Group Practicum Jobsheet: Meeting 3

Frontend & Templating with Blade, Tailwind, and Alpine (Group Work)

Course	Advanced Web Programming (SIB245007)
Meeting	3 (Week 3)
Duration	2 sessions × 170 minutes
Sub-CPMK	Sub-CPMK 2: Students can apply models, templating, and CRUD operations in framework-based web application development.
Work Mode	Group (as assigned by your instructor), one shared GitHub repository per group
Starting Code	template repository `github.com/se-polinema/simple-pos-ch02`

A. Practicum Outcomes

After completing this group jobsheet, you'll be able to:

1. Set up a shared GitHub repository for group work, then contribute through feature branches and Pull Requests recorded under your own name.
2. Build a Blade layout (`@extends`/`@section`/`@yield`) and a navigation component reused across pages.
3. Style the `/pos` cashier page with Tailwind CSS utility classes loaded through Vite, using `npm run dev` for hot reload.
4. Implement a dynamic shopping cart with Alpine.js (`x-data`, `@click`, `x-for`, `x-text`) without a page reload.
5. Explain why `{{ }}` must be used for user data, and why a subtotal computed on the client must be recomputed on the server.

B. Preparation and Prerequisites

- **Tools**: same as Meeting 2 (PHP 8.2+, Composer, Node.js, Git), plus an active GitHub account for every group member.
- **Group assignment**: work in the group your instructor assigned. This jobsheet doesn't decide who does which step, that's the group's own call. Make sure the split leaves Steps 5 through 10, and the independent challenge in Step 12, spread evenly so each member gets at least one meaningful commit recorded through their own Pull Request.
- **Per-member git identity**: before starting, every member checks their own laptop's git identity:

  ```
  git config --global user.name "Full Name"
  git config --global user.email "your-github-account-email@..."
  ```

  This email should match your GitHub account's email, so every commit, including the ones inside your Pull Request, links back to your profile and your contribution counts when graded.

  > ⚠ **If you're using a shared lab computer:** use `git config --local` (no `--global`) inside the project folder you already cloned, so your git identity doesn't linger for the next person.

- **The group's starting code**: the group starts from the `simple-pos-ch02` template repository, not from one member's personal project. Your individual project from Meeting 1 and 2 stays your own and isn't used in this group jobsheet.

C. Work Steps

The group works with feature branches: each build step (Step 5 through 10) happens on its own branch off `main`, then gets merged into `main` through a Pull Request (PR) on GitHub. Some steps

depend on each other, the cashier page needs the layout first, the cart needs the cashier page first, so always start from an up-to-date `main` (`git pull`) before creating a new branch, so your branch doesn't fall behind Pull Requests other members have already merged.

Step 1: Creating the group's repository from the template

`simple-pos-ch02` is already set up as a template repository: the group just makes its own copy through GitHub, with no manual clone or remote setup needed. One member does this step first.

1. Open `https://github.com/se-polinema/simple-pos-ch02`.
2. Click the green **Use this template** button → **Create a new repository**.
3. Choose your account as the owner, name the repository, e.g. `simple-pos-kelompok-03`, choose Public or Private, then click **Create repository**.
4. On the newly created repository's page, open **Settings → Collaborators**, then add every group member and the instructor grading you.

> ✓ **Checkpoint:** the new repository contains every Laravel project file (`composer.json`, `app/`, `routes/`, etc.) but only has **one commit** in its history, because GitHub creates a repository from a template by copying its file contents only, not `simple-pos-ch02`'s commit history. Every member has accepted the collaborator invitation.

> ⚠ **If it fails:** if the **Use this template** button doesn't show up, make sure you're viewing the `simple-pos-ch02` repository page itself (not a search result), and that you're logged into GitHub.

Step 2: Every member clones and sets up

Each member, on their own laptop:

```
git clone https://github.com/<repo-creator-username>/simple-pos-kelompok-03.git
cd simple-pos-kelompok-03
composer install
npm install
cp .env.example .env
php artisan key:generate
touch database/database.sqlite
php artisan migrate:fresh --seed
```

> ✓ **Checkpoint:** `git log --oneline` shows the exact same line on every laptop, and `php artisan serve` runs with no error.

Step 3: Agreeing on the feature branch and Pull Request workflow

Before writing any code, the group agrees to use the following workflow for every work step (Step 5 through 10):

```
git checkout main
git pull
git checkout -b short-branch-name
# ...do your task...
git add .
git commit -m "pesan singkat sesuai tugas"
git push -u origin short-branch-name
```

Give the branch a clear name, e.g. `base-layout` or `alpine-cart`, so the whole group can recognize it easily. After pushing, open the repository on GitHub and create a Pull Request from that branch to `main`. Click **Merge pull request** with the **Create a merge commit** option (not **Squash and merge**), so the original commits and your name stay in the history, then delete the branch. Finally, go back to `main` and pull the result:

```
git checkout main
git pull
```

If there's time, ask another member to review your Pull Request before it's merged, so the group gets used to code review.

> ⚠ **If it fails:** GitHub showing a conflict when creating a Pull Request means your branch has fallen behind `main`. Run `git checkout main && git pull`, then `git checkout your-branch-name && git merge main`, resolve the conflict, and push again.

Step 4: Every member runs Vite

```
npm run dev
```

Run this in a second terminal, separate from `php artisan serve`. While it's running, open `resources/css/app.css` and confirm its first line is `@import 'tailwindcss';`, then peek at `vite.config.js` and notice the `laravel()` and `tailwindcss()` plugins already registered since Meeting 1.

> ✓ **Checkpoint:** the terminal shows a line `VITE vX.X.X ready` followed by a local address like `http://localhost:5173/`.

> ⚠ **If it fails:** opening `/pos` in the browser and seeing a `ViteManifestNotFoundException` means `npm run dev` isn't running in that terminal.

Once you're comfortable with these two terminals, composer run dev runs php artisan serve, npm run dev, and other processes together in a single terminal, through a script already defined in Laravel's default composer.json.

Step 5: Simplifying `TransactionController`

The group's starting code (the `simple-pos-ch02` template) already contains a production version of `TransactionController` that calls the `Product` Model and a few other Models. You'll build those Models yourself in the database design meeting, so for now simplify the controller so the cashier page can be built without waiting on a Model. This file has existed since it was created via `php artisan make:controller TransactionController` in Meeting 2, so this step only edits its contents, it doesn't create a new file.

Create a new branch off the latest `main`, e.g. `simplify-controller`:

```
git checkout main
git pull
git checkout -b simplify-controller
```

Replace the entire contents of `app/Http/Controllers/TransactionController.php` with:

```
<?php

namespace App\Http\Controllers;

class TransactionController extends Controller
{
    public function create()
    {
        $products = collect([
            (object) ['id' => 1, 'name' => 'Kopi Sachet', 'price' => 3000, 'stock' =>
            ↪ 40],
            (object) ['id' => 2, 'name' => 'Teh Celup', 'price' => 2500, 'stock' => 25],
            (object) ['id' => 3, 'name' => 'Mie Instan', 'price' => 3500, 'stock' => 8],
            (object) ['id' => 4, 'name' => 'Air Mineral 600ml', 'price' => 4000, 'stock'
            ↪ => 60],
            (object) ['id' => 5, 'name' => 'Roti Tawar', 'price' => 12000, 'stock' =>
            ↪ 15],
            (object) ['id' => 6, 'name' => 'Gula Pasir 1kg', 'price' => 15000, 'stock' =>
            ↪ 5],
        ]);

        return view('pos.create', ['products' => $products]);
    }

    public function store()
    {
        return 'Transaksi disimpan (belum ada logika penyimpanan)';
```

```
    }

    public function index()
    {
        return 'Daftar transaksi';
    }

    public function show(string $id)
    {
        return "Detail transaksi #{$id}";
    }
}
```

The product data above is deliberately written directly into the controller (not from a database): good enough to build the view for now, and it'll be replaced with a real `Product` model in the next meeting.

```
git add .
git commit -m "sederhanakan TransactionController untuk pertemuan 3"
git push -u origin simplify-controller
```

Open a Pull Request to `main`, merge it (follow Step 3's workflow), then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** opening `/pos` in the browser shows a "view [pos.create] not found" error, no longer a Model error. That's expected, since the view isn't created until Step 8.

Step 6: Building the application layout

Create a new branch off the latest `main`, e.g. `base-layout`. Generate the file via artisan, then replace its contents:

```
php artisan make:view layouts.app
```

The dot notation (`layouts.app`) automatically creates a `layouts/` folder if it doesn't exist, and generates a file at `resources/views/layouts/app.blade.php`. That file's contents become:

```
<!DOCTYPE html>
<html lang="id">
<head>
    <title>@yield('title', 'Simple POS')</title>
    @vite(['resources/css/app.css', 'resources/js/app.js'])
</head>
<body>
    <x-nav />
    <main>@yield('content')</main>
</body>
</html>
```

```
git add .
git commit -m "tambah layout dasar aplikasi"
git push -u origin base-layout
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** the `resources/views/layouts/app.blade.php` file exists on every laptop after pulling. The `/pos` page's appearance hasn't changed yet, this layout only actually gets used in Step 8.

Step 7: Building a navigation component

Create a new branch, e.g. `nav-component`. Generate the file via artisan:

```
php artisan make:component nav --view
```

The `--view` option creates a component with no separate PHP class, just a Blade file, suited to a simple component like this nav. Fill in the newly created `resources/views/components/nav.blade.php`:

```
<nav class="bg-slate-900 text-white px-4 py-3 flex gap-4">
    <span class="font-semibold">Simple POS</span>
```

```
    <a href="{{ route('pos.create') }}" class="hover:underline">Kasir</a>
    <a href="{{ route('transactions.index') }}" class="hover:underline">Transaksi</a>
</nav>
```

Laravel automatically detects this component from its name: it's invoked via `<x-nav />`, as already written in the layout in Step 6.

```
git add .
git commit -m "tambah component nav"
git push -u origin nav-component
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** the `resources/views/components/nav.blade.php` file exists on every laptop.

Step 8: Building the cashier page

Create a new branch, e.g. `cashier-page`. Generate the file via artisan, then replace its contents:

```
php artisan make:view pos.create
```

Fill in `resources/views/pos/create.blade.php`:

```
@extends('layouts.app')

@section('title', 'Kasir')

@section('content')
    <h1 class="text-lg font-semibold mb-4">Transaksi Kasir</h1>
    <div class="grid grid-cols-3 gap-4">
        @foreach ($products as $product)
            <div class="border rounded-md p-3">
                <p class="font-medium">{{ $product->name }}</p>
                <p class="text-sm text-slate-500">Rp {{ number_format($product->price)
                ↪  }}</p>
            </div>
        @endforeach
    </div>
@endsection
```

Notice `{{ $product->name }}`: these double curly braces automatically HTML-escape the value, so a product name containing a character like `<` can't be used to inject foreign markup into the page. Data coming from user input must be shown through `{{ }}`, not `{!! !!}`, which bypasses escaping entirely.

```
git add .
git commit -m "tambah halaman kasir dengan tailwind"
git push -u origin cashier-page
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✅ **Checkpoint:** opening `http://127.0.0.1:8000/pos` shows a grid of six products with the nav above it. While `npm run dev` is running, try changing one Tailwind class in this file and watch the change appear instantly in the browser without a manual refresh.

Step 9: Installing Alpine.js

Create a new branch, e.g. `install-alpine`:

```
npm install alpinejs
```

Replace the entire contents of `resources/js/app.js`:

```
import './bootstrap';
import Alpine from 'alpinejs';

window.Alpine = Alpine;
Alpine.start();
```

```
git add .
git commit -m "instal alpine.js"
git push -u origin install-alpine
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main`, pulls, **and runs `npm install` again** since `package.json` changed too.

> ✔ **Checkpoint:** open the browser console (F12) on the `/pos` page, type `Alpine.version`, and confirm a version number shows up.

> ⚠ **If it fails:** an `Alpine is not defined` error in the console after pulling usually means you forgot to rerun `npm install`, or `npm run dev` hasn't been restarted.

Step 10: Building a dynamic cart

Create a new branch, e.g. `alpine-cart`, then change `resources/views/pos/create.blade.php` to:

```
@extends('layouts.app')

@section('title', 'Kasir')

@section('content')
    <h1 class="text-lg font-semibold mb-4">Transaksi Kasir</h1>
    <div x-data="{
        cart: [],
        addToCart(id, name, price) {
            this.cart.push({ id, name, price });
        },
        subtotal() {
            return this.cart.reduce((sum, item) => sum + item.price, 0);
        }
    }">
        <div class="grid grid-cols-3 gap-4">
            @foreach ($products as $product)
                <div class="border rounded-md p-3 cursor-pointer"
                     @click="addToCart({{ $product->id }}, '{{ $product->name }}', {{
                      ↪  $product->price }})">
                    <p class="font-medium">{{ $product->name }}</p>
                    <p class="text-sm text-slate-500">Rp {{
                     ↪  number_format($product->price) }}</p>
                </div>
            @endforeach
        </div>

        <div class="mt-4 border-t pt-3">
            <template x-for="item in cart" :key="item.id">
                <p x-text="item.name + ' - Rp ' + item.price"></p>
            </template>
            <p class="font-semibold mt-2">Subtotal: Rp <span
             ↪  x-text="subtotal()"></span></p>
        </div>
    </div>
@endsection
```

```
git add .
git commit -m "tambah keranjang dinamis dengan alpine"
git push -u origin alpine-cart
```

Open a Pull Request to `main`, merge it, then everyone goes back to `main` and pulls.

> ✔ **Checkpoint:** clicking a product card immediately adds its name to the summary area below the grid and the subtotal increases, with no page reload (watch the address bar and tab favicon, neither should flicker).

> ⚠ **If it fails**, check the three most common causes: (a) the browser console shows `Alpine is not defined`, meaning `npm run dev` isn't running or Step 9 wasn't saved; (b) the page shows a `ViteManifestNotFoundException`, meaning `npm run dev` hasn't been run at all; (c) make

> sure the entire product grid and the summary area are truly inside the element carrying `x-data`, not sibling elements outside it.

Step 11: Together, integration test and code review

Everyone runs `git checkout main && git pull`, starts both servers, and repeats the click test from Step 10 on their own laptop. After that, the group reads through the code together: each member explains one file they **didn't** write themselves.

> ✓ **Checkpoint:** the `/pos` page behaves exactly the same on every member's laptop.

Step 12: Group independent challenge and the `increment 3` commit

Split the following tasks among the group's members, so each member gets at least one meaningful commit recorded through their own Pull Request:

- Add a *Remove* button to each cart line item, calling a new Alpine method that removes that item from the `cart` array by its `id`.
- Add a small badge that reads "Stok Menipis" using the Tailwind classes `bg-amber-100 text-amber-700`, shown only when a product's `stock` is below 10.
- Add a highlight (e.g. the Tailwind class `ring-2 ring-blue-500`) on the product card that was just clicked.
- Tidy up the nav's appearance: add spacing between links, and mark the currently active page's link.
- Add a different `@section('title', 'Riwayat Transaksi')` for the `index` method. Create its stub view via `php artisan make:view transactions.index`, which generates `resources/views/transactions/index.blade.php`.

For each task: create a new branch (e.g. `remove-cart-item`), do the work, commit, push, then open and merge its Pull Request (follow Step 3's workflow).

Once every Pull Request is in and merged, one member closes out the group's work:

```
git checkout main
git pull
git commit --allow-empty -m "increment 3: tampilan kasir dengan blade, tailwind, dan
↪  alpine"
git push
git log --pretty="%h %an %s"
```

> ✓ **Checkpoint:** the top line of `git log --pretty="%h %an %s"` shows `increment 3: ...`, and the lines below it show every group member's name.

D. Tasks and Deliverables

Submit the following in the format your instructor requests:

- Link to the group's GitHub repository.
- Screenshot of the `/pos` page with at least 2 items in the cart and a filled-in subtotal.
- Output of `git log --pretty="%h %an %s"` showing at least one commit per member.
- A task-division table: member name | step/task done | commit hash.
- **Independent task (done and submitted by each member individually):** explain in your own words, in 3-5 sentences: (a) why `{{ }}` is safer to use for displaying a product name than `{!! !!}`, and (b) why a subtotal computed by Alpine.js in the cart must not be directly trusted by the server as the final transaction total.

E. Grading Rubric

Component	Weight	Full Marks (100%)	Minimum Marks
Work steps completed (group)	30%	Steps 1-12 done, `/pos` works with a dynamic cart	Most steps done, the cashier page displays

Component	Weight	Full Marks (100%)	Minimum Marks
Checkpoints verified (group)	20%	Screenshots, task-division table, and a complete, correct git log	Some checkpoints proven
Per-member contribution (individual)	25%	At least one meaningful commit under each member’s name, matching the task-division table	Commits exist but are small or their relevance is unclear
Independent task (individual)	15%	Both explanations are accurate and self-written	An answer exists but is incomplete
Repository and commit hygiene	10%	The `increment 3` message is exact, no `vendor/`/`node_modules/` is included, PRs merged cleanly (not squashed)	Commits exist, but the message is messy or a PR was squashed