# Task 4: CRUD Operations with Class-Based Views and Form Validation

## Objective

This task focuses on building complete CRUD (Create, Read, Update, Delete) functionality using **Django's Class-Based Views (CBVs)** and **form validation**.

You will learn how to use Django's generic views to simplify CRUD operations, validate form input using both field-level and form-level validation, and handle errors and messages properly.

By the end of this task, you should be able to manage data efficiently through CBVs and ModelForms, ensuring clean, maintainable, and user-friendly code.

---

## Questions to Explore

### Class-Based Views (CBVs)

1. What are the main advantages of using CBVs?

2. What are the roles of the generic views: `ListView`, `DetailView`, `CreateView`, `UpdateView`, and `DeleteView`?

3. What is the purpose of `get_queryset()` and `form_valid()` methods, and how can you override them?
4. How does `form.is_valid()` work, and when is `form_valid()` triggered?

5. How do you connect CBVs to URLs using the `as_view()` method?

6. How do you connect a form to a CBV (e.g., `CreateView` or `UpdateView`)?
7. How can you perform field-level validation using the `clean_<fieldname>()` method?

8. What is the difference between `clean()` (form-level) and `clean_<fieldname>()` (field-level) validation?

9. How can you handle validation errors and display them in templates?
10. How can you use Django's **messages framework** to give users feedback after create, update, or delete actions?

11. How can organizing your code make it easier to maintain and extend in larger projects?

12. Search about CSRF tokens. Why they are important?

---

## Deliverables

You will implement full CRUD functionality for the **Developer** and **Project** models using **Class-Based Views (CBVs)**.
Your code should be well-structured, with clear separation between `forms.py`, `views.py`, and `urls.py`.

**Requirements**

1. **Implement full CRUD** for the Developer and Project models using the following CBVs:

   - `ListView`
   - `DetailView`
   - `CreateView`
   - `UpdateView`
   - `DeleteView`

2. **Use ModelForm** for Create and Update operations to simplify form creation and data saving.
3. **Add form validation**, including:

   - Field-level validation (e.g., `age` must be greater than 18)
   - Form-level validation (e.g., description must not be empty)

4. **Override CBV methods** such as `get_queryset()` and `form_valid()` to customize data retrieval and form processing.
5. **Implement error handling:**

   - Use `get_object_or_404()` for safe object retrieval.
   - Display validation and runtime errors in templates.

- Prevent application crashes on invalid input or missing objects.

6. **Add user feedback using Django's messages framework:**

- Show success messages after creating, updating, or deleting an object.

- Show error messages for invalid forms or failed actions.

7. **Organize your code** into clean and modular files

---

## References


- Django Class-Based Views:
  https://docs.djangoproject.com/en/5.0/topics/class-based-views/

- Django Generic Editing Views (Create, Update, Delete):
  https://docs.djangoproject.com/en/5.0/ref/class-based-views/generic-editing/

- Django Forms and Validation:
  https://docs.djangoproject.com/en/5.0/ref/forms/validation/

- Django Messages Framework:
  https://docs.djangoproject.com/en/5.0/ref/contrib/messages/

- Django ModelForms:
  https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/