Introduction to CRUD

CRUD simply stands for -

C - Create R - Read U - Update D - Delete

CRUD forms the basis of all applications, Django makes it very easy for us to handle this.

Read View

We have already implemented this using list view & detail view.

List View

```
# Home Page / List View
def home_page(request):
    # Retreive all the posts
    all_posts = Post.objects.all()
    context = {
        "posts": all_posts
    }
    return render(request, 'blog/index.html', context)
```

Detail View

```
# Post Detail / Detail View
def post_detail(request, pk):
    # Retreive the post or show a 404 page
    post = get_object_or_404(Post, pk=pk)
    context = {
        'post': post
    }
    return render(request, 'blog/post_detail.html', context)
```

We further wired our views to 'urls' and created 'templates'.

Create View

Till now we have been creating post from admin backend, let's now see how to set up frontend post creation using create views.

```
# Create Post / Create View
def create_post(request):
    # If request is of type POST
    if request.method == 'POST':
```

```
        form = CreatePostForm(request.POST or None)
        if form.is_valid():
            # Save form only if valid
            form.save()
            return redirect('home')
    else:
        # Otherwise request is of type GET
        form = CreatePostForm()
    # Create our context variable and assign our form
    context = {
        'form': form
    }
    return render(request, 'blog/create_post.html', context)
```

Further, as usual we wire up our 'urls' and templates for update view.

Update View

What if you want to edit or update the post ? With update view we can create a simple form to update/edit the post from frontend.

```
# Update Post / Update View
def post_update(request, pk):
    post = get_object_or_404(Post, pk=pk)
    if request.method == 'POST':
        form = UpdatePostForm(request.POST, instance=post)
        if form.is_valid():
            form.save()
            return redirect('post_detail', pk=pk)
    else:
        form = UpdatePostForm(instance=post)
    context = {
        'form': form
    }
    return render(request, 'blog/post_update.html', context)
```

Delete View

Finally, we see how to delete a post using the delete view, but before directly deleting, let us first confirm and then actually delete.

```
# Delete Post Confirmation
def post_delete_confirm(request, pk):
    post = get_object_or_404(Post, pk=pk)
    context = {
        'post': post
    }
    return render(request, 'blog/post_delete_confirm.html', context)
```

Actually deleting the post after confirmation.

```
# Delete Post / Delete View
def post_delete(request, pk):
    post = get_object_or_404(Post, pk=pk)
    post.delete()
    return redirect('home')
```

Note -

After creating the respective views, its important to create the related **'urls'** and **'templates'**.