "**ISBN must be at least 6 characters long**".

### Model Movie

It is a model of type **media**. The model has the following **fields**:

* **director** - character field, **consisting of a maximum of 100 characters.** The director must have at least 8 characters, otherwise, **raise** a **ValidationError** with the message: "**Director must be at least**

**8 characters long**".

### Model Music

It is a model of type **media**. The model has the following **fields**:

* **artist** - character field, **consisting of a maximum of 100 characters.** The artist must have at least 9 characters, otherwise, **raise** a **ValidationError** with the message: "**Artist must be at least 9 characters long**".

### BaseMedia Meta class

* **Order** the fields by **created\_at** (**descending**) and **title** (**ascending**).

### Book Meta class

* The "**Meta**" class should **inherit** its parent class fields**.** Also, set the **verbose name** to "**Model Book**" and the **plural verbose name** to "**Models of type - Book**".

### Movie Meta class

* The "**Meta**" class should **inherit** its parent class fields**.** Also, set the **verbose name** to "**Model Movie**" and the **plural verbose name** to "**Models of type - Movie**".

### Music Meta class

* The "**Meta**" class should **inherit** its parent class fields**.** Also, set the **verbose name** to "**Model Music**" and the **plural verbose name** to "**Models of type - Music** ".

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| book = Book(  title="Short Title",  description="A book with a short title.",  genre="Fiction",  author="A",  isbn="1234" )  try:  book.full\_clean()  book.save()  except ValidationError as e:  print("Validation Error for Book:")  for field, errors in e.message\_dict.items():  print(f"{field}: {', '.join(errors)}") |
| **Output** |
| Validation Error for Book:  author: Author must be at least 5 characters long  isbn: ISBN must be at least 6 characters long |

## Tax-Inclusive Pricing

Write a Django model called "**Product"**. The model has the following **fields**:

* **name** - character field, **consisting of a maximum of 100 characters.**
* **price** - decimal field, **with maximum of 10 digits and 2 decimal places.**

### Model DiscountedProduct

It is a **type** of **product**. Stores information for **discounted** products.

### DiscountedProduct Meta class

* The model "**DiscountedProduct**" shares the same **database** **table** as its parent model "**Product**" and provides **additional** or **customized** functionality.

### Methods inside the Product model

**calculate\_tax()** **returns** the **tax** for the product. The **tax rate** is **8%** of the **price**.

**calculate\_shipping\_cost(weight: Decimal)** **returns** the calculated **shipping cost** for the product. The **shipping cost** is the **weight units** of the product **multiplied** by **2.00**.

**format\_product\_name()** **returns** the **name** of the product in the **format**:

* **"Product: {product\_name}"**

### Methods inside the DiscountedProduct model

**calculate\_price\_without\_discount()** **returns** the calculated **price without discount** for the product. The **original** **price** is **20%** higher than the **price** **without** a **discount**.

**calculate\_tax()** **returns** the **tax** for the product. The **tax rate** is **5%** of the **price**.

**calculate\_shipping\_cost(weight: Decimal)** **returns** the calculated **shipping cost** for the product. The **shipping cost** is the **weight units** of the product **multiplied** by **1.50**.

**format\_product\_name()** **returns** the **name** of the product in the **format**:

* **"Discounted Product: {product\_name}"**

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| # Create a Product instance product = Product.objects.create(name="Gaming Keyboard", price=Decimal(100.00))  # Calculate and print the tax tax\_price = product.calculate\_tax() print(f"Tax for {product.name}: ${tax\_price:.2f}")  # Calculate and print the shipping cost shipping\_cost = product.calculate\_shipping\_cost(Decimal(2.50)) print(f"Shipping Cost for {product.name}: ${shipping\_cost:.2f}")  # Format and print the product name formatted\_name = product.format\_product\_name() print(f"Formatted Product Name: {formatted\_name}")  # Create a DiscountedProduct instance discounted\_product = DiscountedProduct.objects.create(name="Gaming Mouse", price=Decimal(120.00))  # Calculate and print the price without discount (DiscountedProduct) discounted\_price = discounted\_product.calculate\_price\_without\_discount() print(f"Price Without Discount for {discounted\_product.name}: ${discounted\_price:.2f}")  # Calculate and print the tax (DiscountedProduct) tax\_price = discounted\_product.calculate\_tax() print(f"Tax for {discounted\_product.name}: ${tax\_price:.2f}")  # Calculate and print the shipping cost (DiscountedProduct) shipping\_cost = discounted\_product.calculate\_shipping\_cost(Decimal(2.50)) print(f"Shipping Cost for {discounted\_product.name}: ${shipping\_cost:.2f}")  # Format and print the product name (DiscountedProduct) formatted\_name = discounted\_product.format\_product\_name() print(f"Formatted Product Name: {formatted\_name}") |
| **Output** |
| Tax for Gaming Keyboard: $8.00  Shipping Cost for Gaming Keyboard: $5.00  Formatted Product Name: Product: Gaming Keyboard  Price Without Discount for Gaming Mouse: $144.00  Tax for Gaming Mouse: $6.00  Shipping Cost for Gaming Mouse: $3.75  Formatted Product Name: Discounted Product: Gaming Mouse |

## Superhero Universe

Write a Django model called "**Hero"**. The model has the following **fields**:

* **name** - character field, **consisting of a maximum of 100 characters.**
* **hero\_title** - character field, **consisting of a maximum of 100 characters.**
* **energy** - positive integer field.

**Note: the energy should always remain positive integer!**

### Model SpiderHero

It is a **type** of **hero**. Stores information about **spider** hero.

### Model FlashHero

It is a **type** of **hero**. Stores information about **flash** hero.

### Methods inside the SpiderHero model

**swing\_from\_buildings()** is the secret special ability of the **spider** **hero**. Each time the **ability** is used, the hero's **energy** **decreases** by **80** units.

* Before the **ability** is used and the **energy is less than 80,** only **return** as a result: **"{hero\_name} as Spider Hero is out of web shooter fluid"**.
* Otherwise, use the **ability**, **save** all the changes, and **return** as a result: **"{hero\_name} as Spider Hero swings from buildings using web shooters"**. If the **remaining** **energy** is 0, **save** it as 1 in the **database**.

### Methods inside the FlashHero model

**run\_at\_super\_speed()** is the secret special ability of the **flash** **hero**. Each time the **ability** is used, the hero's **energy** **decreases** by **65** units.

* Before the **ability** is used and the **energy is less than 65,** only **return** as a result: **"{hero\_name} as Flash Hero needs to recharge the speed force"**.
* Otherwise, use the **ability**, **save** all the changes, and **return** as a result: **"{hero\_name} as Flash Hero runs at lightning speed, saving the day"**. If the **remaining** **energy** is 0, **save** it as 1 in the **database**.

### SpiderHero Meta class

* The model "**SpiderHero**" shares the same **database** **table** as its parent model "**Hero**" and provides **additional** or **customized** functionality.

### FlashHero Meta class

* The model "**FlashHero**" shares the same **database** **table** as its parent model "**Hero**" and provides **additional** or **customized** functionality.

### Mixin RechargeEnergyMixin

Write a **mixin** called "**RechargeEnergyMixin**". The **mixin** has the following **method**:

**recharge\_energy(amount: int)** **recharges** the **energy** for the **hero** with the given **amount**. The energy **cannot** exceed **100**.

**Note**: **Inherit** the **mixin** in the "**Hero**" model.

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| *# Create instance of SpiderHero* spiderman = SpiderHero(name="Spider-Man", hero\_title="Spider Hero", energy=100)  *# Create instance of FlashHero* flash = FlashHero(name="The Flash", hero\_title="Flash Hero", energy=70)  *# Save the instances to the database* spiderman.save() flash.save()  *# Run the special abilities* print(spiderman.swing\_from\_buildings()) print(flash.run\_at\_super\_speed()) print(spiderman.swing\_from\_buildings())  *# Recharge the energy of Spider-Man and The Flash using the mixin method* spiderman.recharge\_energy(195) flash.recharge\_energy(40)  *# Now you can check the updated energy levels* print(f"{spiderman.name} - Energy: {spiderman.energy}") print(f"{flash.name} - Energy: {flash.energy}") |
| **Output** |
| Spider-Man as Spider Hero swings from buildings using web shooters  The Flash as Flash Hero runs at lightning speed, saving the day  Spider-Man as Spider Hero is out of web shooter fluid  Spider-Man - Energy: 100  The Flash - Energy: 45 |

## \*Vector Searching

In this **exercise**, you will create a Django **model** called "**Document**" for storing **documents** with the ability to perform **efficient** **full-text searches**. The model has the following fields:

* **title** - character field, **consisting of a maximum of 200 characters.**
* **content** - text field.
* **search\_vector** - "**SearchVectorField"**, with **null="True"**.

### Document Meta class

* The model "**Document**" has an **indexes** option. The only **field** in the option is "**search\_vector**".

Read more about the "**SearchVectorField"** at:

* <https://docs.djangoproject.com/en/5.0/ref/contrib/postgres/search/#searchvectorfield>

**This exercise is not amenable to evaluation through the Judge system.**

### Examples

|  |
| --- |
| **Test Code - caller.py** |
| from django.contrib.postgres.search import SearchVector  # Create the first 'Document' object with a title and content. document1 = Document.objects.create(  title="Django Framework 1",  content="Django is a high-level Python web framework for building web applications.", )  # Create the second 'Document' object with a title and content. document2 = Document.objects.create(  title="Django Framework 2",  content="Django framework provides tools for creating web pages, handling URL routing, and more.", )  # Update the 'search\_vector' field in the 'Document' model with search vectors. Document.objects.update(search\_vector=SearchVector('title', 'content'))  # Perform a full-text search for documents containing the words 'django' and 'web framework'. results = Document.objects.filter(search\_vector='django web framework')  # Print the search results. for result in results:  print(f"Title: {result.title}") |
| **Output** |
| Title: Django Framework 1  Title: Django Framework 2 |

### Hint

The "**SearchVectorField"** in Django is a specialized field used in combination with **PostgreSQL's** full-text search capabilities. It allows you to efficiently perform **full-text searches** on the textual content of your **model** instances.

When data is saved to a "**SearchVectorField"**, the text content is preprocessed, which typically includes

* **Tokenization**: Breaking text into words or tokens.
* **Stemming**: Reducing words to their root form (e.g., "running" becomes "run").
* **Lowercasing**: Converting all text to lowercase for case-insensitive searches.
* **Removing stopwords**: Eliminating common words like "and," "the," "is," etc.
* Other text transformations.



The "**SearchVectorField"** field comes from Django's **PostgreSQL**-specific extension for advanced text search features. This extension, part of the "**django.contrib.postgres"** module, extends Django's capabilities to work seamlessly with **PostgreSQL's** advanced text search functionality.

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично