**Model Form Inheritance in Django 5**

**VIDEO 42**

**CODE 36**

# What “ModelForm inheritance” means

* You define a **base ModelForm** that targets a model (Meta.model = Profile) and contains shared config (widgets, labels, help\_texts, clean methods, etc.).
* You make **subclasses** of that form and **override** only what differs (e.g., the fields list, labels for those fields, extra fields, or extra validation).
* You can inherit the base form’s Meta using class Meta(Base.Meta): ... and then **change only what you need** (like fields).

**Why use it?**

* Avoid duplication (DRY).
* Keep one source of truth for common widgets/labels/validation.
* Make multiple specialized forms for the same model (e.g., Student vs Teacher) that save to the **same table**.

# Important gotchas (fix these or you’ll hit validation errors)

1. **Required model fields** you do NOT include in a form must still have a value.  
   In your model, both student\_name and teacher\_name are required (blank=False by default).
   * When you submit the **student form**, there is no teacher\_name value → validation error.
   * When you submit the **teacher form**, there is no student\_name value → validation error.  
     **Fix**: mark the “other” name as optional in the model: blank=True. Or set it programmatically before saving (via form.save(commit=False)).
2. **Never store plain text passwords** in real apps. (I’ll keep your field names but note this is only for learning. In production, use Django’s User model or hash manually.)
3. fields and exclude in Meta are **mutually exclusive**—don’t set both.
4. If you inherit Meta (class Meta(Parent.Meta): ...) and you **replace** fields, that fully replaces the list; it doesn’t merge automatically. You can merge dicts (labels/widgets) yourself if needed.

# Clean, working example (with comments)

I’ll keep your model/field names (including password) and show both **student** and **teacher** forms derived from a common base.

## models.py

*# school/models.py*

**from** django**.**db **import** models

class **Profile**(*models***.***Model*)**:**

*# Mark both names as optional so each specialized form can leave the other blank*

    student\_name **=** models**.**CharField(**max\_length=**255**,** **blank=**True)   *# blank=True -> not required in forms*

    teacher\_name **=** models**.**CharField(**max\_length=**255**,** **blank=**True)   *# same reason*

    email **=** models**.**EmailField(**max\_length=**255)

    password **=** models**.**CharField(**max\_length=**255)  *# for demo only; don't store plain passwords in real apps!*

**def** \_\_str\_\_(*self*)**:**

        who **=** *self***.**student\_name **or** *self***.**teacher\_name **or** "Unnamed"

**return** f"{who} <{*self***.**email}>"

**Why blank=True?**  
So the **student form** can save without a teacher name and vice-versa. If you keep them required, you must fill the missing one in form.save(commit=False).

## forms.py (inheritance done right)

*# school/forms.py*

**from** django **import** forms

**from** school**.**models **import** Profile

*# Base form (shared config for both variants)*

class **BaseProfileForm**(*forms***.***ModelForm*)**:**

    class **Meta:**

        model **=** Profile

*# Put truly shared fields here or leave empty and override in children*

*# We'll leave it empty and let children set their own fields*

        fields **=** []

*# You can also define shared widgets/labels/help\_texts here:*

        labels **=** {

            'email'**:** 'Email'**,**

            'password'**:** 'Password'**,**

        }

        widgets **=** {

            'password'**:** forms**.**PasswordInput()**,**  *# render password as <input type="password">*

        }

*# Shared cross-field validation goes here if needed*

*# def clean(self):*

*#     cleaned = super().clean()*

*#     return cleaned*

class **StudentRegistration**(*BaseProfileForm*)**:**

    """

    Student-specific form: exposes student\_name + common fields.

    Inherits model and widgets/labels from BaseProfileForm.Meta.

    """

    class **Meta**(*BaseProfileForm***.***Meta*)**:**

        fields **=** ['student\_name'**,** 'email'**,** 'password']

*# Optionally tweak labels just for this form*

        labels **=** {

**\*\***BaseProfileForm**.**Meta**.**labels**,**

            'student\_name'**:** 'Student name'**,**

        }

class **TeacherRegistration**(*BaseProfileForm*)**:**

    """

    Teacher-specific form: exposes teacher\_name + common fields.

    """

    class **Meta**(*BaseProfileForm***.***Meta*)**:**

        fields **=** ['teacher\_name'**,** 'email'**,** 'password']

        labels **=** {

**\*\***BaseProfileForm**.**Meta**.**labels**,**

            'teacher\_name'**:** 'Teacher name'**,**

        }

**Notes**

* We used a **BaseProfileForm** to centralize Meta.model and shared widgets/labels.
* Each child form only lists the fields it needs.
* By using \*\*BaseProfileForm.Meta.labels we **merge** dictionaries (so we don’t lose shared labels).

Your original class name teacher\_Registration works, but Django style is TeacherRegistration (CamelCase for classes).

## views.py

*# school/views.py*

**from** django**.**shortcuts **import** render**,** redirect

**from** school**.**forms **import** StudentRegistration**,** TeacherRegistration

**def** student\_form\_view(**request**)**:**

**if** request**.**method **==** 'POST'**:**

        form **=** StudentRegistration(request**.**POST)

**if** form**.**is\_valid()**:**

*# If you needed to set teacher\_name here, you'd use commit=False and set it.*

*# instance = form.save(commit=False)*

*# instance.teacher\_name = ''*

*# instance.save()*

            form**.**save()  *# creates a Profile row with student\_name + email + password*

**return** redirect('student\_form')  *# PRG pattern; define this URL name in urls.py*

**else:**

        form **=** StudentRegistration()

**return** render(request**,** 'school/studentreg.html'**,** {'form'**:** form})

**def** teacher\_form\_view(**request**)**:**

**if** request**.**method **==** 'POST'**:**

        form **=** TeacherRegistration(request**.**POST)

**if** form**.**is\_valid()**:**

            form**.**save()  *# creates a Profile row with teacher\_name + email + password*

**return** redirect('teacher\_form')  *# PRG pattern*

**else:**

        form **=** TeacherRegistration()

**return** render(request**,** 'school/teacherreg.html'**,** {'form'**:** form})

**What’s happening?**

* Each view binds its specific form class.
* After a successful POST, we redirect (Post/Redirect/Get) to prevent “Resubmit on refresh”.
* Because the model allows blank=True for the name that isn’t present, form.save() just works.

## urls.py (example)

*# school/urls.py*

**from** django**.**urls **import** path

**from** school**.**views **import** student\_form\_view**,** teacher\_form\_view

urlpatterns **=** [

    path('student/'**,** student\_form\_view**,** **name=**'student\_form')**,**

    path('teacher/'**,** teacher\_form\_view**,** **name=**'teacher\_form')**,**

]

## Templates

### templates/school/studentreg.html

**<**!doctype html**>**

**<**html**>**

**<**head**>**

**<**meta charset**=**"utf-8"**>**

**<**title**>**Student Registration**</**title**>**

**</**head**>**

**<**body**>**

**<**h1**>**Student Registration**</**h1**>**

**<**form method**=**"post" novalidate**>**

    {**%** csrf\_token **%**}

    {{ form**.**non\_field\_errors }}

    {{ form**.**as\_p }}

**<**button *type***=**"submit"**>**Save**</**button**>**

**</**form**>**

**</**body**>**

**</**html**>**

### templates/school/teacherreg.html

**<**!doctype html**>**

**<**html**>**

**<**head**>**

**<**meta charset**=**"utf-8"**>**

**<**title**>**Teacher Registration**</**title**>**

**</**head**>**

**<**body**>**

**<**h1**>**Teacher Registration**</**h1**>**

**<**form method**=**"post" novalidate**>**

    {**%** csrf\_token **%**}

    {{ form**.**non\_field\_errors }}

    {{ form**.**as\_p }}

**<**button *type***=**"submit"**>**Save**</**button**>**

**</**form**>**

**</**body**>**

**</**html**>**

You can of course render fields manually (for styling) instead of {{ form.as\_p }}.

## admin.py

*# school/admin.py*

**from** django**.**contrib **import** admin

**from** school**.**models **import** Profile

@admin**.**register(Profile)

class **ProfileAdmin**(*admin***.***ModelAdmin*)**:**

    list\_display **=** ['id'**,** 'student\_name'**,** 'teacher\_name'**,** 'email'**,** 'password']

# Variations & Advanced Notes

### 1) If you don’t want to make those model fields optional

Keep blank=False and set the “other” name in the form’s save(commit=False):

**def** student\_form\_view(**request**)**:**

**if** request**.**method **==** 'POST'**:**

        form **=** StudentRegistration(request**.**POST)

**if** form**.**is\_valid()**:**

            instance **=** form**.**save(**commit=**False)

*# teacher\_name must not be blank if model requires it:*

            instance**.**teacher\_name **=** ''  *# or some default; but better is blank=True in model*

            instance**.**save()

**return** redirect('student\_form')

**else:**

        form **=** StudentRegistration()

**return** render(request**,** 'school/studentreg.html'**,** {'form'**:** form})

But realistically, making the unused name blank=True is cleaner.

### 2) Add extra fields only to a child form

For example, a teacher confirmation password field:

class **TeacherRegistration**(*BaseProfileForm*)**:**

    confirm\_password **=** forms**.**CharField(**widget=**forms**.**PasswordInput)

    class **Meta**(*BaseProfileForm***.***Meta*)**:**

        fields **=** ['teacher\_name'**,** 'email'**,** 'password']  *# confirm\_password is not saved to model*

**def** clean(*self*)**:**

        cleaned **=** super()**.**clean()

        pwd **=** cleaned**.**get('password')

        cpwd **=** cleaned**.**get('confirm\_password')

**if** pwd **and** cpwd **and** pwd **!=** cpwd**:**

*self***.**add\_error('confirm\_password'**,** 'Passwords do not match.')

**return** cleaned

### 3) Reusing labels/widgets across children

That’s exactly what the **Base form** is for (BaseProfileForm.Meta). Children can **override** or **extend** them:

class **StudentRegistration**(*BaseProfileForm*)**:**

    class **Meta**(*BaseProfileForm***.***Meta*)**:**

        fields **=** ['student\_name'**,** 'email'**,** 'password']

        labels **=** {

**\*\***BaseProfileForm**.**Meta**.**labels**,**

            'student\_name'**:** 'Student name'**,**

        }

        widgets **=** {

**\*\***BaseProfileForm**.**Meta**.**widgets**,**

*# You can override per-field widget here if you like*

        }

### 4) Alternative: single form, toggle fields in \_\_init\_\_

Sometimes you want one form class but hide/show fields dynamically:

class **OneFormToRuleThemAll**(*forms***.***ModelForm*)**:**

    class **Meta:**

        model **=** Profile

        fields **=** ['student\_name'**,** 'teacher\_name'**,** 'email'**,** 'password']

**def** \_\_init\_\_(*self***,** **\*args,** **\*\*kwargs**)**:**

        role **=** kwargs**.**pop('role'**,** 'student')  *# pass role='teacher' from view*

        super()**.**\_\_init\_\_(**\***args**,** **\*\***kwargs)

**if** role **==** 'student'**:**

*self***.**fields**.**pop('teacher\_name')

**else:**

*self***.**fields**.**pop('student\_name')

Then in views:

form = OneFormToRuleThemAll(request.POST or None, role='student')

### 5) Quick form without a class: modelform\_factory

Useful for admin-type screens:

from django.forms import modelform\_factory

StudentRegistration = modelform\_factory(Profile, fields=['student\_name', 'email', 'password'], widgets={'password': forms.PasswordInput()})

# Summary

* **ModelForm inheritance** lets you build multiple forms for the **same model** with shared config and per-form differences.
* Keep your **model constraints** in mind. If a form doesn’t include a required model field, either make that field blank=True or set it in save(commit=False) before save().
* Centralize shared stuff (model, widgets, labels, clean methods) in a **base form**, and only override what differs in child forms.
* Use the **PRG** pattern in views (redirect after POST) to prevent resubmission dialogs.
* For real auth, **do not** store clear-text passwords; use Django’s User model or hash passwords.

If you want, I can also show a version with **class-based views (CreateView/UpdateView)** using these forms to reduce boilerplate even further.