Treble Eshop — Step■by■Step Implementation (Django + Tailwind v4)

Generated on 2025-09-28 20:39

This document explains, in a practical and verifiable way, how the checklist was implemented for Treble Eshop. Each step describes what was built, why it was needed, how it was wired into Django (views/urls/templates), how Tailwind v4 styling was applied, and what to verify. The intent is to reflect deliberate engineering decisions beyond merely following a tutorial.

1) Core Features: Edit & Delete Product

a) Prepare *ProductForm*

We use a ModelForm to control inputs and validation and to keep view logic thin.

```
# forms.py
class ProductForm(forms.ModelForm):
    class Meta:
        model = Product
        fields = ["name", "category", "price", "rating", "stock", "description", "thumbnail"]
```

b) View: *edit_product*

- Fetch product, enforce ownership, render form on GET, save on POST, then redirect with a success message.

```
# views.py
@login_required
def edit_product(request, pk):
    product = get_object_or_404(Product, pk=pk)
    if product.user != request.user:
        return HttpResponseForbidden()

    form = ProductForm(request.POST or None, request.FILES or None, instance=product)
    if request.method == "POST" and form.is_valid():
        form.save()
        messages.success(request, "Product updated.")
        return redirect("main:show_main")
    return render(request, "edit_product.html", {"form": form, "product": product})
```

c) View: *delete_product* (POST■only)

- Accept POST only to avoid accidental deletes via link prefetching or crawlers.

```
@login_required
def delete_product(request, pk):
    product = get_object_or_404(Product, pk=pk)
    if product.user != request.user:
        return HttpResponseForbidden()
    if request.method != "POST":
        return HttpResponseNotAllowed(["POST"])
    product.delete()
    messages.success(request, "Product deleted.")
    return redirect("main:show_main")
```

d) Routes

```
# urls.py
path("product/<int:pk>/edit/", edit_product, name="edit_product"),
path("product/<int:pk>/delete/", delete_product, name="delete_product"),
```

Validate: owner■only access enforced (403 otherwise), delete via GET returns 405, success messages appear after actions.

2) Templates: Hook Buttons & Modal Trigger

We expose Edit and Delete controls inside each product card. The Delete button opens a confirmation modal and posts to the delete URL.

```
<!-- Card actions (main.html) -->
<a href="{% url 'main:edit_product' product.id %}" class="btn-ghost">Edit</a>
<button type="button"
        data-url="{% url 'main:delete_product' product.id %}"
        data-name="{{ product.name|escapejs }}"
        class="js-open-delete btn-danger-ghost">
  Delete
</button>
```

3) Delete Confirmation Modal (Reusable, POST + CSRF)

```
<!-- One modal for all cards (main.html) -->
<form id="deleteForm" method="POST">{% csrf_token %} ... </form>
<script>
(() => {
  const modal = document.getElementById('deleteModal');
  const form  = document.getElementById('deleteForm');
  const nameEl = document.getElementById('delName');
  document.querySelectorAll('.js-open-delete').forEach(btn => {
    btn.addEventListener('click', () => {
      form.action = btn.dataset.url;
      nameEl.textContent = btn.dataset.name;
      modal.classList.remove('hidden');
    });
  });
  // handlers to close via backdrop/Esc/Cancel...
})();
</script>
```

Why: Prevent accidental deletes; keep semantics (POST) and CSRF protection.

4) Project Wiring (URLs, Navigation, includes)

URLs connect views; templates use `{% url %}` for stable navigation. We include a global navbar, and the detail/edit pages link back to the main list. This keeps flows predictable and testable.

5) Tailwind v4 (CDN) Setup & Global Form Styling

We prefer CDN for speed. In base.html's , load the runtime script, define theme tokens, and add a `.form-style` layer so all loop-rendered fields look consistent without per-field classes.

```
<script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
<style type="text/tailwindcss">
  @theme { --color-button:#54826b; --color-button-hover:#3b5c4c; }
  @layer base {
    .form-style :where(input:not([type=checkbox]):not([type=radio]), select, textarea){
      @apply block w-full rounded-md border border-gray-300 bg-white px-3 py-2 text-sm
             focus:ring-2 focus:ring-[--color-button] focus:border-[--color-button] outline-none
    }
    .form-style textarea{@apply h-32 resize-y;}
  }
</style>
```

6) Styling Login & Register

Center the auth card with `min-h-svh flex items-center justify-center`, use real borders (`border-gray-300`) instead of outlines, and optional focus effects (scale/ring) for better affordance.

7) Product Pages: Create / Edit / Detail / Main

Create/Edit: Loop fields, rely on `.form-style`, and show a small thumbnail preview if a file/URL is provided.
Detail: Two-column responsive grid. Media box with either `object-cover` for immersive look or `object-contain` to avoid cropping. Quick facts in a card, actions grouped.
Main (cards): Flexible card with fixed cover ratio, price emphasized, description clamped, star rating computed via CSS width (`calc(rating/5*100%)`), views on the right, actions fixed to bottom.

```
<!-- Stars inside card -->
<span class="relative inline-block leading-none">
  <span class="text-gray-300 select-none">★★★★★</span>
  <span class="absolute inset-0 overflow-hidden text-amber-500 select-none"
        style="width: calc({{ product.rating|default:'0' }} / 5 * 100%);">
    ★★★★★
  </span>
</span>
```

8) Security & UX Considerations

• Restrict edit/delete to owners (server-side check).
• Use POST for deletions and include CSRF token.
• Display Django messages for user feedback.
• Ensure keyboard accessibility for modal (Esc to close, focus management optional).
• Provide alt text and clear labels for better accessibility.

9) End■to■End Validation Checklist

• User A can create/edit/delete their products; cannot modify User B’s (403).
• Delete via GET returns 405; via modal POST succeeds and shows a success message.
• Responsive grids: mobile → single column, ≥sm → multi-column; filter bar stacks nicely and 'Last login' centers on mobile.
• Card actions (Edit/Delete) work independently; stretched links do not capture clicks.
• Star ratings render correctly for integers and decimals (e.g., 4.5 → 90%).

10) What’s Different from a Tutorial

• Delete confirmation via modal with secure POST semantics.
• Loop■only form rendering + global Tailwind `.form-style` (no per-field classes).
• Production■friendly ownership checks and HTTP method enforcement.
• Reusable, responsive card components with price/rating/metrics.
• Attention to UX details: sticky header spacing, mobile filter behavior, and consistent visual rhythm.