Pharmacy Inventory Management System

WITH LARAVEL & FILAMENT

Dr. Ye Thi Ha Htwe

A Step-by-Step Guide to Creating Web Applications

CREATING PHARMACY INVENTORY MANAGEMENT APPLICATION WITH LARAVEL & FILAMENT

DR. YE THI HA HTWE

For Bun Bun

Creating Pharmacy Inventory Management Application with Laravel Filament

ဤစာအုပ်မှာ HTML ၊ CSS ၊ PHP အကြောင်း အသေးစိတ် ရှင်းလင်းချက်များ မပါဝင်ဘဲ Laravel Framework နှင့် Filament တို့ကိုတိုက်ရိုက်အသုံးပြုမည်ဖြစ်သောကြောင့် အောက်ပါတို့ကိုကြိုတင်လေ့လာထားရန်လိုပါတယ်

- HTML အခြေခံ ဗဟုသုတများ
- CSS အခြေခံ ဗဟုသုတများ
- Visual Studio Code ကဲ့သို့ Web development tools များ အသုံးပြုပုံ
- Browser developer tools အသုံးပြု၍ HTML နှင့် CSS စစ်ဆေးခြင်းနှင့် debug လုပ်ခြင်း

1 Introduction

Laravel နှင့် Filament သုံးပြီးဆေးစတိုပစ္စည်းစာရင်း စီမံခန့်ခွဲမှုစနစ် တည်ဆောက်ခြင်းအကြောင်း ဖော်ပြမှာဖြစ်ပါတယ်။ ပရောဂျက်လုပ်ရင်း နောက်ပြန်လေ့လာတဲ့ project-based learning approach အတိုင်းရေးသားပါတယ်။ web application project မို့လို့ ပြီးစီးသွားတဲ့အခါ website တစ်ခုအနေနဲ့အသုံးပြုနိုင်မှာဖြစ်ပါတယ်။ ဆာဗာနဲ့ ဒိုမိန်းနိမ်းဝယ်ပြီး အွန်လိုင်းပေါ်သုံးဖို့အဆင်သင့်မဖြစ်သေးရင်တောင် ကွန်ပြူတာထဲမှာပဲ localhost မှာ run ပြီးသုံးနေလို့ရပါတယ်။ အွန်လိုင်းပေါ်တင်သည်ဖြစ်စေ၊ မတင်သည်ဖြစ်စေ development လုပ်နေတဲ့အချိန်မှာတော့ ကွန်ပြူတာမှာပဲ localhost run ပြီးဖွင့်ထားဖို့လိုပါတယ်။ အဲဒီလို localhost လုပ်ဖို့အတွက် Laravel Herd နဲ့ XAMPP ဆိုတဲ့ ပရိုဂရမ်လိုအပ်ပါတယ်။

Web application တည်ဆောက်ဖို့အတွက် ဆာဗာတွေကွန်ပြူတာထဲမှာဘာလို့ run ထားဖို့လိုသလဲဆိုတော့ ကျွန်တော်တို့လုပ်မယ့် application မှာ server-side language ဖြစ်တဲ့ PHP ကိုအခြေခံထားတဲ့ Laravel framework ကိုသုံးမှာမို့လို့ပါ။

1.1 Laravel

Laravel ဟာလူကြိုက်အများဆုံး Web Application Framework ထဲကတစ်ခုဖြစ်ပါတယ်။ PHP ဘာသာစကားကို အခြေခံထားပြီး MVC ပုံစံ (Model-View-Controller Pattern) ကို အသုံးပြုထားပါတယ်။ Taylor Otwell က ၂၀၁၁ ခုနှစ်မှာ စတင်တီထွင်ခဲ့ပါတယ်။ ဒေတာအသွင်းအထုတ်လုပ်တဲ့ database transactions တွေအတွက် Eloquent ORM (Object-Relational Mapping) ကို အသုံးပြုထားပါတယ်။

User registration, login, logout authentication အပိုင်းနဲ့ ဘယ် user ကို ဘယ်အတိုင်းအတာထိပေးလုပ်မယ်ဆိုတဲ့ authorization စနစ်တွေကို ကိုယ့်ဘာသာကိုယ်အစအဆုံးရေးစရာမလိုအောင် အသင့်ထည့်သွင်းပေးထားပါတယ်။ Email Services၊ Job Queues၊ Caching တွေလည်း အသင့်ပါပြီးသားမို့လို့ အချိန်တိုတိုနဲ့ အပလီကေးရှင်းတွေဖန်တီးနိုင်ပါတယ်။

1.2 Server-side Languages

PHP

PHP ဟာ server-side scripting ဘာသာစကားတစ်ခုဖြစ်ပြီး၊ ဝက်ဘ်ဆိုဒ်များနှင့် ဝက်ဘ်အပ်ပလီကေးရှင်းများ ဖန်တီးရန် အသုံးများသည့် ပရိုဂရမ်မင်းဘာသာစကားတစ်ခု ဖြစ်ပါတယ်။ PHP ဟာသက်တမ်းရင့်နေတဲ့နည်းပညာဖြစ်ပေမယ့် လွယ်ကူရိုးရှင်းပြီး သင်ယူရ လွယ်ကူသောကြောင့် ကမ္ဘာတစ်ဝှမ်းရှိ ဝက်ဘ်ဆိုဒ်အများစုတွင် အသုံးပြုနေကြဆဲဖြစ်ပါတယ်။ ယခုစာအုပ်မှာဆက်လက်ဖော်ပြမယ့် Laravel နည်းပညာဟာ PHP ကိုအခြေခံတည်ဆောက်ထားတာဖြစ်ပါတယ်။

ဒေတာဘေ့စ် (Database) နှင့် ချိတ်ဆက်အသုံးပြုရာတွင် လွယ်ကူခြင်း၊ စာမျက်နှာ (HTML) နှင့် ရောနှောရေးသားနိုင်ခြင်း၊ ဆာဗာ (Server) ပေါ်တွင် အလွယ်တကူ တပ်ဆင်အသုံးပြုနိုင်ခြင်း၊ အခမဲ့သုံးနိုင်ခြင်း၊ ဝက်ဘ်ဆာဗာ (Web Server) အမျိုးမျိုးပေါ်တွင် အသုံးပြုနိုင်ခြင်းတို့ကြောင့် JavaScript လိုနည်းပညာအသစ်တွေပေါ်လာပေမယ့် PHP ကိုလည်းဆက်လက်အသုံးပြုနေကြဆဲဖြစ်ပါတယ်။

Python

Python ဟာ ရိုးရှင်းလွယ်ကူပြီး အသုံးများတဲ့ ပရိုဂရမ်းမင်းဘာသာစကားတစ်ခုဖြစ်ပါတယ်။ ဆာဗာဘက်ခြမ်းသာမက သိပ္ပံနည်းကျတွက်ချက်မှုများ (scientific calculations)၊ စက်မှုလုပ်ငန်းသုံး ပရိုဂရမ်များ (industrial programs)၊ အလိုအလျောက်လုပ်ဆောင်ချက်များ (automations) အတွက်ပါ အသုံးပြုနိုင်ပါတယ်။ အထူးသဖြင့် စက်ယန္တရားသင်ယူမှု (Machine Learning) နှင့် ဒေတာသိပ္ပံ (Data Science) နယ်ပယ်တွေမှာ အသုံးအများဆုံး ဘာသာစကားတစ်ခု ဖြစ်ပါတယ်။

Ruby

Ruby ဟာ ဂျပန်နိုင်ငံက ဖန်တီးထားတဲ့ ပရိုဂရမ်းမင်းဘာသာစကားဖြစ်ပြီး အထူးသဖြင့် ဝဘ်အပ်ပလီကေးရှင်းတွေ ဖန်တီးဖို့အတွက် Ruby on Rails နည်းပညာနဲ့တွဲဖက်ပြီး အသုံးများပါတယ်။ ရေးသားရတာ သပ်ရပ်လှပပြီး နားလည်ရလွယ်ကူတဲ့ ဘာသာစကားတစ်ခုဖြစ်ပါတယ်။

Java

Java ဟာ ကွန်ပျူတာပလက်ဖောင်းအမျိုးမျိုးပေါ်မှာ အလုပ်လုပ်နိုင်တဲ့ အားသာချက်ရှိပြီး လုံခြုံရေးကောင်းမွန်တဲ့ ဘာသာစကားတစ်ခုဖြစ်ပါတယ်။ အင်တာနက်ခေတ်မတိုင်ခင်ကတည်းက ကျယ်ပြန့်စွာအသုံးပြုခဲ့ကြပြီး ယနေ့ခေတ်မှာလည်း အန်းဒရွိုက်လက်ကိုင်ဖုန်းအက်ပ်များ၊ စီးပွားရေးလုပ်ငန်းသုံးဆော့ဖ်ဝဲများ ဖန်တီးရာမှာ အဓိကအသုံးပြုကြပါတယ်။

Node.js

Node.js ဟာ JavaScript ကို ဆာဗာဘက်ခြမ်းမှာ အသုံးပြုနိုင်အောင် ဖန်တီးထားတဲ့ ပလက်ဖောင်းတစ်ခုဖြစ်ပါတယ်။ တခြားနည်းပညာတွေထက် မြန်နှုန်းသိသိသာသာမြင့်ပြီး non-blocking I/O နဲ့ asynchronous processing လိုတစ်ပြိုင်နက်တည်း လုပ်ဆောင်မှုများစွာကို ကိုင်တွယ်နိုင်တဲ့ စွမ်းဆောင်ရည်ကြောင့် ခေတ်စားလာတဲ့ နည်းပညာတစ်ခုဖြစ်ပါတယ်။

ASP.NET

ASP.NET ဟာ မိုက်ခရိုဆော့ဖ်ကုမ္ပဏီက ဖန်တီးထားတဲ့ နည်းပညာတစ်ခုဖြစ်ပါတယ်။ စီးပွားရေးလုပ်ငန်းကြီးများနဲ့ ကုမ္ပဏီကြီးများမှာ အသုံးများပြီး၊ C# ဘာသာစကားကို အခြေခံထားပါတယ်။ လုံခြုံရေးကောင်းမွန်ပြီး ထိန်းသိမ်းရလွယ်ကူတဲ့ အားသာချက်တွေရှိပါတယ်။

1.3 Client-side Languages

HTML (Hypertext Markup Language)

HTML ဟာ ဝက်ဘ်စာမျက်နှာများဖွဲ့စည်းတည်ဆောက်ပေးတဲ့ စနစ်တစ်ခုဖြစ်ပါတယ်။ ဝက်ဘ်စာမျက်နှာပေါ် ရှိ စာသား၊ ပုံရိပ်၊ လင့်ခ်များကိုဖော်ပြရန် အသုံးပြုပြီး Chrome ၊ Edge ၊ Safari စတဲ့ ဘရောက်ဇာများက နားလည်နိုင်သော အခြေခံဘာသာစကားတစ်ခု ဖြစ်ပါတယ်။

CSS (Cascading Style Sheets)

CSS ဟာ HTML elements များ၏ ပုံသဏ္ဌာန်နှင့်အပြင်အဆင်ပိုင်းအတွက် style sheet language တစ်ခုဖြစ်ပါတယ်။ HTML ကို ပိုမိုလှပပြီး စနစ်ကျစေရန် အရောင်၊ ဖောင့်၊ layout နှင့် အခြားဒီဇိုင်းဆိုင်ရာ ပြင်ဆင်မှုများပြုလုပ်နိုင်ပါတယ်။

JavaScript

JavaScript ကတော့ dynamic ဖြစ်တဲ့၊ interactive ဖြစ်တဲ့ ဝက်ဘ်စာမျက်နှာများ ဖန်တီးဖို့ programming language တစ်ခုဖြစ်ပါတယ်။ အသုံးပြုသူနှင့် ချိတ်ဆက်မှု (user interaction)၊ animations များ၊ အချိန်နှင့်တပြေးညီ real-time updates များကို ဖန်တီးနိုင်စွမ်း ရှိပါတယ်။ JavaScript ဟာ ဝက်ဘ်ဘရောက်ဇာပေါ်တွင် run နိုင်သကဲ့သို့ ဆာဗာပေါ်တွင်လည်း (Node.js မှတဆင့်) run နိုင်ပါတယ်။

1.4 Client-side and server-side

Client-side scripting languages များဟာ အသုံးပြုသူ၏ ဝက်ဘ်ဘရောက်ဇာပေါ်တွင် run သော programming language များ ဖြစ်ပါတယ်။ ဥပမာအားဖြင့် JavaScript ဟာ client-side scripting language တစ်ခုဖြစ်ပါတယ်။ ဝက်ဘ်စာမျက်နှာပတ်ဝန်းကျင်တွင်လှုပ်ရှားနိုင်ပြီး server နှင့် ထပ်မံဆက်သွယ်စရာမလိုဘဲ လုပ်ဆောင်နိုင်ပါတယ်။ တနည်းအားဖြင့် ဆာဗာနှင့်အသွားအပြန်ချိတ်ဆက်လုပ်ဆောင်ရန် မလိုသောကြောင့် Client-side scripting က ပိုမိုလျင်မြန်သော user experience ကို ပေးစွမ်းနိုင်ပါတယ်။

ဝက်ဘ်စာမျက်နှာကို View Source/Inspect နှင့်ဖွင့်ကြည့်ပါက Client-side script အားလုံးကိုလူတိုင်းမြင်နိုင်ပါတယ်။

Server-side ဆိုသည်မှာ ဆာဗာပေါ်တွင် run သော code များဖြစ်ပါတယ်။ PHP, Python, Ruby, Java တို့ဟာ server-side programming languages များ ဖြစ်ကြပါတယ်။ Server-side code များမှာ ဒေတာဘေ့စ်နှင့် ချိတ်ဆက်ခြင်း၊ file processing နှင့်

ရှုပ်ထွေးတဲ့တွက်ချက်မှုများဆောင်ရွက်နိုင်ပါတယ်။ လုံခြုံရေးအရ အရေးကြီးသော သတင်းအချက်အလက်များကို **server-side** တွင်သာစီမံခန့်ခွဲလေ့ရှိပါတယ်။ **Server-side code** များဟာ **server** ပေါ်တွင်သာရှိသောကြောင့် ဝက်ဘ်စာမျက်နှာကို **View Source/Inspect** ထောက်ကြည့်ပါက **server-side code** များကို မတွေ့ရနိုင်ပါ။

1.5 Static and Dynamic websites

ဝက်ဘ်စာမျက်နှာတွေကို အကြမ်းဖျဉ်းအားဖြင့် အမျိုးအစားနှစ်မျိုးခွဲခြားနိုင်ပါတယ်။

(၁) Static Websites (၂) Dynamic Websites

Static Website

ဝက်ဘ်ဆိုက်ပေါ်က အကြောင်းအရာတွေက အမြဲတမ်း တစ်သမတ်တည်း ရှိနေတာမျိုးဖြစ်ပါတယ်။ **Static website** တွေကို **HTML, CSS** နဲ့ အခြေခံ **JavaScript** တွေကိုပဲ သုံးထားလေ့ရှိပါတယ်။ **Server** ဆီကနေ **pre-built HTML** ဖိုင်တွေကိုပဲ ပြန်ပို့ပေးတာဖြစ်လို့ **loading** မြန်ပါတယ်။ **Content** ပြောင်းလဲချင်ရင် **code** ထဲက **HTML** ဖိုင်တွေကို တိုက်ရိုက်ပြင်ဆင်ရပါတယ်။ ဥပမာ - **Company profile website, Portfolio website, Landing page** စတာတွေဟာ **static website** အမျိုးအစားဖြစ်ပါတယ်။

Dynamic Website

ဝက်ဘ်ဆိုက်ပေါ်က အကြောင်းအရာတွေဟာ အသုံးပြုသူထည့်သွင်းလိုက်တဲ့ **input, database** ပေါ်က အချက်အလက်တွေပေါ်မူတည်ပြီး အမြဲပြောင်းလဲနေပါတယ်။ **Server-side programming languages (PHP, Python, Node.js)** နဲ့ **database** တွေချိတ်ဆက်ပြီးမှ **dynamic website** တွေဖန်တီးနိုင်ပါတယ်။ အသုံးပြုသူက တခုခု **request** လုပ်တိုင်း **server** က **data** တွေကို **process** လုပ်ပြီးမှ **response** ပြန်ပို့ပေးတာမို့ **static** ထက် နည်းနည်းနှေးပါတယ်။ ပြင်စရာရှိတဲ့အခါမှာလည်း **code** သွားရေးဖို့မလိုဘဲ **content** တွေကို **database** ကနေ တိုက်ရိုက် **update** လုပ်လို့ရပါတယ်။ ဥပမာ - **Social media websites, E-commerce websites, News websites** စသည်တို့ဖြစ်ပါတယ်။

အားသာချက်/အားနည်းချက်များ

Static Website တွေဟာ **Loading** မြန်၊ **hosting** ကုန်ကျစရိတ်သက်သာ၊ **security** ပိုကောင်းပါတယ်။ အားနည်းချက်ကတော့ **Content update** လုပ်ရခက်၊ **user interaction** နည်းပါတယ်။

Dynamic Website တွေက **Content update** လုပ်ရလွယ်၊ **user interaction** များ၊ **user** အကြိုက် **theme** များ၊ ဘာသာစကားများ **personalization** လုပ်နိုင်ပါတယ်။ အားနည်းချက်ကတော့ **Loading** နှေး၊ **hosting** ကုန်ကျစရိတ်များ၊ **maintenance** လုပ်ရခက်တာမျိုးရှိပါတယ်။

ဒါ့အပြင် **Static** နဲ့ **Dynamic** ကို ရောပြီးသုံးတဲ့ **Hybrid websites** တွေလည်း ရှိပါသေးတယ်။ ဥပမာ - **Static site generator** တွေဖြစ်တဲ့ **Gatsby, Next.js** တို့လိုမျိုးပါ။ လိုအပ်တဲ့နေရာမှာ **dynamic feature** တွေထည့်သုံးပြီး **static** ရဲ့အားသာချက်တွေကိုလည်း ရယူနိုင်ပါတယ်။

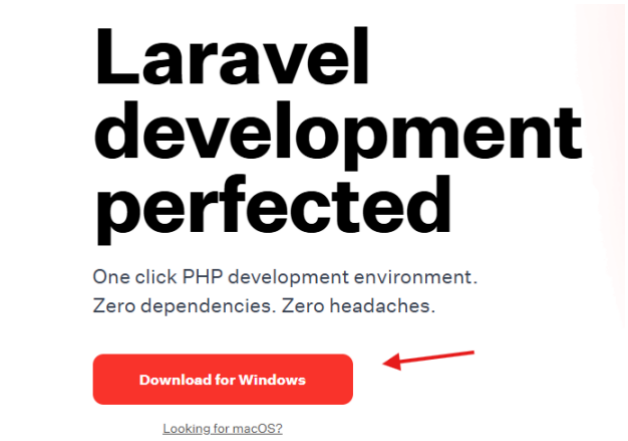
2 Development Setup

2.1 Install Laravel Herd

Laravel Herd ဟာ **Laravel project** တွေဖန်တီးနိုင်မယ့် **developement enviroment** ဖြစ်ပါတယ်။ ပုံမှန်ဆိုရင် **XAMPP** နဲ့တင် **Laravel project** တွေတည်ဆောက်နိုင်ပေမယ့် အခုစာရေးနေတဲ့အချိန်မှာ **XAMPP** နောက်ဆုံး **version** မှာ **php 8.2** ကိုပဲ **support** ပေးသေးတာမို့ နောက်ပိုင်းမှာ **Filament install** လုပ်ရင် **error** တွေရှိနိုင်လို့ပါ။ **Laravel Herd** မှာတော့ **PHP version** ကိုကြိုက်သလောက်ထည့်သုံးနိုင်ပြီး **PHP version** အတိုးအလျှော့ပါလုပ်နိုင်ပါတယ်။

ဒါပေမယ့် **MySQL server** နဲ့ **PHPMyAdmin** တို့သုံးချင်ရင် **Laravel Herd** ကို **Pro version** ဝယ်သုံးဖို့လိုတာမို့ **XAMPP** နဲ့ပါတွဲသုံးရတာဖြစ်ပါတယ်။

အရင်ဆုံး **Laravel Herd** ကို **install** လုပ်ပါမယ်။ **herd.laravel.com** ကိုသွားပါ



Download for Windows ကိုနှိပ်ပါ။ **Download** လုပ်လို့ရလာတဲ့ **installer file** ကိုဖွင့်ပြီး **install** လုပ်ပါ။ ပြီးသွားရင် **Finish** ခလုတ်ကိုနှိပ်ပါ။

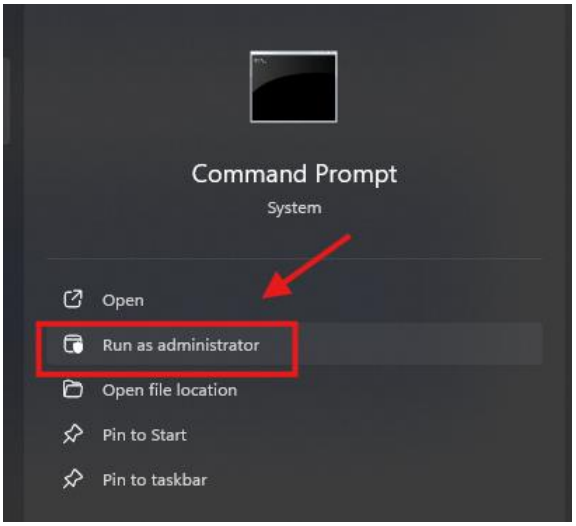
Windows Key နှိပ်ပြီး **Herd** လို့ရိုက်ထည့်ပါ။ **Herd application** ပေါ်လာရင် **click** နှိပ်ပြီးဖွင့်ပါ။

Do you want to allow public and private network မေးလာရင် **Allow** ကိုနှိပ်ပါ။ **local network** ကို **access** ပေးဖို့လိုတာကြောင့်ပါ။ ပြီးရင် **Let's get started** ကိုနှိပ်ပါ။ **Yes** နှိပ်လိုက်ရင် **powershell** ပွင့်လာပြီး **install** ဆက်လုပ်သွားပါလိမ့်မယ်။

Herd Pro သုံးမလားမေးရင် **Skip for now** ကိုနှိပ်ပါ။ ဒါဆိုရင် **Herd** ကိုပိတ်လို့ရပါပြီ။

2.2 Create Laravel Project

Windows Key နှိပ်ပြီး **cmd** လို့ရိုက်ထည့်ပါ။ **Command Prompt** ပေါ်လာရင် သူ့အောက်မှာ **Run as administrator** ကိုနှိပ်ပါ



Command Prompt ပွင့်ပွင့်ချင်းမှာ လက်ရှိရောက်နေတဲ့ **directory** က **C:\Windows\System32** ဖြစ်ပါတယ်။ အဲဒီကနေ မိမိ **Profile** အောက်က **Herd folder** ကိုသွားဖို့လိုပါတယ်။ **Users folder** ကိုသွားဖို့အတွက် **change directory command** ဖြစ်တဲ့ **cd** ကိုသုံးပြီးအောက်ပါအတိုင်းရိုက်ထည့်ပါ။ **Enter** နှိပ်ပါ။

```
cd C:/Users
```

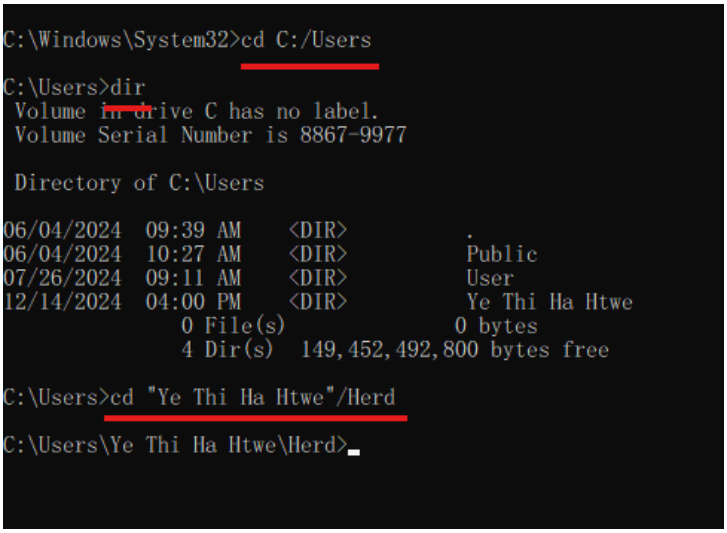
Users folder ထဲကို **list** လုပ်ကြည့်ဖို့ **dir command** ကို **run** ပါ။

```
dir
```

အဲဒီမှာ **Public** နဲ့ **User** မဟုတ်တဲ့ ကျန်တဲ့ **folder** ဟာ မိမိ **profile folder** ဖြစ်ပါတယ်။ အဲဒီအထဲမှာ **Herd folder** ရှိပါတယ်။ တစ်ခါတည်း **change directory** လုပ်ပါမယ်။

```
cd "<မိမိ Username>"/Herd
```

command တွေကိုတဆင့်ချင်း **run** သွားတဲ့ပုံစံက ပုံမှာမြင်ရတဲ့အတိုင်းဖြစ်ပါတယ်။



Herd folder ထဲရောက်သွားရင် **laravel project create** လုပ်တဲ့ **command** ကို **run** လို့ရပါပြီ။

```
laravel new pharmacy-training-app
```

Authentication အတွက် ဘာ **method** သုံးမလဲမေးပါလိမ့်မယ်။ နောက်ပိုင်းမှာ **Filament** ရဲ့ **built-in authentication** သုံးမှာဖြစ်တဲ့အတွက် လောလောဆယ်ဘာမှသုံးစရာမလိုပါဘူး။ **No starter kit** အတွက် **none** ကိုရွေးပါ။

Test မေးလာရင်ဘာမှမထည့်ဘဲ **enter** နှိပ်ပါ။

Database မေးလာရင် **mysql** ထည့်ပေးပါ။

default database migration လုပ်မလားမေးရင် **no** ထည့်ပေးပါ။ ကျွန်တော်တို့ **mysql database** မဆောက်ရသေးတာကြောင့် **migration** လုပ်လို့မရသေးလို့ပါ။

ဒါပြီးသွားရင် **project setup** ကိုခဏရပ်ထားပြီး **XAMPP install** လုပ်တဲ့အပိုင်းကိုဆက်သွားပါမယ်။ **Command Prompt window** ကိုမပိတ်ဘဲခဏထားထားပါ။

2.3 Install XAMPP on Windows

XAMPP ဆိုတာ ဝက်ဘ်ဆိုက်များနှင့် ဝက်ဘ်အပ်ပလီကေးရှင်းများ ဖန်တီးရေးသားရန်အတွက် အခမဲ့ဆော့ဖ်ဝဲတစ်ခုဖြစ်ပါတယ်။ ဝက်ဘ်ဆာဗာနှင့် ဒေတာဘေ့စ်ဆာဗာတွေကို စုစည်းပေးထားသည့် ဆော့ဖ်ဝဲတစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့်အွန်လိုင်းပေါ်မှာ ငွေကုန်ကြေးကျခံပြီးဆာဗာတွေ ဝယ်စရာမလိုဘဲ ကိုယ့်ကွန်ပြူတာထဲမှာပဲ **XAMPP** နဲ့ ဆာဗာနည်းပညာတွေအသုံးပြုနိုင်ပါတယ်။

- **X: Cross-platform** (စနစ်အမျိုးမျိုးတွင် အသုံးပြုနိုင်ခြင်း)
- **A: Apache HTTP Server** (အပါချီဝက်ဘ်ဆာဗာ)
- **M: MariaDB** (ဒေတာဘေ့စ်)
- **P: PHP** (ပရိုဂရမ်မင်းဘာသာစကား)
- **P: Perl** (ပရိုဂရမ်မင်းဘာသာစကား)

XAMPP ရဲ့ဝက်ဘ်ဆိုက်ဖြစ်တဲ့ **https://www.apachefriends.org** ကိုသွားပါ။

Windows အတွက် XAMPP နောက်ဆုံးဗားရှင်းကို ဒေါင်းလုဒ်လုပ်ပါ။

ဒေါင်းလုဒ်လုပ်ပြီးတဲ့ installer ဖိုင်ကို ဖွင့်ပါ။

User Account Control ခွင့်ပြုချက်တောင်းလျှင် "Yes" ကိုနှိပ်ပါ။

Setup wizard ပေါ်လာပါမည်။ "Next" ကိုနှိပ်ပါ။

XAMPP တွင်ပါဝင်သော components များကို ရွေးချယ်ပါ။ အားလုံးကို default အတိုင်းထားခဲ့နိုင်ပါတယ်။ "Next" ကိုနှိပ်ပါ။

XAMPP ကို install လုပ်မည့် နေရာကို ရွေးပါ။ ပုံမှန်အားဖြင့် C:\xampp ဖြစ်ပါတယ်။ "Next" ကိုနှိပ်ပါ။

Bitnami အကြောင်းစာမျက်နှာတွင် အမှန်ခြစ်ကို ဖြုတ်ထားပါ။ "Next" ကိုနှိပ်ပါ။

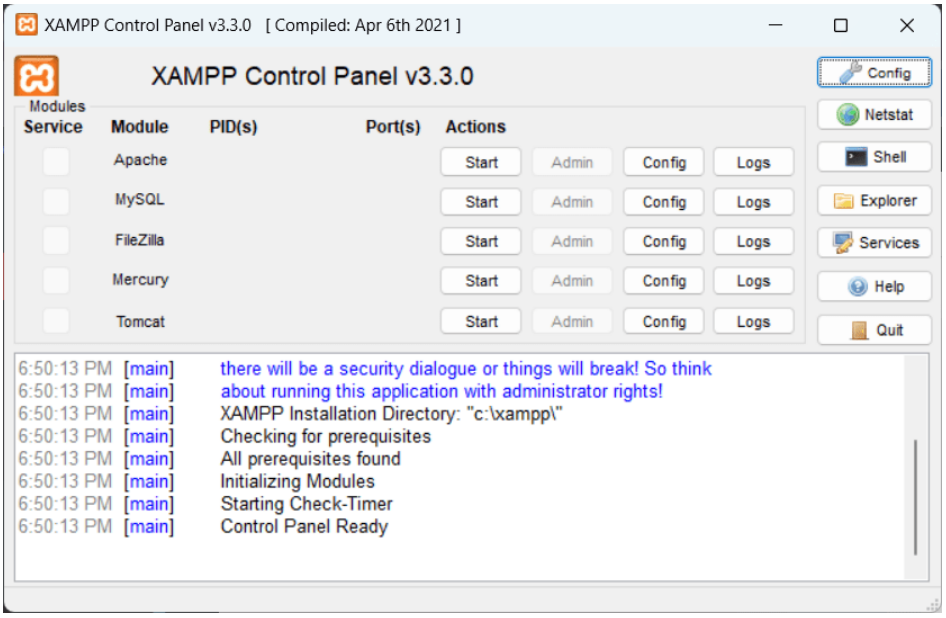
"Next" ကိုနှိပ်၍ installation စတင်ပါ။

Firewall သတိပေးချက် ပေါ်လာပါက သင့်လျော်သလို ရွေးချယ်ပါ။ private network များအတွက် access ခွင့်ပြုပေးထားရန်လိုအပ်ပါတယ်။

Installation ပြီးဆုံးသွားပါက "Finish" ကိုနှိပ်ပါ။

XAMPP Control Panel ကို စတင်ရန် မေးလာပါက "Yes" ကိုရွေးပါ။

XAMPP Control Panel တွင် Apache နှင့် MySQL တို့၏ "Start" ခလုတ်များကို နှိပ်၍ server များ စတင်နိုင်ပါပြီ။



XAMPP သွင်းပြီးရင် Node.JS နဲ့ Composer တို့ကိုဆက်သွင်းရပါမယ်။

2.4 Install Node.js on Windows

Node.js ဟာ JavaScript runtime environment တစ်ခုဖြစ်ပါတယ်။ Laravel က php ၊ Node က JavaScript တခြားစီဖြစ်ပေမယ့် Laravel project ရဲ့ frontend အပိုင်းကို Node.js သုံးပြီးစီမံခန့်ခွဲဖို့လိုပါတယ်။ Laravel မှာ package manager ဖြစ်တဲ့ composer ရှိသလိုပဲ Node.js မှာ npm (Node Package Manager) ရှိပါတယ်။ Laravel project ရဲ့ frontend CSS ၊ JavaScript တွေကို compile လုပ်ဖို့အတွက် Laravel Mix (webpack wrapper) က Node.js ကိုသုံးရပါတယ်။ Laravel ဟာ server side language ဖြစ်တဲ့ PHP ကိုအခြေခံထားတဲ့ backend framework မို့လို့ front-end အတွက် Vue.js, React စတာတွေနဲ့တွဲဖက်အသုံးပြုမယ်ဆိုရင် Node.js ရှိဖို့လိုပါတယ်။

၁။ Node.js တရားဝင်ဝက်ဘ်ဆိုက် (https://nodejs.org) သို့သွားပါ။

၂။ LTS (Long Term Support) ဗားရှင်းကို ဒေါင်းလုဒ်လုပ်ပါ။

၃။ ဒေါင်းလုဒ်ပြီးတဲ့ ဖိုင်ကို ဖွင့်ပါ။

၄။ **Install wizard** ပေါ်လာပါမည်။ "**Next**" ကိုနှိပ်ပါ။

၅။ **License agreement** ကို လက်ခံပြီး "**Next**" ကိုနှိပ်ပါ။

၆။ ထည့်သွင်းမည့် နေရာကို ရွေးချယ်ပါ။ **default** အတိုင်းထားနိုင်ပါတယ်။ "**Next**" ကိုနှိပ်ပါ။

၇။ **Features** များကို ရွေးချယ်ပါ။ ပုံမှန်အားဖြင့် **default** အတိုင်းထားခဲ့နိုင်ပါတယ်။ "**Next**" ကိုနှိပ်ပါ။

၈။ "**Install**" ကိုနှိပ်၍ ထည့်သွင်းခြင်းကို စတင်ပါ။

၉။ ပြီးဆုံးပါက "**Finish**" ကိုနှိပ်ပါ။

၁၀။ **Command Prompt** ဖွင့်၍ **node -v** နှင့် **npm -v** ဟုရိုက်ပြီး **Node.js** နှင့် **npm** ကို အောင်မြင်စွာ ထည့်သွင်းပြီးကြောင်း စစ်ဆေးပါ။

2.5 Install Composer on Windows

Composer ဟာ **PHP** အတွက် **package manager** တစ်ခုဖြစ်ပါတယ်။ **Laravel framework** ဟာလည်း **Composer** ပေါ်မှာအခြေခံထားတာဖြစ်ပါတယ်။ **Laravel project** တစ်ခု စတင်တည်ဆောက်ဖို့လိုတဲ့ **package** တွေကို **Composer** ကတဆင့် **install** လုပ်ရပါတယ်။ သက်ဆိုင်ရာ **website** တွေမှာတခုချင်း **download** လုပ်ပြီး **project** ထဲကိုသွားထည့်ရတာထက် **package manager** ကနေဆွဲသွင်းလိုက်တာက အချိန်ကုန်သက်သာစေမှာဖြစ်ပါတယ်။

Composer ဝက်ဘ်ဆိုက် (**https://getcomposer.org/download/**) သို့သွားပါ။

"**Composer-Setup.exe**" ကို ဒေါင်းလုဒ်လုပ်ပါ။ ဒေါင်းလုဒ်ထားတဲ့ **.exe** ဖိုင်ကို ဖွင့်ပါ။

Developer mode ကို အမှန်ခြစ်စရာမလိုပါဘူး။ "**Next**" ကိုနှိပ်ပါ။

PHP ဖိုင်တည်နေရာမှာ **C:\xampp\php\php.exe** ဖြစ်နိုင်ပါတယ်။ **Add this PHP to your path?** ကိုအမှန်ခြစ်ပါ။ **Next** နှိပ်ပါ။

proxy server settings မှာ ထည့်ဖို့မလိုပါဘူး။ "**Next**" ကိုနှိပ်ပါ။

"**Install**" ကိုနှိပ်ပါ။

ပြီးဆုံးပါက "**Finish**" ကိုနှိပ်ပါ။

Command Prompt ဖွင့်၍ **composer --version** ဟုရိုက်ပြီး **Composer** ကို အောင်မြင်စွာ ထည့်သွင်းပြီးကြောင်း စစ်ဆေးပါ။

2.6 Install Visual Studio Code on Windows

Visual Studio Code ဟာ **Microsoft** က ထုတ်လုပ်ထားတဲ့ **source code editor** တစ်ခုဖြစ်ပြီး **developer** တွေအတွက် အခမဲ့အသုံးပြုနိုင်တဲ့ **software** တစ်ခုဖြစ်ပါတယ်။ **Windows, macOS** နဲ့ **Linux** စတဲ့ **operating system** အမျိုးမျိုးမှာ အသုံးပြုနိုင်ပါတယ်။ **Integrated terminal** ပါဝင်တာကြောင့် **editor** ထဲကနေပဲ **command line** တွေကို **run** နိုင်ပါတယ်။ **feature** ပေါင်းစုံပါဝင်ပြီး **performance** ကောင်းမွန်တာကြောင့် **beginner** တွေအတွက်ရော **professional developer** တွေအတွက်ပါ သင့်တော်တဲ့ **code editor** တစ်ခုဖြစ်ပါတယ်။

Visual Studio Code ကို **Windows** ကွန်ပြူတာမှာ **install** လုပ်ဖို့

https://code.visualstudio.com သို့သွားပါ

"**Download for Windows**" ခလုတ်ကို နှိပ်ပါ။ **Windows** ဗားရှင်း (**User Installer 64 bit**) ကို ရွေးချယ်ပါ။

Download လုပ်ထားတဲ့ **VSCodeUserSetup-x64.exe** ဖိုင်ကို **double-click** နှိပ်ပါ။ "**I accept the agreement**" ကို ရွေးပြီး **Next** ကိုနှိပ်ပါ။

install လုပ်ချင်တဲ့ နေရာကို ရွေးချယ်ပါ။ **default** အတိုင်းထားလို့လည်းရပါတယ်။

Additional Tasks တွေကို အမှန်ခြစ်ပါ (**recommended options** တွေကို အကုန်အမှန်ခြစ်နိုင်ပါတယ်)။ အောက်ကအချက်တွေကိုတော့ အမှန်ခြစ်ထားသင့်ပါတယ်

"Add 'Open with Code' action to Windows Explorer file context menu"

"Add 'Open with Code' action to Windows Explorer directory context menu"

"Register Code as an editor for supported file types"

"Add to PATH"

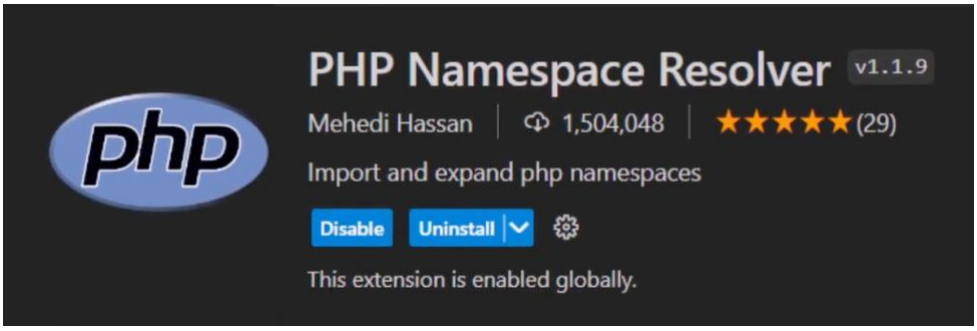
ဒီ options တွေရွေးထားခြင်းအားဖြင့် file တွေကို right-click လုပ်ပြီး VS Code နဲ့ တိုက်ရိုက်ဖွင့်နိုင်သလို၊ command prompt ကနေလည်း VS Code ကို run နိုင်မှာ ဖြစ်ပါတယ်။

Install ကိုနှိပ်ပါ။ Installation ပြီးဆုံးရင် Finish ကိုနှိပ်ပါ

Installation ပြီးရင် PHP namespace resolver extension ထည့်သွင်းဖို့အတွက် အောက်ပါအတိုင်း လုပ်ဆောင်ပါ။

Start menu ကနေ Visual Studio Code ကို ရိုက်ထည့်ပြီးဖွင့်ပါ။

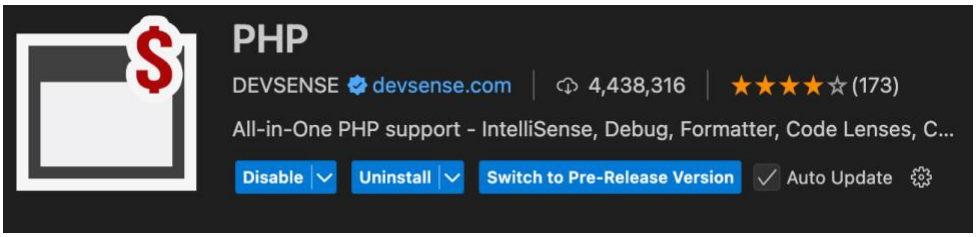
ဘယ်ဘက်အစွန်းမှာရှိတဲ့ Extensions icon (puzzle piece ပုံစံ) ကို နှိပ်ပါ။ ဒါမှမဟုတ် Ctrl+Shift+X ကို နှိပ်လို့လည်း ရပါတယ်။ Extensions marketplace ပွင့်လာတဲ့အခါ search box မှာ "PHP namespace resolver" လို့ရိုက်ရှာပါ။



PHP Namespace Resolver extension ကို Install ခလုတ်နှိပ်ပြီး ထည့်သွင်းနိုင်ပါတယ်။ installation ပြီးဆုံးသွားရင် Visual Studio Code ကို ပိတ်လို့ရပါပြီ။

PHP namespace resolver သွင်းထားခြင်းအားဖြင့် PHP ဖိုင်တစ်ခုဖွင့်ထားတဲ့အခါ Ctrl+Alt+I ကို နှိပ်ခြင်းဖြင့် လိုအပ်တဲ့ namespace တွေကို auto-import လုပ်ပေးမှာဖြစ်ပါတယ်။ အကယ်၍ settings ထဲက auto-import feature ကို enable လုပ်ထားရင်တော့ PHP class တွေ၊ interface တွေ ရေးတဲ့အခါ သက်ဆိုင်ရာ namespace တွေကို အလိုအလျောက် import လုပ်ပေးသွားမှာ ဖြစ်ပါတယ်။

နောက်တစ်ခုက PHP extension သွင်းပေးပါ။ Icon ကပုံမှာမြင်ရတဲ့အတိုင်းဖြစ်ပါတယ်။ php code တွေကို format လုပ်ပေးဖို့ရယ်၊ auto-suggest လုပ်ပေးဖို့ရယ်အတွက်အသုံးဝင်ပါတယ်။



Extension ထည့်ပြီးရင် VSCode Window ကိုပိတ်လို့ရပါပြီ။

Laravel မှာ namespace ဆိုတာ class တွေကို organize လုပ်တဲ့စနစ်တစ်ခုဖြစ်ပါတယ်။ PHP အခြေခံ feature တစ်ခုဖြစ်ပြီး class တွေ၊ interface တွေ၊ function တွေနဲ့ constant တွေကို logical group အလိုက် စုစည်းထားတဲ့ directory system တစ်ခုလို့ပြောလို့ရပါတယ်။

Namespace တွေကို အသုံးပြုခြင်းအားဖြင့် name conflict တွေကို ရှောင်ရှားနိုင်ပါတယ်။ ဥပမာ - project ထဲမှာ Controller ဆိုတဲ့ class နှစ်ခု ရှိနေတယ်ဆိုပါစို့။ namespace မသုံးရင် PHP က ဘယ် Controller ကိုခေါ်မှန်းမသိဘဲ error တက်သွားမှာဖြစ်ပါတယ်။ ဒါပေမယ့် namespace သုံးထားမယ်ဆိုရင် App\Http\Controllers\UserController နဲ့ App\Http\Controllers\Admin\UserController ဆိုပြီး သီးခြားစီ ခွဲခြားသတ်မှတ်နိုင်တာကြောင့် name conflict ဖြစ်စရာမရှိတော့ပါဘူး။

Laravel မှာ namespace တွေကို PSR-4 autoloading စံသတ်မှတ်ချက်အတိုင်း directory structure နဲ့ တိုက်ရိုက်ဆက်စပ်ထားပါတယ်။ ဥပမာအားဖြင့် app/Models directory အောက်မှာရှိတဲ့ User.php ဆိုတဲ့ class ရဲ့ namespace က App\Models ဖြစ်ပါတယ်။

Namespace တွေကို use keyword နဲ့ import လုပ်ပြီးသုံးနိုင်ပါတယ်။ ဥပမာ - use App\Models\User ဆိုပြီး import လုပ်ထားမယ်ဆိုရင် User class ခေါ်သုံးတိုင်း \App\Models\User လို့မရေးတော့ဘဲ User ဆိုပြီး တိုက်ရိုက်သုံးလို့ရပါပြီ။ Class တွေကို organize လုပ်တဲ့အပြင် code တွေကို ပိုပြီး readable ဖြစ်စေတဲ့အတွက် Laravel project တွေမှာ namespace တွေကို အသုံးများကြပါတယ်။

3 Project Setup

Laravel Herd သွင်းတုန်းက မပိတ်ဘဲထားထားတဲ့ Command Prompt Window ဆီပြန်လာပါ။ project folder ထဲကို change directory လုပ်ပါ။

```
cd pharmacy-training-app
```

Project directory ထဲရောက်ပြီဆိုရင် visual studio code ဖွင့်ဖို့အတွက် code . ကို run ပါ။ Visual Studio Code ဟာ လက်ရှိအလုပ်လုပ်နေတဲ့ folder မှာပွင့်လာပါလိမ့်မယ်။ Trust the author မေးလာရင်နှိပ်ပေးပါ။

Visual Studio Code ပွင့်လာရင် အောက်ပါ folder တွေ တွေ့ရပါမယ်

- app/ - Models, Controllers စတဲ့ core logic တွေရှိပါတယ်
- database/ - Migrations နဲ့ seeds တွေရှိပါမယ်
- resources/ - Views နဲ့ assets
- routes/ - URL routes တွေ
- public/ - Static files တွေ

လက်ရှိ folder ထဲမှာ .env ဖိုင်ကိုဖွင့်ပါ။ တချို့တွေမှာ .env ဖိုင်မရှိသေးရင် .env.example ကို copy လုပ်ပြီး .env နာမည်နဲ့ save ပါ။

3.1 MVC (Model-View-Controller)

MVC ဆိုတာ ဆော့ဖ်ဝဲတည်ဆောက်မှုပုံစံ (Software Architecture Pattern) တစ်ခုဖြစ်ပြီး application တစ်ခုကို အပိုင်း ၃ ပိုင်းခွဲပြီး ရေးသားတဲ့ပုံစံဖြစ်ပါတယ်။ MVC အကြောင်းကိုစာနဲ့ဖတ်ရတာက ပိုခက်ပြီးတကယ်လုပ်ကြည့်လိုက်မှ မျက်စိထဲချက်ချင်းမြင်လာတတ်တာမျိုးပါ။

Model

Data နဲ့ Business Logic တွေကို ကိုင်တွယ်ပြီး Database နဲ့ ချိတ်ဆက်ခြင်း၊ data validation လုပ်ခြင်းစတဲ့ လုပ်ငန်းတွေ ပါဝင်ပါတယ်။ View နဲ့ Controller တို့နဲ့ တိုက်ရိုက်ဆက်သွယ်မှု မရှိပါဘူး

View

User Interface ကို ဖော်ပြပေးတဲ့အပိုင်း ဖြစ်ပါတယ်။ User တွေမြင်ရတဲ့ အပိုင်းအားလုံး (buttons, forms, tables စသည်) ပါဝင်ပါတယ်။ Model ထဲက data တွေကို ပြသပေးပါတယ်

Controller

Model နဲ့ View ကို ချိတ်ဆက်ပေးတဲ့ ကြားခံအပိုင်း ဖြစ်ပါတယ်။ User input တွေကို လက်ခံပြီး လိုအပ်သလို process လုပ်ပါတယ်။ Model ကနေ data တွေယူပြီး View မှာ ပြသဖို့ စီမံပေးပါတယ်

ဥပမာ Facebook post တစ်ခုမှာ post ရဲ့စာသား likes ၊ comments စတဲ့ data တွေကို database နဲ့ ချိတ်ဆက်ထားတဲ့အပိုင်းကို Model ကတာဝန်ယူပါတယ်။ post ကို newsfeed မှာ ပြသပေးတဲ့အပိုင်းကို View ကတာဝန်ယူပါတယ်။ like နှိပ်လိုက်ရင် Model ထဲက data ကို update လုပ်ပြီး View မှာ ပြန်ပြပေးတဲ့အပိုင်းမျိုးကို Controller ကတာဝန်ယူပါတယ်။

MVC pattern ကို Laravel, Ruby on Rails, ASP.NET MVC စတဲ့ framework တွေမှာ အသုံးများပါတယ်။

3.2 .env

DB_CONNECTION ကနေ **DB_PASSWORD=** အထိ အောက်ပါအတိုင်းပြောင်းပေးပါ

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=pharmacy_training_app
DB_USERNAME=root
DB_PASSWORD=
```

DB_CONNECTION မှာ **database** အမျိုးအစားထည့်ပေးရပါတယ်

DB_DATABASE database နာမည်ထည့်ပေးရပါမယ်

DB_USERNAME database user name

DB_PASSWORD database password

လောလောဆယ် **database** မဆောက်ရသေးပေမယ့် တန်ဖိုးလေးတွေအရင်ထည့်ထားလိုက်လို့ရပါတယ်။

.env ဖိုင်ကို **save** ပြီးပိတ်လို့ရပါပြီ။

အခုနောက်ပိုင်းကစပြီး **Visual Studio Code** ရဲ့ **Terminal** ကနေပဲ **command** တွေ **run** ပါတော့မယ်။ **Terminal** ထဲက **New Terminal** ကိုရွေးပါ။

Visual Studio Code ရဲ့ **Terminal** က **window** ပြောင်းစရာမလိုဘဲ **code** ရေးရင်း **terminal command** တွေကို လွယ်လွယ်ကူကူ **run** နိုင်ပါတယ်။ **Split terminal** တွေခွဲပြီး သုံးနိုင်ခြင်း၊ **Command history** ကို **arrow key** နဲ့ လွယ်လွယ်ကူကူ ရှာနိုင်ခြင်း၊ **Copy/paste** လုပ်ရတာ ပိုလွယ်ကူခြင်း၊ **Syntax highlighting** ပါဝင်ခြင်း စတဲ့ အဆင့်မြင့် **terminal feature** တွေ ပါဝင်ပါတယ်

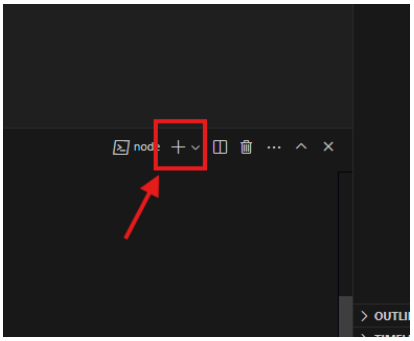
3.3 NPM

Node Package Manager (NPM) သွင်းရပါမယ်။ **NPM run** ဖို့အတွက် အစောပိုင်းမှာပါခဲ့တဲ့ **Node.JS install** လုပ်ထားရပါမယ်။

```
npm install

npm run dev
```

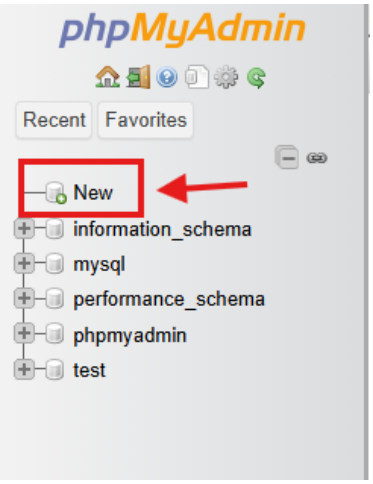
ကျွန်တော်တို့ရဲ့ **project** ဟာ **pharmacy-training-app.test** လိပ်စာနဲ့ **browser** မှာသွားဖွင့်လို့ရပါပြီ။

npm run dev နောက်မှာ ဒီ **terminal tab** ကိုဆက်သုံးလို့မရတော့ပါဘူး။ **Visual Studio Code terminal** အပေါ်နားကအပေါင်းလက္ခဏာကိုနှိပ်ပြီး **new terminal** ဖွင့်ပါ။

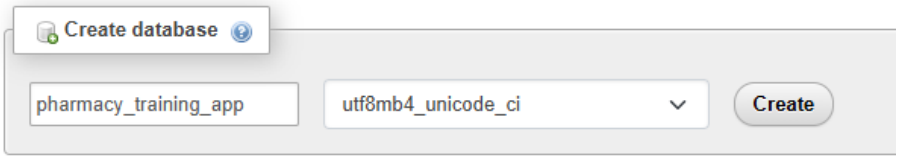


3.4 Create database

Database ဆောက်တဲ့အပိုင်းကိုဆက်သွားပါမယ်။ MySQL Database တည်ဆောက်ဖို့ http://localhost/phpmyadmin ကို browser မှာဖွင့်ပါ။ New ခလုတ်ကိုနှိပ်ပြီး Database အသစ်တစ်ခုပြုလုပ်ပါ။



"pharmacy_training_app" နာမည်နဲ့ create လုပ်ပါ။ Character set: utf8mb4_unicode_ci ရွေးပါ။ ဒီနေရာမှာသတိပြုရမှာက pharmacy ၊ training ၊ app တို့ကြားမှာ underscore _ ဖြစ်ပါတယ်။ dash - နဲ့ဖြစ်ဖြစ်၊ space နဲ့ဖြစ်ဖြစ်ခြားလို့မရပါဘူး။



MySQL မှာ space သုံးလို့မရပါဘူး။ ဒါကြောင့် စာလုံးတွေကို ခွဲခြားဖို့ underscore ကို သုံးပါတယ်။ ဥပမာ - "user_profile" လို့ရေးရင် user နဲ့ profile ကို ခွဲခြားပြီး ဖတ်ရလွယ်ပါတယ်။

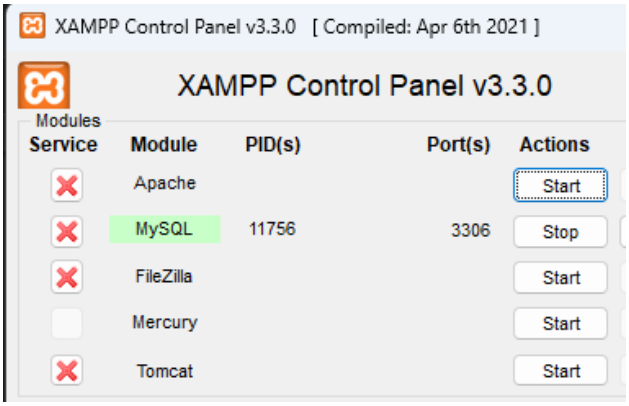
MySQL က အမည်တွေမှာ စာလုံးအကြီး၊ အသေး ခွဲခြားမှုမရှိပါဘူး။ ဒါကြောင့် "UserProfile" လို့ရေးရင် "userprofile" လို့ပဲ ဖတ်ပါတယ်။ underscore သုံးခြင်းဖြင့် စာလုံးတွေကို ပိုရှင်းအောင် ခွဲခြားနိုင်ပါတယ်။

underscore မဟုတ်ဘဲ dash - နဲ့ခံလို့လည်းရပါတယ်။ ဒါပေမယ့် SELECT * FROM user-profile လိုမျိုး SQL query တွေရေးတဲ့အခါ user minus profile လို့ အဓိပ္ပါယ်ကောက်လွဲပြီး error တွေပေါ်တတ်တဲ့အတွက် SELECT * FROM 'user-profile' လို့ quote တွေထည့်ရေးရပါတယ်။

phpMyAdmin မှာလုပ်စရာကဒီလောက်ပါပဲ။ database user က default အားဖြင့် root ဖြစ်ပြီး password blank မို့လို့ .env ဖိုင်မှာ အဲဒီအတိုင်းပေးထားခဲ့တာဖြစ်ပါတယ်။ database ဆောက်ပြီးရင် table တွေဆောက်ဖို့ အောက်ပါ command ကို run ပါ။ VSCode မှာ စောစောတုန်းက အသစ်ယူထားခဲ့တဲ့ terminal tab မှာ run မှာဖြစ်ပါတယ်။

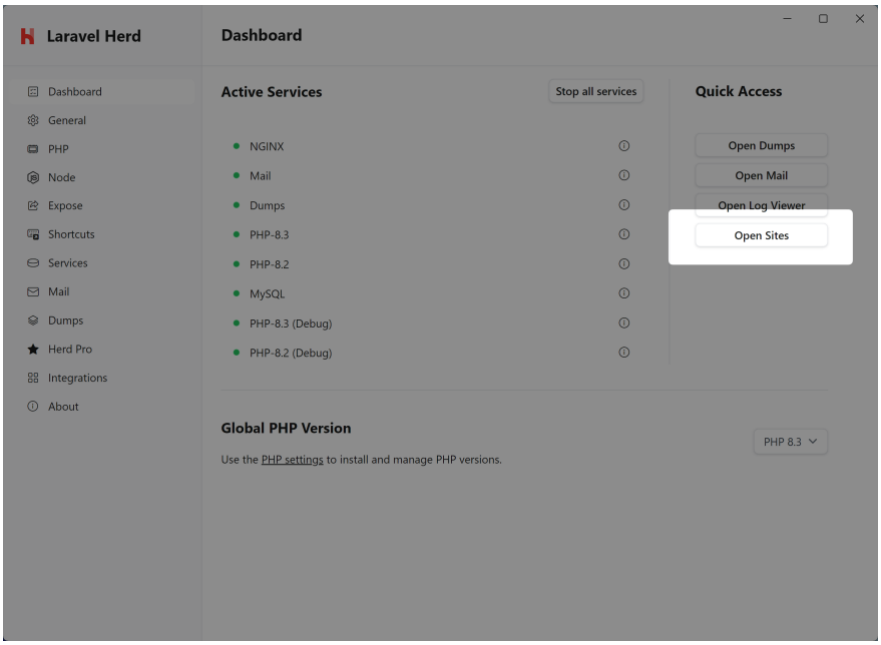
```
php artisan migrate
```

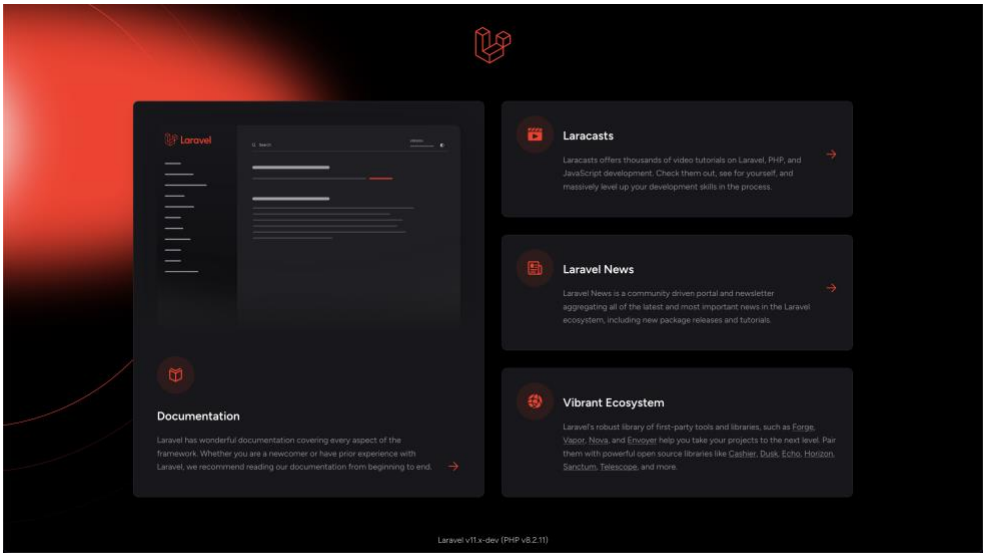
users table အပါအဝင် default table တွေဆောက်သွားတာ terminal မှာတွေ့ရပါလိမ့်မယ်။ ဒီအပိုင်းပြီးရင် XAMPP control panel ကိုပြန်လာပြီး Apache server ကို Stop နှိပ်ပြီးပိတ်လို့ရပါပြီ။ Laravel Herd လည်း run နေပြီမို့လို့ phpMyAdmin မဖွင့်တော့ဘူးဆိုရင် Apache stop လုပ်လို့ရပါတယ်။



3.5 PHP Development Server

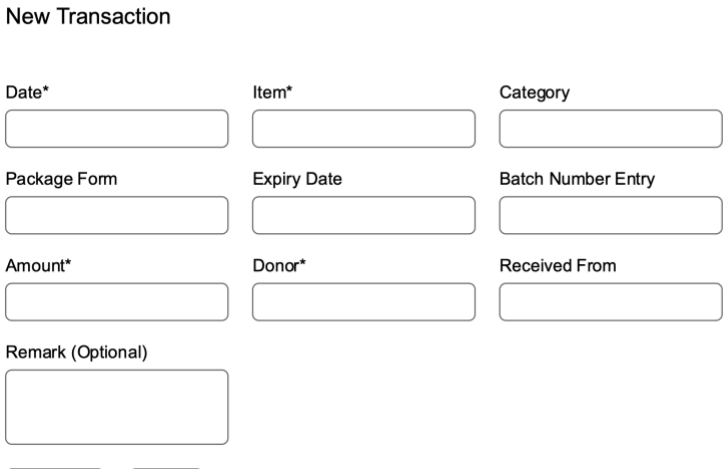
Laravel Herd နဲ့ဆိုရင် php development server သပ်သပ် run စရာမလိုတော့ပါဘူး။ Browser မှာ pharmacy-training-app.test ကို ဖွင့်ကြည့်ပါ။ Laravel welcome page ပေါ်လာရပါမယ်။ ဒီစာရေးနေတဲ့အချိန်မှာ Laravel version 11 ရှိပါပြီ။ ဒီလိပ်စာကဘယ်ကရသလဲဆိုရင် Laravel Herd က generate လုပ်ပေးပါတယ်။ Laravel Herd window ကို Task bar ကနေပြန်ခေါ်ကြည့်လို့ရပါတယ်။ Open Sites ထဲကနေ လက်ရှိ project လိပ်စာကိုသွားဖွင့်လို့ရပါတယ်။





4 Transaction Wireframe

ဒါကတော့ **application** မှာ **data** အသွင်းအထုတ်အများဆုံးပြုလုပ်မယ့် **transaction** စာမျက်နှာပုံကြမ်းပါ။ ဆေးပစ္စည်းအဝင်မှတ်တမ်းတွေကို စာရင်းသွင်းမှာဖြစ်ပါတယ်။



Wireframe ဆိုတာ **website** တွေ၊ **mobile app** တွေကို ပုံကြမ်းဆွဲထားတာပါ။ ဒီဇိုင်းအချောမဆွဲခင်ပုံကြမ်းအရင်ဆွဲလေ့ရှိကြပါတယ်။ **wireframe** ကိုအရောင်တွေ၊ ဓါတ်ပုံတွေမပါဘဲ မျဉ်းကြောင်းတွေနဲ့ပဲဆွဲပါတယ်။

Wireframe ဆွဲခြင်းအားဖြင့် ဘယ်နေရာမှာဘာထားချင်တယ်၊ **data flow** ကဘယ်လိုသွားမယ်ဆိုတာမြင်သာစေပါတယ်။ **wireframe** ဆွဲလိုက်တဲ့အခါ သူများကိုနားလည်အောင်ရှင်းပြဖို့ထက် ကိုယ့်ဘာသာကိုယ်မျက်စိထဲရှင်းသွားပြီး **database table** တွေကိုပါ စောစောစီးစီး **schema** ထိုင်ထားလို့ရပါတယ်။ မဟုတ်လို့ **production** အဆင့်ရောက်ပြီးမှ **database schema** လိုက်ပြင်ရမတာမလွယ်ပါဘူး

Wireframe ကို **Figma, Adobe XD, Sketch** စတဲ့ **software** တွေနဲ့ ဆွဲနိုင်သလို၊ စက္ကူပေါ်မှာလည်း လက်နဲ့ရေးဆွဲနိုင်ပါတယ်။

Date နေရာမှာ **date picker** ထည့်ပါမယ်။ လိုချင်တဲ့ **date** ကို တစ်ခုချင်းရိုက်ထည့်မနေဘဲ ပြက္ခဒိန်မှာ ရက်စွဲတန်းရွေးလိုက်လို့ရအောင်ပါ။

Item မှာ **dropdown** လုပ်ပါမယ်။ စာရင်းသွင်းချင်တဲ့ပစ္စည်းနာမည်ကို အစအဆုံးရိုက်ထည့်မနေဘဲ ရှိပြီးသား **list** ထဲကရွေးလို့ရအောင်ဖြစ်ပါတယ်။ မဟုတ်ရင် ပစ္စည်းကတစ်မျိုးတည်း၊ စာလုံးပေါင်းမှားလို့ ပစ္စည်းနှစ်မျိုးအနေနဲ့ စာရင်းဝင်နေမှာစိုးလို့ပါ။

Category ကိုလည်း **dropdown** နဲ့ရွေးလို့ရအောင်လုပ်ပါမယ်။

Expiry date ကိုလည်း **date** အတိုင်းပဲ **date picker** ထည့်ပါမယ်။

Batch number ကတော့အလွတ်အတိုင်းရိုက်ထည့်လို့ရမယ့် **Text Field** ဖြစ်ပါမယ်။

Amount လည်းအလွတ်ရိုက်ထည့်လို့ရမယ့် **number field** ဖြစ်ပါမယ်။

Donor က ရွေးရမယ့် **dropdown** လုပ်ပေးပါမယ်။

Recieved from က ဘယ်ကနေရသလဲဆိုတဲ့ **data** ထည့်ဖို့ပါ။ **dropdown** ဖြစ်ပါမယ်။

Remark က မှတ်ချက်ထည့်ဖို့အတွက် အလွတ်ရိုက်ထည့်လို့ရမယ့် **Textarea** ဖြစ်ပါမယ်။

ဒီ **Form** မှာ **data** ဖြည့်လိုက်ပြီဆိုတာနဲ့ **database** ထဲသိမ်းသွားမယ်၊ သိမ်းထားတဲ့ **data** တွေကို **table** နဲ့ပြန်ပြပေးဖို့လိုပါတယ်။ အဲဒီတော့ ဒေတာတွေသိမ်းလို့ရအောင် **database table** တစ်ခုဆောက်ပေးရပါမယ်။

Laravel မှာ **database table** ဆောက်တာ၊ ပြင်ဆင်တာကို **migrate** လုပ်တယ်လို့ခေါ်ပါတယ်

5 Create models and migrations

Model နဲ့ **Migration** တွေမတည်ဆောက်ခင် သူတို့အကြောင်းသိထားဖို့လိုပါတယ်

5.1 Migrations

Migration ဆိုတာ **database** တည်ဆောက်ပုံကို ထိန်းချုပ်နိုင်တဲ့ လုပ်ငန်းတစ်ခုပါ။ ပုံမှန်ဆိုရင် **RAW SQL command** တွေနဲ့ **database table** တွေတည်ဆောက်ရပေမယ့် **Laravel Migration** မှာ **code** ရေးရုံနဲ့တင် **Database table** တွေတည်ဆောက်နိုင်ပါတယ်။ **Migration** ကို **rollback** လုပ်နိုင်တာကြောင့် **MySQL cli** ဖြစ်ဖြစ်၊ **phpMyAdmin** ဖြစ်ဖြစ်သွားပြီး **table** တွေသွားဖျက်တာ၊ **column** တွေပြန်ဖျက်တာမျိုး လုပ်စရာမလိုတော့ဘဲ **php artisan command** နဲ့ပဲ **database table** တွေကိုပါပြင်ဆင်နိုင်ပါတယ်။

Example Migration

```
public function up()
{
   Schema::create('users', function (Blueprint $table) {
      $table->id();
      $table->string('name');
      $table->string('email')->unique();
      $table->string('password');
      $table->timestamps();
   });
}
```

Migration အသုံးပြုပုံ

```
// Migration create
php artisan make:migration create_users_table


// Migration run
php artisan migrate
```

```
// Migration rollback
php artisan migrate:rollback
```

5.2 Migration Column types

MySQL database table တွေဆောက်တဲ့အခါ table column အမျိုးအစားသတ်မှတ်ပေးရသလိုပဲ Laravel migration ကနေ table တွေတည်ဆောက်တဲ့အခါမှာလည်း column အမျိုးအစားကြေငြာပေးဖို့လိုပါတယ်။

Integer Types

integer() - ပုံမှန် ကိန်းပြည့်တန်ဖိုးများသိမ်းရန်

bigInteger() - ကြီးမားတဲ့ကိန်းပြည့်တန်ဖိုးများသိမ်းရန်

tinyInteger() - သေးငယ်တဲ့ကိန်းပြည့်တန်ဖိုးများသိမ်းရန်

unsignedInteger() - အနုတ်ကိန်းပြည့်မပါဝင်ဘဲ အပေါင်းကိန်းပြည့်များသာသိမ်းရန်

ဥပမာ -

```
$table->integer('quantity'); // ပုံမှန်ကိန်းပြည့်
$table->bigInteger('population'); // လူဦးရေကဲ့သို့ကြီးသည့်ကိန်းပြည့်တန်ဖိုး
$table->tinyInteger('is_active'); // 1 သို့ 0
$table->unsignedInteger('views'); // ကြည့်ရှုသူအရေအတွက်ဆိုတာမျိုးက အနုတ်လက္ခဏာမရှိနိုင်
```

String Types

Here are the maximum character lengths for Laravel's string column types:

- string: 255 characters
- text: 65,535 characters (64KB)
- longtext: characters (4GB)

Note: These values correspond to MySQL's limits. If using a different database driver, the limits may vary.

string() - စာသားတိုတွေ (VARCHAR) 255 လုံးအထိသိမ်းနိုင်

text() - စာပိုဒ်ရှည် စာလုံးရေ 65,535 ထိသိမ်းနိုင်

longText() - စာအရှည်ကြီးများသိမ်းရန် (4,294,967,295 လုံးအထိ)

char() - စာလုံးအရေအတွက်အတိအကျသိမ်းရန်

```
$table->string('name', 100); // တွေ့နေကျနာမည်တွေ
$table->text('description'); // အကျယ်ဖွင့်ဆိုချက်လိုစာပိုဒ်မျိုးတွေ
$table->longText('content'); // facebook post လိုစာအရှည်ကြီးတွေ
$table->char('country_code', 2); // country code ကို integer နဲ့သိမ်းရင် + prefix ထည့်မရလို့ char နဲ့သိမ်း
```

Date & Time Types

date() - ရက်စွဲသီးသန့်

time() - အချိန်သီးသန့်

dateTime() - ရက်စွဲနဲ့အချိန်

timestamp() - timestamp

timestamps() - created_at နဲ့ updated_at columns နှစ်ခုလုံး

```
$table->date('birth_date');
$table->time('opening_time');
$table->dateTime('published_at');
$table->timestamp('last_login_at');
$table->timestamps();
```

Boolean & Enumeration:

boolean() - true/false တန်ဖိုးတွေအတွက်

enum() - သတ်မှတ်ထားတဲ့တန်ဖိုးတွေထဲကတစ်ခုရွေးချယ်ဖို့

```
$table->boolean('is_admin'); // admin ဆိုရင် true ၊ admin မဟုတ်ရင် false

$table->enum('status', ['pending', 'active', 'cancelled']); // ပေးထားချက်ထဲက တန်ဖိုးတစ်ခုပဲသိမ်းလို့ရ
```

Decimal & Float Types:

decimal() - total digits နဲ့ decimal places ကို သတ်မှတ်ပေးနိုင်ပါတယ်။

ဥပမာ $table->decimal('amount', 8, 2) ဆိုရင် total digits 8 လုံး၊ decimal place 2 နေရာ ရှိမယ်လို့ သတ်မှတ်လိုက်တာပါ။ အဲ့ဒါဆိုရင် အများဆုံး 999999.99 ထိ သိမ်းလို့ရပါတယ်။

float() နဲ့ double() ကတော့ decimal ထက် ပိုကြီးတဲ့ နံပါတ်တွေကို သိမ်းဆည်းတဲ့အခါ သုံးပါတယ်။ double က float ထက် ပိုကြီးတဲ့ နံပါတ်တွေ သိမ်းလို့ရပြီး ပိုတိကျမှုရှိပါတယ်။ သို့သော် နေရာပိုယူပါတယ်။ ဥပမာ $table->float('price') နဲ့ $table->double('amount')လို့သုံးနိုင်ပါတယ်။

Unsigned ဆိုတဲ့ data type modifier ကိုလည်း သုံးနိုင်ပါတယ်။ အဲ့ဒါကတော့ အနုတ်ကိန်းတွေကို လက်မခံပဲ အပေါင်းကိန်းတွေပဲ သိမ်းဆည်းစေချင်တဲ့အခါ အသုံးပြုပါတယ်။

ဥပမာ $table->unsignedDecimal('price', 8, 2)လို့ သုံးနိုင်ပါတယ်။

```
$table->decimal('price', 8, 2);  // total 8 digits, 2 after decimal
$table->float('temperature');
$table->double('scientific_calculation');
```

Special Types

json() type ကို complex data structure တွေသိမ်းဖို့သုံးပါတယ်။ ဥပမာ user settings, configuration options, သို့မဟုတ် flexible attributes တွေကို JSON format နဲ့ column တစ်ခုထဲမှာ သိမ်းလို့ရပါတယ်။ Laravel က automatic serialization နဲ့ deserialization ပြုလုပ်ပေးတဲ့အတွက် PHP array သို့မဟုတ် object တွေကို database ထဲမှာ JSON string အနေနဲ့ လွယ်လွယ်ကူကူ သိမ်းနိုင်ပါတယ်။

binary() type ကတော့ binary dataတွေဖြစ်တဲ့ images, files, encrypted data စတာတွေကို သိမ်းဆည်းဖို့ အသုံးပြုပါတယ်။ ဒီ type က raw byte data တွေကို သိမ်းဆည်းနိုင်ပြီး data corruption မဖြစ်အောင် ကာကွယ်ပေးပါတယ်။ ဒါပေမယ့်လက်တွေ့မှာ large binary files တွေကို database ထဲတိုက်ရိုက်သိမ်းတာထက် file system မှာသိမ်းပြီး database ထဲမှာ file path ကိုပဲ သိမ်းတာက ပိုကောင်းပါတယ်။ database ထဲကို တိုက်ရိုက်သိမ်းရင် database size က သိသိသာသာ ကြီးလာပြီး backup နဲ့ recovery လုပ်တာ ပိုခက်ခဲသွားပါတယ်။ ဒါ့အပြင် database queries တွေရဲ့performance ကိုလည်း ထိခိုက်စေနိုင်ပါတယ်။

File system မှာက ဖိုင်ဆိုဒ်ကြီးတွေကို သိမ်းဆည်းနိုင်ဖို့ သေချာဒီဇိုင်းထုတ်ထားပြီးသားဖြစ်တာကြောင့် ပိုမြန်ဆန်ပြီး ချဲ့ရင်ချဲ့သလောက် scalable ဖြစ်ပါတယ်။ ဒါ့အပြင် content delivery networks (CDN) တွေသုံးပြီး files တွေကို distribute လုပ်တာလည်း ပိုလွယ်ကူပါတယ်။ File system မှာ သိမ်းထားတဲ့အတွက် access control နဲ့ encryption တွေကို OS level မှာ ပဲလုပ်နိုင်ပြီး Database load ကိုလည်း လျှော့ချနိုင်တဲ့အတွက် application ရဲ့ overall performance ပိုကောင်းမွန်စေပါတယ်။

uuid() type က globally unique identifierတွေကို သိမ်းဆည်းဖို့သုံးပါတယ်။ UUID တွေဟာ 32 characters ရှည်တဲ့ hex string တွေဖြစ်ပြီး SQL မှာ auto-increment လုပ်ထားတဲ့ ID တွေထက် SQL injection ရန်ကပိုကာကွယ်နိုင်ပါတယ်။

```
$table->json('settings');
$table->binary('file_data');
$table->uuid('id');
```

5.2.1 Column Modifiers

nullable() - null တန်ဖိုးလက်ခံနိုင်အောင် သတ်မှတ်ပေးတာဖြစ်ပါတယ်။ မဖြည့်ဘဲ optional ချန်ခဲ့လို့ရတဲ့ တန်ဖိုးတွေအတွက်ဖြစ်ပါတယ်။ ဥပမာ နေရပ်လိပ်စာမဖြည့်ချင်ရင် ချန်ထားခဲ့လို့ရအောင် nullable() ပေးထားနိုင်ပါတယ်။

default() - default တန်ဖိုးသတ်မှတ်ဖို့ဖြစ်ပါတယ်။

ဥပမာ $table->boolean('is_active')->default(true); ပေးထားရင် is_active column တန်ဖိုးဟာ တမင်တကာ false မထားမချင်း အမြဲ true နဲ့သိမ်းပေးနေမှာပါ။

unique() - unique ဖြစ်စေဖို့ရည်ရွယ်ပါတယ်။ user account ဖွင့်တဲ့အခါ email တစ်ခုတည်းနဲ့ account နှစ်ခုဖွင့်တာမျိုးမရှိရလေအောင် email column ကို unique() ပေးထားလို့ရပါတယ်

index() - index ထည့်ဖို့ဖြစ်ပါတယ်

unsigned() - အနုတ်လက်မခံအောင်ဖြစ်ပါတယ်

```
$table->string('email')->unique();
$table->integer('points')->default(0);
$table->string('middle_name')->nullable();
$table->integer('user_id')->unsigned()->index();
```

Column type တွေကို project အလိုက် လိုအပ်ချက်ပေါ်မူတည်ပြီး သင့်တော်တာကိုရွေးသုံးသင့်ပါတယ်။ ဥပမာ - သာမန် user name တစ်ခုဆိုရင် string() နဲ့လုံလောက်ပေမယ့် blog post content တစ်ခုဆိုရင် text() သို့မဟုတ် longText() သုံးသင့်ပါတယ်။

5.2.2 Relationship Modifiers

foreignId()

Table တွေကြားမှာ relationship ဖန်တီးဖို့သုံးတဲ့ method ဖြစ်ပါတယ်။ Foreign key column တစ်ခုလုပ်ပေးပြီး default အနေနဲ့ BIGINT UNSIGNED data type နဲ့ column လုပ်ပေးပါတယ်

```
$table->foreignId('user_id'); // creates user_id column as foreign key
```

constrained()

Foreign key constraint လုပ်ပေးတဲ့ method ပါ။ constrained() method ရဲ့parameter ထဲမှာ table name မထည့်ပေးလည်း Laravel အမည်ပေးစနစ်အရ column name ကနေ သူနဲ့ဆက်စပ် table name ကို ခန့်မှန်းနိုင်ပါတယ်။

```
// user_id လို့ပေးထားလို့ users table ကို reference လုပ်မယ်လို့ အလိုအလျောက်ခန့်မှန်းနိုင်ပါတယ်

$table->foreignId('user_id')->constrained();
```

```
// user_id ဆိုပေမယ့် တခြား table ကို reference လုပ်ချင်ရင် table name parameter ပေးနိုင်

$table->foreignId('user_id')->constrained('customers');
```

cascadeOnDelete()

မိခင် **record** ကို **delete** လုပ်တဲ့အခါ ဆက်စပ်နေတဲ့ **child records** တွေကိုပါ အလိုအလျောက်ဖျက်ဖို့ သုံးပါတယ်။ **user** တစ်ယောက်ကိုဖျက်လိုက်ရင် သူနဲ့ဆက်စပ်နေတဲ့ **table** တွေက **data** တွေပါပျက်သွားအောင်ဖြစ်ပါတယ်။

```
$table->foreignId('user_id')
    ->constrained()
    ->cascadeOnDelete();
```

5.2.3 How to create files

Project အတွက်လိုအပ်တဲ့ **model file** ၊ **migrate file** တွေကို **new file** တစ်ခုရယူပြီးဖန်တီးနိုင်သလို **artisan command** တွေ **terminal** မှာ **run** ပြီးလည်းဖန်တီးနိုင်ပါတယ်။ **Laravel** မှာ **model** နဲ့ **migration** တွေကို **command** နဲ့တည်ဆောက်တာက **convention over configuration** ဆိုတဲ့ စည်းမျဉ်းကို လိုက်နာထားတာဖြစ်ပါတယ်။ **right click > new file** နှိပ်ပြီး **model** နဲ့ **migration** တွေကို ကိုယ်တိုင်ရေးလို့ရပေမယ့် သတိလက်လွတ်ဖြစ်သွားတတ်တဲ့ အမှားလေးတွေရှိတတ်ပါတယ်။

Command တွေကနေ တည်ဆောက်တဲ့အခါ **Laravel** က လိုအပ်တဲ့ **boilerplate code** တွေကို အလိုအလျောက် ထည့်သွင်းပေးပြီး သက်ဆိုင်ရာ နေရာမှန်မှာ ဖိုင်တွေကို တည်ဆောက်ပေးပါတယ်။ ဥပမာ **model** တွေအတွက် **app/Models directory** ထဲမှာ၊ **migration** တွေအတွက် **database/migrations directory** ထဲမှာ သူ့ဘာသာသူဖန်တီးပေးပါတယ်။

command နဲ့ ဖိုင်တွေဖန်တီးတာက **Laravel** အမည်ပေးစနစ်တွေကိုလည်း လိုက်နာတာကြောင့် **model** နဲ့ **migration** ချိတ်ဆက်တဲ့အပိုင်းကိုလည်း နောက်ကွယ်မှာအလိုအလျောက်လုပ်ဆောင်ပေးပါတယ်။

Laravel မှာ **model** တစ်ခုတည်း **create** လုပ်ချင်ရင်

```
php artisan make:model ModelName
```

ဆိုတဲ့ **command** ကို အသုံးပြုနိုင်ပါတယ်။ **migration** တစ်ခုတည်းပဲ **create** လုပ်မယ်ဆိုရင်

```
php artisan make:migration create_tablename_table
```

ဆိုတဲ့ **command** ကို သုံးရပါတယ်။ **Model** နဲ့ **migration** ကို တစ်ခါတည်း တပြိုင်နက်တည်း **create** လုပ်ချင်တယ်ဆိုရင်တော့

```
php artisan make:model ModelName -m
```

ဆိုပြီး **-m flag** လေးထည့်ပေးရပါတယ်။ **Migration filename** က **create_modelnames_table** ဆိုပြီး အများကိန်းနဲ့ အလိုအလျောက် ထွက်လာမှာပါ။ ဒီနေရာမှာသတိပြုဖို့က **Model** နာမည်တွေကို **PascalCase** နဲ့ရေးပြီး **Singular form** သုံးရပါတယ်။ **Migration** နာမည်တွေကိုတော့ **snake_case** နဲ့ရေးပြီး **plural form** သုံးရပါတယ်။ ဥပမာ ဆေးစတိုတွေကို ကိုယ်စားပြုဖို့ **Warehouse model class** ဖန်တီးတဲ့အခါ **Warehouse** လို့ရေးပြီး **table** တည်ဆောက်ဖို့ **migration class** ဖန်တီးတဲ့အခါ **warehouses** လို့ရေးရပါတယ်။

5.3 Laravel Naming Convention

Laravel မှာ **naming convention** အတိုင်းနာမည်ပေးခြင်းက **project** တွေကိုစနစ်တကျနဲ့ မြန်မြန်ဆန်ဆန်တည်ဆောက်နိုင်ပါတယ်။ တခြားနာမည်တွေပေးလို့ရပေမယ့် **Laravel** ရဲ့ **auto-loading** လုပ်ဆောင်ချက်တွေ၊ **route-model binding** တွေနဲ့ တခြား **built-in features** တွေက **naming convention** အတိုင်းမှ အလိုအလျောက် အလုပ်လုပ်မှာဖြစ်ပါတယ်။

ဥပမာအနေနဲ့ **model** တွေကို **singular form** နဲ့ပေးပြီး **table names** တွေကို **plural form** နဲ့ပေးရတာဟာ **Laravel** က **database table** တွေကို အလိုအလျောက်ရှာဖွေတဲ့အခါ အဆင်ပြေစေဖို့ဖြစ်ပါတယ်။ **User model** ဆိုရင် **users table** ကို အလိုအလျောက်ရှာမှာဖြစ်ပါတယ်။ တကယ်လို့ကိုယ့်ဘာသာနာမည်ပြောင်းချင်ရင် **model** ထဲမှာ **protected $table property** နဲ့ သတ်မှတ်ပေးရပါတယ်။

Controller တွေကိုလည်း **PascalCase** နဲ့ **Controller** ဆိုတဲ့ **suffix** ပါအောင်ပေးရတာဟာ **route provider** က **controller** တွေကို **auto-load** လုပ်တဲ့အခါ မှန်ကန်တဲ့ **class** တွေကို အလွယ်တကူရှာဖွေနိုင်စေဖို့ဖြစ်ပါတယ်။ တကယ်လို့ **convention** အတိုင်းမဟုတ်ဘဲ ကိုယ်ပိုင်နာမည်ပေးချင်ရင် **route definition** တွေမှာ **fully qualified class name** တွေကို အသုံးပြုပေးရပါတယ်။

Migration files တွေကိုလည်း **timestamp_create_table_name_table.php** ဆိုတဲ့ **format** နဲ့ပေးရတာဟာ **database migrations** တွေကို အစီအစဉ်တကျ **run** နိုင်စေဖို့နဲ့ **rollback** လုပ်တဲ့အခါ မှန်ကန်တဲ့ အစီအစဉ်အတိုင်းဖြစ်စေဖို့ဖြစ်ပါတယ်။ တခြားပုံစံနဲ့နာမည်ပေးမယ်ဆိုရင် **migration sequence** ကို **manual** ထိန်းချုပ်ပေးရမှာဖြစ်ပါတယ်။

Class Names နဲ့ပတ်သက်ပြီး **Laravel** မှာ **PascalCase** ကို အသုံးပြုပါတယ်။ ဥပမာအနေနဲ့ **UserController, ProductModel** စသည်ဖြင့် ရေးသားပါတယ်။ **Controller** တွေကို အမည်ပေးတဲ့အခါ နောက်ဆုံးမှာ **Controller** ဆိုတဲ့ စကားလုံးကို ထည့်ပေးရပါတယ်။ ဥပမာ **PostsController, HomeController** စသဖြင့်ပါ။

Method Names တွေအတွက် **camelCase** ကိုအသုံးပြုပါတယ်။ ဥပမာ **getUserDetails(), createPost(), updateProfile()** စသဖြင့် ရေးသားပါတယ်။ **Method** တွေရဲ့နာမည်တွေက သူတို့လုပ်ဆောင်တဲ့ အလုပ်ကို ရှင်းရှင်းလင်းလင်း ဖော်ပြနိုင်ရပါမယ်။

Variable Names တွေကိုလည်း **camelCase** နဲ့ပဲ ရေးသားပါတယ်။ ဥပမာ **firstName, totalAmount, userEmail** စသဖြင့် နာမည်ပေးသင့်ပါတယ်။ **Variable** တွေရဲ့နာမည်တွေဟာ သူတို့ထဲမှာ သိမ်းဆည်းထားတဲ့ တန်ဖိုးတွေကို ကိုယ်စားပြုနိုင်ရပါမယ်။

Database Table Names တွေကို **snake_case** နဲ့ ရေးသားပြီး **plural form** ကို သုံးလေ့ရှိပါတယ်။ ဥပမာအနေနဲ့ **users, blog_posts, product_categories** စသဖြင့် ရေးသားပါတယ်။ **Pivot table** တွေအတွက် **singular form** နဲ့ **alphabetical order** အတိုင်း ရေးရပါမယ်။ ဥပမာ **article_user, permission_role** စသဖြင့်ပါ။

Model Names တွေအတွက် **PascalCase** ကို သုံးပြီး **singular form** ဖြစ်ရပါမယ်။ ဥပမာ **User, Post, ProductCategory** စသဖြင့် ရေးသားပါတယ်။ **Model** တွေဟာ **database table** တွေရဲ့**singular version** တွေဖြစ်ကြပါတယ်။

Migration Files တွေရဲ့နာမည်တွေကိုတော့ **snake_case** နဲ့ ရက်စွဲပါတဲ့ **prefix** နဲ့တွဲပြီး ရေးလေ့ရှိပါတယ်။ ဥပမာ **2024_12_02_create_users_table.php, 2024_12_02_add_avatar_to_users_table.php** စသဖြင့် ရေးသားပါတယ်။

View Files တွေအတွက် **snake_case** ကိုသုံးပြီး **dot notation** နဲ့ခွဲခြားရေးသားပါတယ်။ ဥပမာ **all.blade.php, user.profile.blade.php, admin.dashboard.blade.php** စသဖြင့် ရေးသားပါတယ်။

Configuration Files တွေကိုလည်း **snake_case** နဲ့ပဲ ရေးသားပါတယ်။ ဥပမာ **mail.php, database.php, cache.php** စသဖြင့် ရေးသားပါတယ်။ **Config file** တွေထဲက **array key** တွေကိုလည်း **snake_case** နဲ့ပဲ ရေးပါတယ်။

Laravel ရဲ့နာမည်ပေးပုံတွေက **PSR standards** တွေနဲ့ **PHP community practices** တွေကို အခြေခံထားတာဖြစ်ပါတယ်။ ဒီ **convention** တွေကို လိုက်နာခြင်းအားဖြင့် **code** ဟာ ပိုပြီး **readable** ဖြစ်လာပြီး **maintainable** ဖြစ်စေပါတယ်။ အထူးသဖြင့် **team project** တွေမှာ **developer** တွေကြား **code** ကို နားလည်ရလွယ်ကူစေပါတယ်။

ယခု **project** မှာ ပထမဆုံးအနေနဲ့ **Transactionmodel** နဲ့ **transactions table** ဆောက်ကြည့်ပါမယ်။

5.4 Transaction

ပစ္စည်းအဝင်အထွက်ကိုမှတ်တမ်းတင်မယ့် **table** ဖြစ်ပါတယ်။ ဒီ **command** ကို **run** ကြည့်ပါ။

```
php artisan make:model Transaction -m
```

app\Models\Transaction.php နဲ့ database\migrations\<datetime>_create_transactions_table.php နှစ်ခုရလာပါလိမ့်မယ်။ _create_transactions_table.php migrate file ကိုဖွင့်ပါ။

Visual Studio Code မှာ php artisan make command နဲ့ ရလာတဲ့ file တွေကို အမြန်ဖွင့်တဲ့နည်းလေးတွေရှိပါတယ်။

Ctrl + P (Windows) သို့မဟုတ် Cmd + P (Mac) ကိုနှိပ်ပြီး file နာမည်ကို ရိုက်ထည့်၊ Enter ခေါက်ရုံနဲ့ ဖွင့်နိုင်ပါတယ်။

Ctrl + B (Windows) သို့မဟုတ် Cmd + B (Mac) နှိပ်ပြီး Explorer ကိုဖွင့်၊ ပြီးရင် file တွေကို folder structure အတိုင်းရှာပြီး click လုပ်ကာဖွင့်နိုင်ပါတယ်။

ခုလေးတင် php artisan make: command သုံးပြီးလုပ်ထားတဲ့ဖိုင်တွေကိုတော့ Ctrl key ဖိထားပြီး Terminal မှာပေါ်နေတဲ့ file name ကို click နှိပ်ဖွင့်နိုင်ပါတယ်။

transactions migration ရဲ့ public function up() default id နဲ့ timestamps column တွေရှိပါတယ်။ အဲဒီကြားထဲမှာ အောက်ပါအတိုင်းထည့်ပေးပါ။

```php
public function up(): void
{
    Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item');
        $table->string('category');
        $table->string('package_form');
        $table->date('exp_date')->nullable();
        $table->string('batch')->nullable();
        $table->integer('amount');
        $table->string('donor');
        $table->string('source');
        $table->string('destination')->nullable();
        $table->text('remarks')->nullable();
        $table->timestamps();
    });
}
```

Schema::create('transactions') ဆိုတာက transactions ဆိုတဲ့နာမည်နဲ့ database table တစ်ခုတည်ဆောက်မယ်လို့ဆိုလိုပါတယ်။

date column က ရက်စွဲတွေသိမ်းမှာမို့လို့ date အမျိုးအစားပေးရပါမယ်။

type က ပစ္စည်းအဝင်၊ အထွက်၊ သက်တမ်းလွန်၊ ပျက်စီးစတဲ့ အမျိုးအစားကိုထည့်သွင်းရမှာမို့လို့ string ဖြစ်ပါတယ်

item ကဝင်လာတဲ့၊ ထွက်သွားတဲ့ ပစ္စည်းနာမည်တွေသိမ်းဖို့ string ပါ

category က ပစ္စည်းအမျိုးအစားမို့လို့ string ပါပဲ

package_form ကထုပ်ပိုးပုံစံ string တန်ဖိုးပါ။

exp_date က date ဖြစ်ပါမယ်။

batch က နံပါတ်နဲ့စာသားအတွဲတွေသိမ်းမှာမို့လို့ **string** ဖြစ်ပါတယ်။

amount က အပေါင်းလက္ခဏာနဲ့ အနုတ်လက္ခဏာ ကိန်းပြည့်တွေပဲသိမ်းမှာဖြစ်တဲ့အတွက် **integer** ပါ

donor မှာ အလှူရှင်အဖွဲ့အစည်းနာမည်တွေသိမ်းဖို့အတွက် **string** ပါ

source ကတော့ ဘယ်ဆေးစတိုကရတာလဲဆိုတာမှတ်ချင်တဲ့အတွက် **string** ဖြစ်ပါတယ်။

destination - ပစ္စည်းအထွက်မှတ်တမ်းသိမ်းတဲ့အခါ ဘယ်စတိုကိုပေးလိုက်တာလဲဆိုတာသိမ်းဖို့ပါ

remarks မှတ်ချက်က ၂၅၅ လုံးထက်ပိုရှည်နိုင်တာမို့ **text** အမျိုးအစားပေးပါတယ်။

နောက်ပိုင်းမှာ တခြား **table** တွေဆောက်ပြီးရင် **transactions table** နဲ့ **relationship** လာချိတ်ဖို့အတွက် **column** အမျိုးအစားတွေပြောင်းပေးရမှာပါ။ အခုလောလောဆယ်တော့နားလည်လွယ်အောင် ဒီလောက်နဲ့အရင်စလုပ်ကြည့်ပါမယ်။

ပြီးသွားရင် ဒီဖိုင်ကို **save** လုပ်ပြီးပိတ်လို့ရပါပြီ။

app\Models\Transaction.php ကိုဆက်ဖွင့်ပါ။ **migration file** မှာအသစ်ထပ်ဖြည့်ခဲ့တဲ့ **column** တွေကို **Transaction class** ထဲမှာ **$fillable array variable** ထဲထည့်သွင်းပေးပါ။

$fillable ထဲမှာ **id** နဲ့ **timestamps** တို့ကိုထည့်စရာမလိုတော့ပါဘူး။

```
class State extends Model
{
  protected $fillable = [
    'date',
    'type',
    'item',
    'category',
    'package_form',
    'exp_date',
    'batch',
    'amount',
    'donor',
    'source',
    'destination',
    'remarks'
  ];
}
```

5.4.1 protected, private and public

protected ဆိုတာ **OOP (Object-Oriented Programming)** ရဲ့ **access modifier** တစ်ခုဖြစ်ပါတယ်။ **protected** နဲ့ရေးထားတဲ့တန်ဖိုးတွေကို သက်ဆိုင်ရာ **class** ထဲကနေပဲ ခေါ်သုံးလို့ရပြီး အဲဒီ **class** ကို ဆက်ခံ **(inherit** လုပ်ထား**)** တဲ့ **child classes** တွေကနေလည်း ခေါ်သုံးလို့ရပါတယ်။ မူရင်း **class** မှာ **protected** တန်ဖိုးတစ်ခုကြေညာထားရင် အဲ့ဒီတန်ဖိုးကို မူရင်း **(Parent) class** ရဲ့ **methods** တွေကနေရော၊ မူရင်း **class** ကို **extend** လုပ်ထားတဲ့ ဆင့်ပွား **(Child) class** တွေကပါ အသုံးပြုနိုင်ပါတယ်။

protected ဟာ private နဲ့ public ကြားက access level တစ်ခုလို့ပြောလို့ရပါတယ်။ private ကတော့ ကြေငြာထားတဲ့ class ထဲမှာပဲ access လုပ်လို့ရပြီး၊ public ကတော့ ဘယ်နေရာကမဆို access လုပ်လို့ရပါတယ်။ ဒါပေမယ့် protected ကတော့ မူရင်းနဲ့ ဆင့်ကဲ inheritance chain ထဲမှာ access လုပ်လို့ရမှာဖြစ်ပါတယ်။

Laravel framework မှာဆိုရင် protected properties တွေကို Models class တွေမှာအများဆုံး တွေ့ရပါတယ်။ ဥပမာ $fillable, $guarded, $table စတဲ့ properties တွေဟာ protected ဖြစ်ကြပါတယ်။

5.4.2 $

Laravel မှာ $ သင်္ကေတကို အဓိကအားဖြင့် variable တွေကြေညာတဲ့အခါ သုံးပါတယ်။ PHP မှာ $ သင်္ကေတကို variable တွေအတွက် မဖြစ်မနေသုံးရတဲ့ syntax ဖြစ်တာကြောင့် Laravel framework မှာလည်း အဲ့ဒီ PHP ရဲ့ syntax ကိုပဲ ဆက်လက်အသုံးပြုထားပါတယ်။ ဥပမာအနေနဲ့ $name = "Mg Mg" လို့ variable တစ်ခုကြေညာတဲ့အခါ $ သင်္ကေတကို အရှေ့မှာ ထည့်ပေးရပါတယ်။

ထို့အပြင် Laravel ရဲ့ Blade Template တွေထဲမှာလည်း PHP variable တွေကို access လုပ်တဲ့အခါ $ သင်္ကေတကို သုံးရပါတယ်။ ဥပမာ {{ $user->name }} လိုမျိုး user object ထဲက property တွေကို access လုပ်တဲ့အခါ သုံးပါတယ်။ အလားတူပဲ controller ထဲမှာ database query တွေရေးတဲ့အခါ query မှရလာတဲ့ result တွေကို variable ထဲ သိမ်းဖို့လည်း $ သင်္ကေတနဲ့ variable တွေကို အသုံးပြုရပါတယ်။ ဥပမာ $users = User::all() လိုမျိုးပါ။

Dependency Injection လုပ်တဲ့အခါမှာလည်း parameter တွေကို $ နဲ့ ကြေညာပေးရပါတယ်။ ဥပမာအနေနဲ့ controller ရဲ့ method တစ်ခုထဲမှာ Request object ကို inject လုပ်တဲ့အခါ public function store(Request $request) လိုမျိုး $ သင်္ကေတနဲ့ parameter ကို ကြေညာပေးရပါတယ်။ ဒါ့အပြင် global helper function တွေဖြစ်တဲ့ $_GET၊ $_POST၊ $_SESSION စတာတွေမှာလည်း $ သင်္ကေတနဲ့ စရေးရပါတယ်။

5.4.3 =, == and ===

= သင်္ကေတက assignment operator ဖြစ်ပါတယ်။ variable တစ်ခုကို တန်ဖိုးတစ်ခု သတ်မှတ်ပေးဖို့အတွက် သုံးပါတယ်။ ဥပမာ $name = "Mg Mg" လို့ရေးရင် $name ဆိုတဲ့ variable ထဲကို "Mg Mg" ဆိုတဲ့ string တန်ဖိုးကို assign လုပ်လိုက်တာပါ။

== ကတော့ နှိုင်းယှဉ်မယ့် တန်ဖိုးနှစ်ခုရဲ့တန်ဖိုးချင်း တူမတူကိုပဲ စစ်ဆေးပါတယ်။ data type ကို ထည့်မစစ်ပါဘူး။ ဥပမာ 5 == "5" ဆိုရင် true ပဲဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ integer 5 နဲ့ string "5" က တန်ဖိုးအရ တူညီနေလို့ပါပဲ။

=== ကတော့ တန်ဖိုးတွေရဲ့တန်ဖိုးရော၊ data type ပါ တူမတူကို စစ်ဆေးပေးပါတယ်။ ဥပမာ 5 === "5" ဆိုရင် false ဖြစ်ပါတယ်။ ဘာကြောင့်လဲဆိုတော့ တန်ဖိုးက တူပေမယ့် တစ်ခုက integer type ဖြစ်ပြီး နောက်တစ်ခုက string type ဖြစ်နေလို့ပါ။ Laravel မှာ တန်ဖိုးတွေကို တိတိကျကျ နှိုင်းယှဉ်ချင်တဲ့အခါ === ကို သုံးတာက ပိုပြီး security ကောင်းပါတယ်။

assignment အတွက် = ကို သုံးပြီး၊ condition တွေမှာ နှိုင်းယှဉ်တဲ့အခါ == နဲ့ === ကို သုံးကြပါတယ်။ တန်ဖိုးချင်း တူရုံနဲ့ လုံလောက်ရင် == ကို သုံးပြီး၊ data type ပါ တူဖို့လိုအပ်တဲ့ နေရာမှာ === ကို သုံးလေ့ရှိပါတယ်။

5.4.4 $fillable property

protected $fillable = []; ထဲဟာ [] က array တစ်ခုကို ကြေငြာပေးတာဖြစ်ပါတယ်။ အဲ့ဒီ array ထဲမှာ mass assignment လုပ်လို့ရမယ့် field တွေကို ထည့်သွင်းပေးရမှာဖြစ်ပါတယ်။ ဥပမာ protected $fillable = ['name']; ဆိုရင် name ဆိုတဲ့ field ကို mass assignment လုပ်လို့ရမယ့် field အဖြစ် သတ်မှတ်ထားတာပါ။

Mass assignment ဆိုတာက database record တွေကို တစ်ခါတည်း အများကြီး insert လုပ်တာ (သို့) update လုပ်တာကို ခေါ်ပါတယ်။ ဥပမာအနေနဲ့ User::create() method ကိုသုံးပြီး user တစ်ယောက်ကို create လုပ်တဲ့အခါ request data တွေအားလုံးကို တစ်ခါတည်း assign လုပ်တာမျိုးဖြစ်ပါတယ်။

```
$user = User::create([
  'name' => 'Mg Mg',
  'email' => 'mgmg@gmail.com',
```

```
    'password' => 'password123'
]);
```

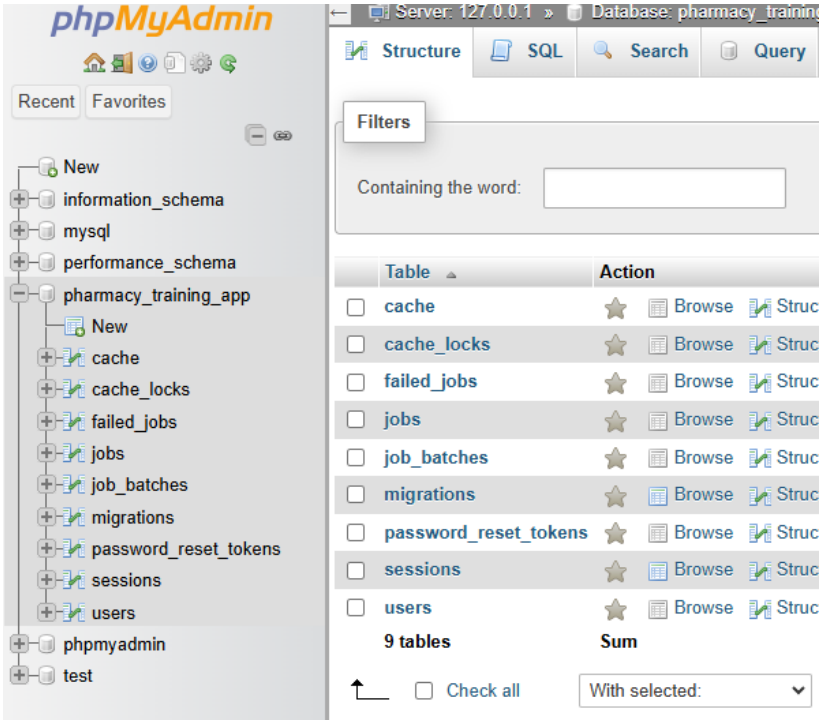
အထက်ပါ **code** မှာဆိုရင် **name, email** နဲ့ **password** ဆိုတဲ့ **data** တွေကို တစ်ခါတည်း **assign** လုပ်နေတာဖြစ်ပါတယ်။ ဒါကြောင့် **fillable property** မှာ ဘယ် **field** တွေကို **mass assignment** လုပ်လို့ရမယ်ဆိုတာ **define** လုပ်ပေးရတာဖြစ်ပါတယ်။ မဟုတ်ရင် **user** ကနေ **sensitive** ဖြစ်တဲ့ **field** တွေကိုပါ **update** လုပ်သွားနိုင်တဲ့အတွက် **security risk** ဖြစ်နိုင်ပါတယ်။ ဥပမာ **is_admin** ဆိုတဲ့ **field** ကို **fillable** ထဲမှာ မထည့်ထားရင် **user** တွေက သူ့ကိုယ်သူ **admin** ဖြစ်အောင် လုပ်လို့ရမှာ မဟုတ်ပါဘူး။ **create() method** ကို သုံးပြီး **record** တစ်ခု **create** လုပ်တဲ့အခါ **fillable** မှာ **define** လုပ်မထားတဲ့ **field** တွေကို **Laravel** က အလိုအလျောက်ဖယ်ထုတ်ပေးသွားမှာဖြစ်ပါတယ်။

သဘောကတော့ **migration** မှာ **column** ဘယ်နှခုရှိရှိ **fillable** မှာ **column** နာမည်တွေမကြေငြာရသေးရင် **user** ဘက်ခြမ်းကနေ **data** လာသိမ်းလို့မရပါဘူး။

Model မှာ **fillable property** တွေသတ်မှတ်ပြီးရင် **save** လုပ်ပြီး ဖိုင်ပိတ်လို့ရပါပြီ။ ရှေ့ဆက်ပြီး **migration** နဲ့ **model** တွေမလုပ်ခင် လုပ်ပြီးသလောက်ကို **run** ကြည့်ပါမယ်။

XAMPP မှာ **Apache Start** သွားလုပ်ပါ။

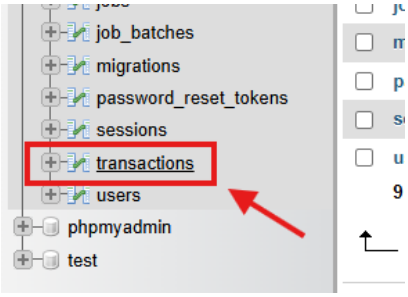
browser မှာ **http://localhost/phpmyadmin** ကိုဖွင့်ကြည့်ပါမယ်။ **pharmacy_training_app database** ဘေးနားက အပေါင်းလက္ခဏာကိုနှိပ်ပြီးအကျယ်ဖွင့်ကြည့်လိုက်ရင် လက်ရှိ **table** တွေကိုမြင်ရပါမယ်။



Visual Studio Code ကိုပြန်လာပြီး **terminal** မှာ **migrate run** ကြည့်ပါ။

```
php artisan migrate
```

transactions table migrate လုပ်သွားတာကို တွေ့ရပါလိမ့်မယ်။ **browser** မှာ **phpMyAdmin** ကို **refresh** လုပ်ကြည့်ပါ။ **pharmacy_training_app database** ထဲမှာ **transactions table** ဝင်လာတာကိုတွေ့ရပါလိမ့်မယ်။



ဒါဆို XAMPP မှာ Apache ပြန် Stop လို့ရပါပြီ။ MySQL ကိုတော့ ဒီတိုင်းထားပါ။

Data လက်ခံဖို့ database table ဆောက်ပြီးပြီဆိုတော့ လက်ရှိလုပ်ပြီးသလောက်ကိုပဲ CRUD features တွေတည်ဆောက်ကြပါမယ်။

5.5 CRUD (Create, Read, Update, Delete)

CRUD ဆိုတာ Create, Read, Update, Delete ဆိုတဲ့ အခြေခံ လုပ်ဆောင်ချက် (၄) မျိုးကို ဆိုလိုပါတယ်။

Create (ဖန်တီးခြင်း)

အချက်အလက်အသစ်များ ထည့်သွင်းခြင်း၊ ဥပမာ - form တစ်ခုမှတဆင့် user အသစ် စာရင်းသွင်းခြင်း

Read (ဖတ်ခြင်း)

ရှိပြီးသား အချက်အလက်များကို ကြည့်ရှုခြင်း၊ ဥပမာ - database ထဲမှ user များ၏ စာရင်းကို ပြသခြင်း

Update (ပြင်ဆင်ခြင်း)

ရှိပြီးသား အချက်အလက်များကို ပြောင်းလဲခြင်း၊ ဥပမာ - user profile အချက်အလက်များကို edit လုပ်ခြင်း

Delete (ဖျက်ခြင်း)

မလိုအပ်တော့သော အချက်အလက်များကို ဖျက်ပစ်ခြင်း၊ ဥပမာ - user account တစ်ခုကို ဖျက်ခြင်း

CRUD interface ကို database အခြေပြု application များတွင် အသုံးများပါတယ်။ ဥပမာ -

1. Web Application များ
2. Mobile App များ
3. Desktop Software များ
4. Content Management System များ

CRUD လုပ်ဆောင်ချက်များတည်ဆောက်တဲ့အခါ လူတိုင်းပေးသုံးမလား၊ ဘယ်သူတွေကို အသုံးပြုခွင့်ပေးမလဲ (Security) ၊ ထည့်သွင်းတဲ့ အချက်အလက်များ မှန်/မမှန် စစ်ဆေးခြင်း (Data validation) ၊ user ထည့်သွင်းတဲ့ error ဖြစ်ဖြစ်၊ application ကပြန်ပြတဲ့ error ဖြစ်ဖြစ် နားလည်လွယ်စေဖို့နဲ့ error ကြိုလာရင်ဘာဆက်လုပ်ခိုင်းမလဲဆိုတဲ့ error handling ၊ အသုံးပြုရ လွယ်ကူအောင် ဒီဇိုင်းဆွဲခြင်း (User Experience) စတာတွေကိုထည့်သွင်းစဉ်းစားဖို့လိုပါတယ်။

Laravel မှာ CRUD လုပ်ဆောင်ချက်တွေကို ကိုယ်တိုင်ရေးသားတည်ဆောက်တဲ့အခါ လိုအပ်တဲ့ controllers, models, views, routes တွေကို တစ်ခုချင်းစီ အသေးစိတ် ရေးသားဖို့လိုပါတယ်။ ဒါ့အပြင် သူများလာဖြည့်တဲ့ data တွေမှန်မမှန်စစ်ဖို့ validation တွေ၊ user authorization တွေ၊ form handling တွေကိုပါ ကိုယ်တိုင်ရေးသားရပါတယ်။ ဒီလိုရေးသားတဲ့အခါ ကိုယ်လိုချင်တဲ့ပုံစံအတိအကျရနိုင်ပြီး၊ flexible ဖြစ်ပေမယ့် အချိန်ပိုကုန်ပြီး code အများကြီးရေးရပါတယ်။

Filament ကတော့ Laravel အတွက် admin panel တည်ဆောက်ဖို့ package တစ်ခုဖြစ်ပါတယ်။ TALL stack (Tailwind CSS, Alpine.js, Laravel, Livewire) ကိုအခြေခံထားပြီး၊ database tables တွေကနေ admin panel တွေကို အလိုအလျောက်တည်ဆောက်ပေးနိုင်ပါတယ်။ CRUD operations တွေအပြင် filtering, sorting, exporting စတဲ့ features တွေပါဝင်ပါတယ်။

Filament သုံးခြင်းအားဖြင့် code အများကြီးရေးစရာမလိုဘဲ admin panel တွေကို လျင်မြန်စွာတည်ဆောက်နိုင်ပါတယ်။ ဒါပေမယ့် အစအဆုံးကိုယ်တိုင်ရေးတာလောက်တော့ customize လုပ်လို့မရပါဘူး။ ဒါကဖြစ်နေကျသဘာဝပါ။ တနည်းအားဖြင့် development speed နဲ့ flexibility ကို trade-off လုပ်ရတယ်လို့ ပြောလို့ရပါတယ်။ အသင့်သုံးလို့ရတာတွေက အချိန်တိုအတွင်း အလုပ်ပြီးပေမယ့် ချဲ့လို့မရတာ၊ အသေးစိတ်ပေးမပြင်တာတွေရှိပါတယ်။ အထူးသဖြင့် drag-and-drop ရတဲ့ low code/no code တွေမှာ development speed မြန်ပေမယ့် လိုချင်သလိုပြင်လို့မရတဲ့အားနည်းချက်ရှိပါတယ်။

CRUD operations တွေများတဲ့ admin dashboards တွေအတွက် Filament ကအသုံးဝင်ပါတယ်။ built-in user roles နဲ့ permissions system တွေပါပါဝင်တဲ့အတွက် authorization အတွက်သပ်သပ်ရေးဖို့မလိုတော့ပါဘူး။ အရမ်း customize လုပ်ရေးရတဲ့ feature တွေအတွက်လည်း LiveWire နဲ့ AlpineJS အထိဆင်းရေးလို့ရပါတယ်။

6 Install Filament and user panel

6.1 Laravel Routing

Routing ဆိုတာ URL requests တွေကို လက်ခံပြီး သက်ဆိုင်ရာ controller/action တွေဆီ လမ်းညွှန်ပေးတဲ့ စနစ်ဖြစ်ပါတယ်။ Laravel project တစ်ခုမှာ routes/web.php ထဲမှာ route တွေသတ်မှတ်ပေးနိုင်ပါတယ်။

```
// routes/web.php ထဲမှာ define လုပ်ပါတယ်

Route::get('/hello', function () {
   return 'Hello World';
});

// Controller နဲ့ချိတ်ဆက်ခြင်း

Route::get('/posts', [PostController::class, 'index']);

// Parameters သုံးခြင်း

Route::get('/post/{id}', [PostController::class, 'show']);

// Named Routes
Route::get('/profile', [UserController::class, 'show'])->name('profile');
```

Filament install လုပ်ဖို့အတွက် terminal မှာ အောက်ပါ command run ပါ။

```
composer require filament/filament
```

Filament နဲ့ သက်ဆိုင်တဲ့ package တွေအားလုံးကို install လုပ်ပေးသွားမှာဖြစ်ပါတယ်။ ပြီးရင် panel install ထပ်လုပ်ပေးပါ

```
php artisan filament:install --panels
```

ID မေးလာရင် user လို့ထည့်ပေးပါ

```
What is the ID? [admin]
› user
```

Github star မေးရင် **no** ထည့်ပေးပါ

```
All done! Would you like to show some love by starring the Filament repo on GitHub? (yes/no) [yes]
› no
```

UserPanelProvider.php နာမည်နဲ့ ဖိုင်အသစ်ရလာပါမယ်။ ဖိုင်ကိုဖွင့်ပြီး **path** ပြောင်းပေးပါ။ ကျန်တာဒီတိုင်းထားပါ။

```
class UserPanelProvider extends PanelProvider
{
   public function panel(Panel $panel): Panel
   {
      return $panel
         ->default()
         ->id('user')
         ->path('/') // 'user' ကနေ '/' ပြောင်းပေးပါ

         ->login()
         ->colors([
            'primary' => Color::Amber,
         ])
         // ...
         ;
   }
}
```

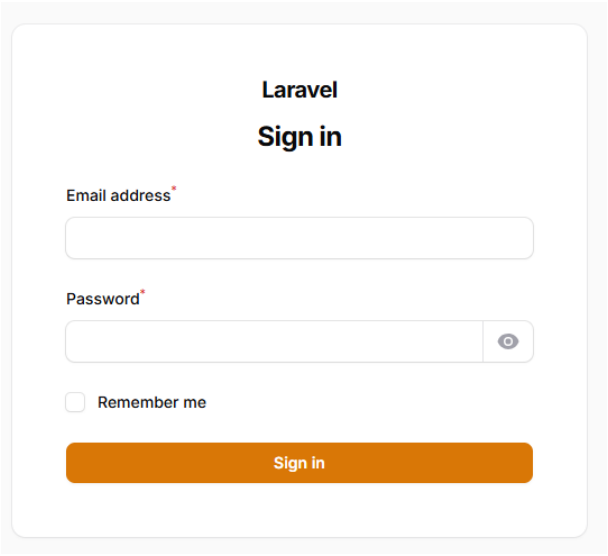
User panel ရဲ့လိပ်စာကို **root** မှာထားမယ်လို့ဆိုလိုတာပါ။ **website** ကိုဝင်ဝင်ချင်း **user panel** ကိုတန်းရောက်ပါမယ်။ ဒါပေမယ့် **Laravel** ရဲ့ **default route** ထဲက **root** (’/’) နဲ့သွားပြီး **conflict** ဖြစ်မှာမို့လို့ **route** ထဲက **web.php** မှာ ပြင်ပေးဖို့လိုပါတယ်။ **Ctrl + P** နှိပ်ပြီး **web.php** လို့ရိုက်ထည့်လိုက်ရင် **route file** ကို အောက်ပါအတိုင်းတွေ့ရပါမယ်။

```
Route::get('/', function () {
   return view('welcome');
});
```

ဒီ **code** သုံးကြောင်းကို **select** လုပ်ပြီး **Ctrl + /** တွဲနှိပ်ပါ။ အလုပ်မလုပ်တော့အောင် **Comment** ပိတ်လိုက်တဲ့သဘောပါ။

```
// Route::get('/', function () {
//    return view('welcome');
// });
```

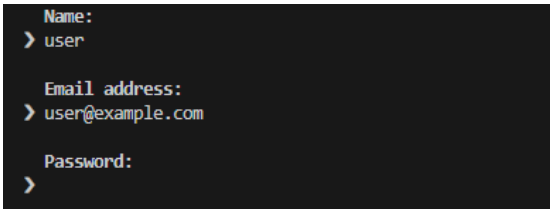
web.php ကို **save** လုပ်ပြီး ပိတ်ပါ။ **browser** မှာ **refresh** ပြန်လာလုပ်ရင် **Filament login page** တွေ့ရပါမယ်။



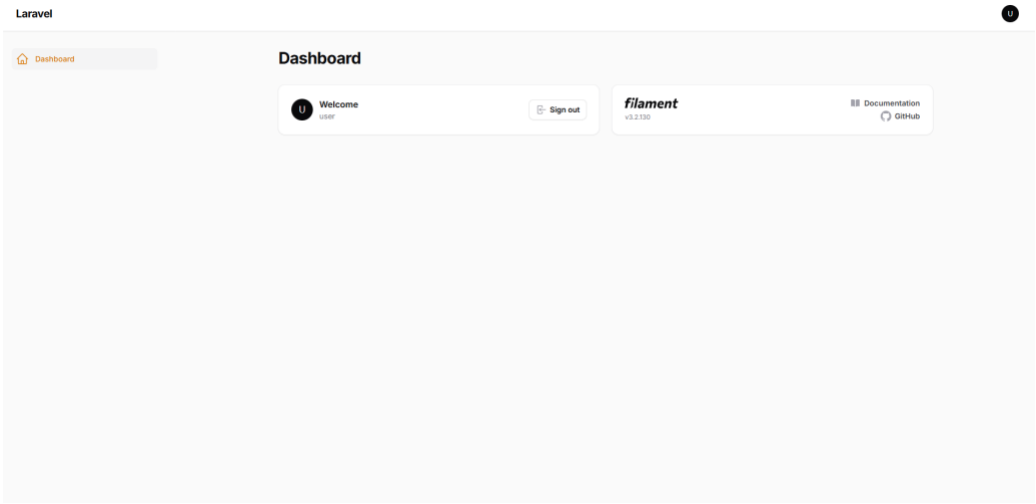
လောလောဆယ် **login** ဝင်ရအောင် **user** တစ်ယောက်မှ မထည့်ရသေးပါဘူး။ ဒါကြောင့် **filament command** နဲ့ပဲ **user registration** လုပ်ပါမယ်။

```
php artisan make:filament-user
```

နှစ်သက်ရာ နာမည်၊ အီးမေးလ်၊ **password** တို့ကိုပေးလို့ရပါတယ်။ ဒီမှာတော့မှတ်ရလွယ်အောင် **user, user@example.com** နဲ့ **password** လို့ပဲပေးလိုက်ပါမယ်။



User registration ပြီးရင် အခုလုပ်ခဲ့တဲ့ **email, password** နဲ့ **login** ဝင်ကြည့်ပါ။ **User Dashboard** ကို ရောက်သွားပါမယ်။



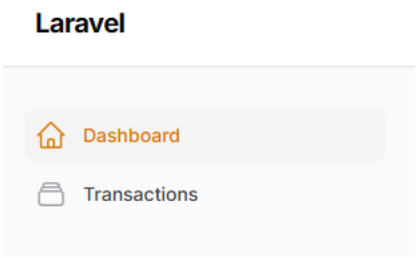
ဒါက **UserPanel** ကိုဝင်ဝင်ချင်းတွေ့ရမယ့် **Dashboard** ဖြစ်ပါတယ်။ ဒီ **panel** မှာ ဆေးပစ္စည်းမှတ်တမ်းတွေ ထည့်သွင်းဖို့စတင်တည်ဆောက်မှာဖြစ်ပါတယ်။

6.2 Transaction Resource

Filament မှာ Create, Read, Update, Delete တွေလုပ်လို့ရမယ့် interface ကို resource လို့ခေါ်ပါတယ်။ ဆေးပစ္စည်းအဝင်အထွက် Transaction မှတ်တမ်းတွေ CRUD လုပ်ဖို့ resource တစ်ခုကိုအောက်ပါ command အတိုင်းပြုလုပ်ပါ။

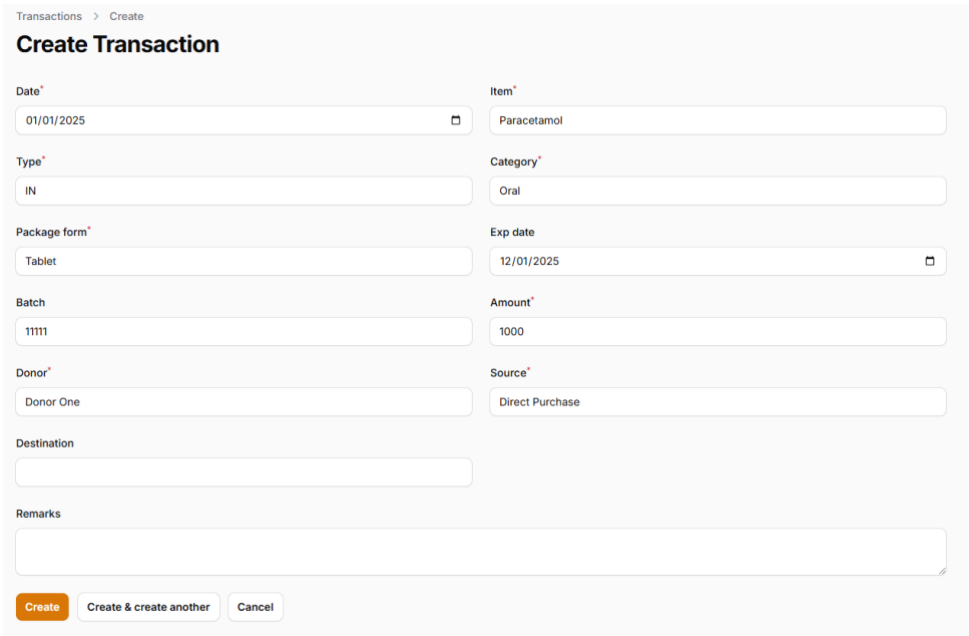
```
php artisan make:filament-resource Transaction --generate
```

TransactionResource.php နာမည်နဲ့ ဖိုင်ရလာပါမယ်။ browser မှာ Transaction menu တစ်ခုတိုးလာပါလိမ့်မယ်။



Transactions ထဲက New transaction ခလုတ်ကိုနှိပ်ပါ။ Create လုပ်လို့ရမယ့် form စာမျက်နှာကိုရောက်ပါမယ်။ --generate flag နဲ့ resource ဖန်တီးခဲ့တာမို့လို့ database schema နဲ့အနီးစပ်ဆုံးဖြစ်တဲ့ form field တွေအလိုအလျောက်တည်ဆောက်ပေးတာကို တွေ့ရမှာဖြစ်ပါတယ်။

Date မှာ နှစ်သက်ရာရက်စွဲတစ်ခုရွေးပါ။ Type မှာ IN ၊ Item မှာ Paracetamol ၊ Category မှာ Oral ၊ Package Form မှာ Tablet ၊ Exp Date မှာ နှစ်သက်ရာရက်စွဲ၊ Batch မှာ နှစ်သက်ရာဂဏန်းအတွဲ၊ Amount မှာ 1000 ၊ Donor မှာ Donor One ၊ Source မှာ Direct Purchase ၊ Destination နဲ့ Remarks မှာ အလွတ်ထားပါ။ ပြီးရင် Create ခလုတ်ကိုနှိပ်ပြီး data သိမ်းကြည့်ပါ။



Edit Transaction လို့ပြောင်းသွားပါလိမ့်မယ်။ Data သိမ်းပြီးသွားလို့ Edit page ကိုရောက်သွားတဲ့သဘောပါ။ ဘယ်ဘက် menu က Transactions ကိုပြန်နှိပ်ကြည့်ပါ။ Table view ကိုရောက်သွားပါမယ်။

Data သိမ်းပြီးတာနဲ့ table view ကိုအလိုလိုရောက်သွားအောင် ဆက်လုပ်ကြည့်ပါမယ်။

6.3 Redirect after create

6.3.1 Static in Object Oriented Programming

Static မဟုတ်တဲ့ class ကို instance လုပ်ချင်ရင် new keyword သုံးရပါတယ်။

```php
// Class definition
class User {
    public function getName() {
        return "Aung Aung";
    }
}

// Creating instance
$user = new User();
$user->getName(); // "Aung Aung" ကို return ပြန်ပါမယ်
```

Constructor နဲ့လည်း instance လုပ်လို့ရပါတယ်။

```php
class Product {
    private $name;
    private $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

    public function getDetails() {
        return "Product: " . $this->name . ", Price: $" . $this->price;
    }
}

// Creating instance with constructor parameters
$product = new Product("iPhone", 999);
echo $product->getDetails();
```

6.3.2 Static and Non-static difference

Static

```php
class Math {
    public static function add($a, $b) {
        return $a + $b;
    }
}

// static မို့လို့တိုက်ရိုက်ခေါ်သုံးလို့ရ
Math::add(5, 3); // class အတွက် instance တည်ဆောက်စရာမလို
```

Non-static

```
class Calculator {
   public function add($a, $b) {
      return $a + $b;
   }
}

// instance အရင်တည်ဆောက်ရန်လို

$calc = new Calculator();
$calc->add(5, 3);
```

data သိမ်းလိုက်တဲ့အခါ table view နဲ့ပြပေးတဲ့ list စာမျက်နှာကိုပဲသွားပါမယ်။ ဒီလိုအလိုအလျောက်ခေါ်ဆောင်သွားပေးတာကို redirect လုပ်ပေးတယ်လို့သုံးနှုန်းပါတယ်။ CreateTransaction.php ကိုဖွင့်ပါ။ Ctrl + P နှိပ်ပြီး CreateTransaction ရိုက်ထည့်ပြီးဖွင့်လို့ရပါတယ်။ ဒါမှမဟုတ် Sidebar မှာတွေ့ရတဲ့ TransactionResource/Pages ထဲက CreateTransaction.php ကိုနှိပ်ဖွင့်လို့လည်းရပါတယ်။ CreateTransaction class ထဲမှာ getRedirectUrl() နဲ့ getCreatedNotificationTitle() method နှစ်ခုထည့်ပေးပါ။

```
class CreateTransaction extends CreateRecord
{
   protected static string $resource = TransactionResource::class;

   protected function getRedirectUrl(): string {
      return $this->getResource()::getUrl('index');
   }

   protected function getCreatedNotificationTitle():?string{
      return 'Transaction successfully added.';
   }
}
```

getRedirectUrl() method က Transaction အသစ်တစ်ခု သိမ်းဆည်းပြီးတဲ့အခါ redirect လုပ်မယ့် URL ကို သတ်မှတ်ပေးတာဖြစ်ပါတယ်။ $this->getResource()::getUrl('index') က Transaction list page (index page) ကို redirect လုပ်မယ်လို့ သတ်မှတ်ထားတာပါ။ ဥပမာ - Transaction အသစ်တစ်ခု create လုပ်ပြီးရင် /transactions URL ကို redirect လုပ်သွားမှာဖြစ်ပါတယ်။

getCreatedNotificationTitle() method က Transaction အသစ်တစ်ခု သိမ်းဆည်းပြီးတဲ့အခါ ပြမယ့် notification ကို သတ်မှတ်ပေးပါတယ်။?string ဆိုတာက null လည်း return ပြန်နိုင်တယ်၊ string လည်း return ပြန်နိုင်တယ်ဆိုတဲ့ သဘောပါ။ ဒီ method နှစ်ခုလုံးက မိခင် class ဖြစ်တဲ့ CreateRecord ထဲက default method တွေကို override လုပ်ထားတာပါ။

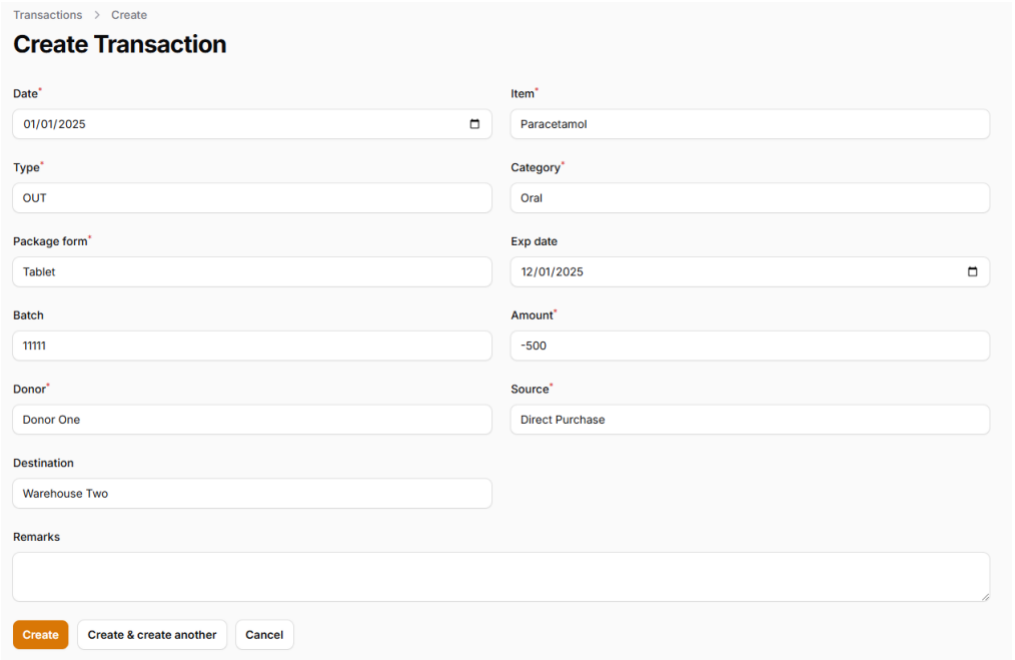
6.4 Redirect after update

Edit ပြီးလို့ save တဲ့အခါမှာလည်း table view ကိုပြန်ပြချင်ပါတယ်။ EditTransaction.php ကိုဖွင့်ပါ။ ခုနကလိုပဲ getRedirectUrl() နဲ့ edit လုပ်တာမို့လို့ getSavedNotificationTitle() ထည့်ပေးပါ။

```
class EditTransaction extends EditRecord
{
   protected static string $resource = TransactionResource::class;
```

```
    protected function getHeaderActions(): array
    {
        return [
            Actions\DeleteAction::make(),
        ];
    }

    protected function getRedirectUrl(): string {
        return $this->getResource()::getUrl('index');
    }

    protected function getSavedNotificationTitle():?string{
        return 'Transaction successfully updated.';
    }
}
```

save လုပ်ပြီးပိတ်ပါ။ **Create Transaction** နှိပ်ပြီး **Type** မှာ **OUT** ၊ **amount -500** နဲ့ **Destination** မှာ **Warehouse Two** ထားပြီး ကျန်တာအရင်အတိုင်းထည့်ကြည့်ပါ။ အရင်ဖြည့်ထားတဲ့ပစ္စည်းနဲ့ ကျန်တဲ့ **data** အားလုံးတူအောင်ဂရုစိုက်ပါ။ **Create** နှိပ်ပါ။



Notification ပေးထားသလိုပြပြီး **table view** ကိုရောက်သွားပါလိမ့်မယ်။ ပစ္စည်းတစ်မျိုးတည်း အဝင်တစ်ကြောင်း၊ အထွက်တစ်ကြောင်း သိမ်းပြီးပါပြီ။

7 Form Components

Data သွင်းတဲ့ **form field** တွေကို အသေးစိတ်ဆက်ပြင်ပါမယ်။ **TransactionResource.php** ကိုဖွင့်ပါ။

--generate flag နဲ့ လုပ်ခဲ့တဲ့အတွက် **form function** ထဲအလိုအလျောက်ထည့်ပေးထားတဲ့ **form field** တွေကို လိုအပ်ချက်နဲ့ကိုက်ညီအောင် ပြင်ပေးဖို့လိုပါတယ်။ ပထမဆုံးတွေ့ရမှာ **DatePicker** ပါ။

```
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\DatePicker::make('date')
                ->required(),
            // ...
        ]);
}
```

ရက်စွဲတွေရွေးချယ်နိုင်မယ့် **form component** ဖြစ်ပါတယ်။ **required()** ပါတာကြောင့် **form** ဖြည့်ရင်ချန်ခဲ့လို့မရပါဘူး။ **->native(false)** တစ်ကြောင်းထပ်ထည့်ပေးပါ။

```
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\DatePicker::make('date')
                ->required()
                ->native(false),
            // ...
        ]);
}
```

Save ပြီး **browser** မှာ **Create Transaction** နှိပ်ပါ။ **Date** ရွေးတဲ့ပုံစံပြောင်းသွားပါလိမ့်မယ်။ အရင်က **icon** ကိုနှိပ်မှ **calendar** ပေါ်တဲ့ပုံစံဟာ **Windows** စနစ်ရဲ့ **native date picker** ပါ။ **native style** ကို **false** ပေးလိုက်တဲ့အတွက် ဘယ်စနစ်မှာသုံးသုံး **calendar** တပုံစံတည်းဖြစ်နေပါမယ်။

ဒီ **form** ဟာ ဆေးပစ္စည်းအဝင်စာရင်းပဲမှတ်လို့ရတာကြောင့် **type** ဟာ အမြဲတမ်း **IN** ဖြစ်မှာပါ။ ဒါကြောင့် **type form field** ကိုမလိုတော့တဲ့အတွက် **select** လုပ်ပြီးဖျက်ပါ။

item form field ကို **dropdown** လိုချင်တဲ့အတွက် **Select** ပြောင်းပေးပါ။ **maxLength(255)** ဆိုတာ စာလုံး ၂၅၅ လုံးထက်ပိုဖြည့်လို့မရအောင် ကာကွယ်ထားတာပါ။ **dropdown** မှာ မလိုတော့တဲ့အတွက်ဖျက်ပေးပါ။

->options([]) **array** ထပ်ထည့်ပြီး **item option** နှစ်မျိုးထည့်ပေးပါ။

placeholder() က **dropdown** မှာ ဘာမှမရွေးရင် ပြပေးမယ့်စာသားဖြစ်ပါတယ်။

native(false) က **date picker** မှာလိုပဲ **native style** မသုံးချင်လို့ပါ။

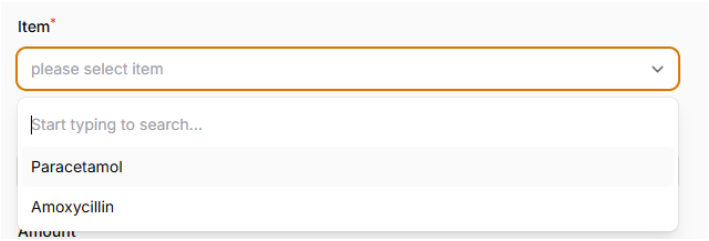
searchable() က **option** တွေကို **search** ပေးလုပ်ပါတယ်။

preload() ထည့်ထားရင် **search** မလုပ်ပေမယ့် **option** တွေပြပေးပြီး **search** လုပ်လိုက်မှ **result** ကိုဆက်ပြပါတယ်။

```
public static function form(Form $form): Form
{
    return $form
        ->schema([
            Forms\Components\DatePicker::make('date')
                ->required()
```

```
            ->native(false),
        Forms\Components\Select::make('item')
            ->required()
            ->options([
                                1 => 'Paracetamol',
                                2 => 'Amoxycillin',
            ])
            ->placeholder('please select item')
            ->native(false)
            ->searchable()
            ->preload(),
    ]);
}
```

Save ပြီး **browser** မှာ **refresh** ကြည့်ပါ။



Select component မှာ **options** တွေကို **array** အနေနဲ့ သတ်မှတ်ထားတာဖြစ်ပါတယ်။

```
->options([
  1 => 'Paracetamol',
  2 => 'Amoxycillin',
])
```

php array တွေမှာ **key : value pair** တွေနဲ့တည်ဆောက်ထားပါတယ်။ ဒီနမူနာမှာ **Array key** တွေက 1 နဲ့ 2 ဖြစ်ပြီး **database** မှာ သိမ်းမယ့် **value** တွေ ဖြစ်ပါတယ်။ **Array value** တွေက **'Paracetamol'** နဲ့ **'Amoxycillin'** ဖြစ်ပြီး **user interface** မှာမျက်စိနဲ့မြင်ရမယ့်တန်ဖိုးတွေ ဖြစ်ပါတယ်။

ဥပမာ - **user** က **Paracetamol** ကို ရွေးလိုက်ရင် **database** ထဲမှာ 1 ဆိုတဲ့ တန်ဖိုးကို သိမ်းသွားမှာ ဖြစ်ပါတယ်။

array ထဲမှာ **item** တွေ ထပ်ထည့်ချင်ရင် ဒီပုံစံအတိုင်း **key => value format** နဲ့ ထပ်ထည့်လို့ရပါတယ်။

Category form field ကိုလည်း **Select component** ပြောင်းပြီး **option** တွေထည့်ပေးပါ။ **option** နည်းတဲ့အတွက် **searchable** နဲ့ **preload** မထည့်တော့ပါဘူး။

```
        Forms\Components\Select::make('category')
            ->required()
            ->options([
                                1 => 'Oral',
                                2 => 'Injection',
                                3 => 'Topical',
```

```
                4 => 'Equipment',
                5 => 'Supplies',
        ])
        ->placeholder('please select category')
        ->native(false),
```

Package Form ကို **Select** ပြောင်းပါ။ **option** ထည့်ပေးပါ။

```
    Forms\Components\Select::make('package_form')
        ->required()
        ->options([
                1 => 'Tablet',
                2 => 'Capsule',
                3 => 'Ampoule',
                4 => 'Bottle',
                5 => 'Tube',
                6 => 'Sheet',
                7 => 'Piece',
                8 => 'Pair',
        ])
        ->placeholder('please select package form')
        ->native(false),
```

Exp date ကို နားလည်လွယ်အောင် **label()** ထည့်ပေးပါမယ်။ **expiry date** မရှိတဲ့ပစ္စည်းတွေလည်းပါနိုင်တဲ့အတွက် **required()** မထည့်ပါဘူး။

```
    Forms\Components\DatePicker::make('exp_date')
                ->label('Expiry Date')
        ->native(false),
```

Batch number က အလွတ်ရိုက်ထည့်ရတာမို့ **TextInput** အတိုင်းပဲဆက်ထားပါမယ်။ **label** ပြောင်းပေးပြီး **placeholder** ထည့်ပေးပါ။

```
    Forms\Components\TextInput::make('batch')
                ->label('Batch number')
                ->placeholder('please enter batch number')
        ->maxLength(255),
```

Amount field မှာ **placeholder** ထည့်ပါ။ ဒီ **form** ကို ဆေးပစ္စည်းအဝင်မှတ်တမ်းအတွက်ပဲ သုံးမှာမို့လို့ **minValue(1)** က "0" နဲ့ **minus** တန်ဖိုးတွေလာထည့်လို့မရအောင် ကာကွယ်ပေးပါတယ်။

```
    Forms\Components\TextInput::make('amount')
                ->placeholder('please enter amount')
        ->required()
        ->numeric()
        ->minValue(1),
```

Donor နဲ့ **Source** ကိုလည်းအောက်ပါအတိုင်းပြင်ပေးပါ။

```
        Forms\Components\Select::make('donor')
                            ->required()
          ->options([
                                1 => 'Donor One',
                                2 => 'Donor Two',
          ])
          ->placeholder('please select donor')
          ->native(false),

        Forms\Components\Select::make('source')
                            ->required()
          ->options([
                                1 => 'Direct Purchase',
                                2 => 'Warehouse One',
                                3 => 'Warehouse Two',
          ])
          ->placeholder('please select source')
          ->native(false),
```

ပစ္စည်းအဝင်မှတ်တမ်းပဲသိမ်းမှာမို့လို့ **destination** ကိုဖျက်ပါ။ **Remarks** ကိုတော့ သူ့အတိုင်းထားခဲ့ပါ။

အားလုံးပြီးရင် ဒီတိုင်းဖြစ်ပါမယ်။

```
  public static function form(Form $form): Form
  {
    return $form
      ->schema([
        Forms\Components\DatePicker::make('date')
          ->required()
          ->native(false),
        Forms\Components\Select::make('item')
          ->required()
          ->options([
            1 => 'Paracetamol',
            2 => 'Amoxycillin',
          ])
          ->placeholder('please select item')
          ->native(false)
          ->searchable()
          ->preload(),
        Forms\Components\Select::make('category')
          ->required()
          ->options([
            1 => 'Oral',
            2 => 'Injection',
```

```
            3 => 'Topical',
            4 => 'Equipment',
            5 => 'Supplies',
        ])
        ->placeholder('please select category')
        ->native(false),
    Forms\Components\Select::make('package_form')
        ->required()
        ->options([
            1 => 'Tablet',
            2 => 'Capsule',
            3 => 'Ampoule',
            4 => 'Bottle',
            5 => 'Tube',
            6 => 'Sheet',
            7 => 'Piece',
            8 => 'Pair',
        ])
        ->placeholder('please select package form')
        ->native(false),
    Forms\Components\DatePicker::make('exp_date')
        ->label('Expiry Date')
        ->native(false),
    Forms\Components\TextInput::make('batch')
        ->label('Batch number')
        ->placeholder('please enter batch number')
        ->maxLength(255),
    Forms\Components\TextInput::make('amount')
        ->placeholder('please enter amount')
        ->required()
        ->numeric()
        ->minValue(1),
    Forms\Components\Select::make('donor')
        ->required()
        ->options([
            1 => 'Donor One',
            2 => 'Donor Two',
        ])
        ->placeholder('please select donor')
        ->native(false),
    Forms\Components\Select::make('source')
        ->required()
        ->options([
            1 => 'Direct Purchase',
```

```
                2 => 'Warehouse One',
                3 => 'Warehouse Two',
            ])
            ->placeholder('please select source')
            ->native(false),
        Forms\Components\Textarea::make('remarks')
            ->columnSpanFull(),
    ]);
}
```

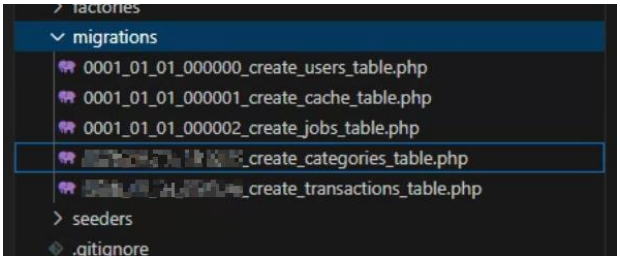
7.1 Category

Transaction form ထဲက **dropdown** တွေဟာ နားလည်ရလွယ်အောင် **option** တွေထည့်ထားတာပါ။ တကယ်အသုံးပြုသူတွေကြားမှာ **dropdown option** တွေဟာ ပုံသေမရှိဘဲ လိုတိုးပိုလျော့အမြဲရှိနေတတ်ပါတယ်။ တနည်းအားဖြင့် **static** မဟုတ်ဘဲ **dynamic** ဖြစ်ကြပါတယ်။ ဒါကြောင့် **dropdown option** တွေအတွက် **database table** တည်ဆောက်ပေးရပြီး **result** ကို **database table** ထဲကနေပဲ ခေါ်ပြပါမယ်။ **option** လိုတိုးပိုလျော့လုပ်ဖို့လည်း **user** ကို **form** သပ်သပ်ဆောက်ပေးထားဖို့လိုပါတယ်။ ဒါမှ **option** ပြင်ဆင်ဖို့လိုတိုင်း **developer** ကိုလိုက်ရှာရတဲ့ဒုက္ခကင်းဝေးစေမှာဖြစ်ပါတယ်။

Category dropdown အတွက် **table** စဆောက်ကြည့်ပါမယ်။ **table** ဆောက်တာက **migration** ဖြစ်ပြီး **table** ဆောက်ပြီဆိုတာနဲ့ **model** ပါတွဲဆောက်ပေးရပါမယ်။

```
php artisan make:model Category -m
```

migrate နဲ့ **model** နှစ်ဖိုင်ရလာပါမယ်။ **\database\migrations\ folder** ကိုဖွင့်ကြည့်ရင် **create_categories_table.php** က **create_transactions_table.php** နောက်မှာရှိနေပါမယ်။ **create_categories_table.php** ကို **right click > rename** နှိပ်ပြီး **transactions_table** ဖိုင်ထက်စောတဲ့ရက်စွဲတစ်ခုပြောင်းပေးပါ။ ဘယ်ဂဏန်းပဲပြောင်းပြောင်း **categories** က **transactions** အပေါ်ရောက်သွားရင်ရပါပြီ။



categories migrate ဖိုင်ကိုဖွင့်ပါ။ **string** အမျိုးအစား **name column** ထပ်ထည့်ပါ။

```
Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
```

Category.php model ဖိုင်မှာ **$fillable** ထည့်ပေးပါ

```
class Category extends Model
{
```

```
    protected $fillable = ['name'];
}
```

transactions migrate ဖိုင်ကိုဖွင့်ပါ။ **category column** ကို **foreignId** ပြောင်းပေးပါ။ အနီရောင်ခြယ်ထားတာကို အစိမ်းရောင်ခြယ်ထားတာနဲ့အစားထိုးရမှာဖြစ်ပါတယ်။

```
Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item');
        $table->string('category'); // foreignId အမျိုးအစားပြောင်းရန်

        $table->foreignId('category_id')->constrained('categories')->cascadeOnDelete();
        // အခြား column များ

    });
```

Ctrl + P နှိပ်ပြီး **Transaction.php model** ဖိုင်ကိုဖွင့်ပါ။ **$fillable** ထဲမှာလည်း လိုက်ပြောင်းပေးရပါမယ်။

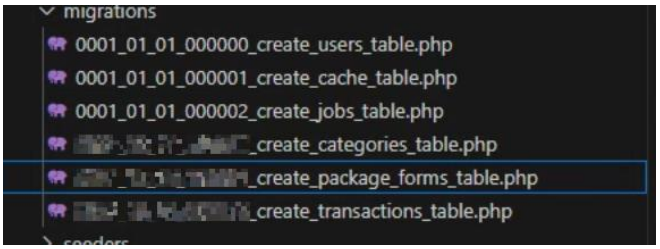
```
    protected $fillable = [
        'date',
        'type',
        'item',
        'category',
        'category_id',
        // အခြား property များ

    ];
```

7.2 Package Form

Package Form dropdown အတွက် **table** ပြုလုပ်ပါမယ်။

```
php artisan make:model PackageForm -m
```

migrate နဲ့ **model** နှစ်ဖိုင်ရလာပါမယ်။ **\database\migrations\ folder** ထဲမှာ **create_package_forms_table.php** ကို **create_transactions_table.php** ဖိုင်ထက်စောတဲ့ရက်စွဲတစ်ခုပြောင်းပေးပါ။ **transactions_table** ဖိုင်က အောက်ဆုံးမှာဖြစ်နေရင်ရပါပြီ။



package_forms migrate ဖိုင်ကိုဖွင့်ပါ။ **string** အမျိုးအစား **name column** ထပ်ထည့်ပါ။

```
Schema::create('package_forms', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
```

PackageForm.php model ဖိုင်မှာ $fillable ထည့်ပေးပါ

```
class PackageForm extends Model
{
   protected $fillable = ['name'];
}
```

transactions migrate ဖိုင်ကိုဖွင့်ပါ။ package_form column ကို foreignId ပြောင်းပေးပါ။

```
Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item');
        $table->foreignId('category_id')->constrained('categories')->cascadeOnDelete();
        $table->string('package_form'); // foreignId အမျိုးအစားပြောင်းရန်

        $table->foreignId('package_form_id')->constrained('package_forms')->cascadeOnDelete();
        // အခြား column များ

    });
```

Transaction.php model ဖိုင်ကိုဖွင့်ပါ။ $fillable ထဲမှာပြောင်းပေးပါ

```
   protected $fillable = [
      'date',
      'type',
      'item',
      'category_id',
      'package_form',
      'package_form_id',
      // အခြား property များ

   ];
```

7.3 Donor

Donor dropdown အတွက် table ပြုလုပ်ပါမယ်။

```
php artisan make:model Donor -m
```

migrate နဲ့ **model** နှစ်ဖိုင်ရလာပါမယ်။ **\database\migrations\ folder** ထဲမှာ **create_donors_table.php** ကို **create_transactions_table.php** ဖိုင်ထက်စောတဲ့ရက်စွဲတစ်ခုပြောင်းပေးပါ။

donors migrate ဖိုင်ကိုဖွင့်ပါ။ **string** အမျိုးအစား **name column** ထပ်ထည့်ပါ။

```
Schema::create('donors', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
```

Donor.php model ဖိုင်မှာ **$fillable** ထည့်ပေးပါ

```
class Donor extends Model
{
   protected $fillable = ['name'];
}
```

transactions migrate ဖိုင်ကိုဖွင့်ပါ။ **donor column** ကို **foreignId** ပြောင်းပေးပါ။

```
Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item');
        $table->foreignId('category_id')->constrained('categories')->cascadeOnDelete();
        $table->foreignId('package_form_id')->constrained('package_forms')->cascadeOnDelete();
        $table->string('donor'); // foreignId အမျိုးအစားပြောင်းရန်

        $table->foreignId('donor_id')->constrained('donors')->cascadeOnDelete();
        // အခြား column များ

    });
```

Transaction.php model ဖိုင်မှာလည်းပြောင်းပေးပါ။

```
   protected $fillable = [
      'date',
      'type',
      'item',
      'category_id',
      'package_form_id',
      'exp_date',
      'batch',
      'amount',
      'donor',
      'donor_id',
```

```
    // အခြား property များ
];
```

7.4 Source Warehouse

Source Warehouse အတွက် **table** ပြုလုပ်ပါမယ်။

```
php artisan make:model Warehouse -m
```

\database\migrations\ folder ထဲမှာ **create_warehouses_table.php** ကို **create_transactions_table.php** ဖိုင်ထက်စောတဲ့ရက်စွဲတစ်ခုပြောင်းပေးပါ။

warehouses migrate ဖိုင်ကိုဖွင့်ပါ။ **string** အမျိုးအစား **name column** ထပ်ထည့်ပါ။

```
Schema::create('warehouses', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
```

Warehouse.php model ဖိုင်မှာ **$fillable** ထည့်ပေးပါ

```
class Warehouse extends Model
{
   protected $fillable = ['name'];
}
```

transactions migrate ဖိုင်ကိုဖွင့်ပါ။ **source column** ကို **foreignId** ပြောင်းပေးပါ။

```
Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item');
        $table->foreignId('category_id')->constrained('categories')->cascadeOnDelete();
        $table->foreignId('package_form_id')->constrained('package_forms')->cascadeOnDelete();
        $table->foreignId('donor_id')->constrained('donors')->cascadeOnDelete();
        $table->string('source'); // foreignId အမျိုးအစားပြောင်းရန်

        $table->foreignId('source')->constrained('warehouses')->cascadeOnDelete();
        // အခြား column များ

    });
```

source column နာမည်ကိုမပြောင်းဘဲဆက်သုံးတာမို့လို့ **Transaction.php model** ဖိုင်မှာ **update** လုပ်စရာမလိုတော့ပါဘူး။

7.5 Item

Item dropdown အတွက် **table** ပြုလုပ်ပါမယ်။

```
php artisan make:model Item -m
```

\database\migrations\ folder ထဲမှာ **create_items_table.php** ကို **create_transactions_table.php** ဖိုင်ထက်စောတဲ့ရက်စွဲတစ်ခုပြောင်းပေးပါ။ ဒါပေမယ့် **categories_table** ဖိုင်ထက်တော့နောက်ကျဖို့လိုပါတယ်။

items migrate ဖိုင်ကိုဖွင့်ပါ။ **string** အမျိုးအစား **name column** ထပ်ထည့်ပါ။ သူ့ဆီမှာတော့ **category** ပါတွဲမှတ်ချင်တဲ့အတွက် **category_id foreignId column** ပါထည့်ပေးပါ။

```
Schema::create('items', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->foreignId('category_id')->constrained('categories')->cascadeOnDelete();
        $table->timestamps();
    });
```

Item.php model ဖိုင်မှာ **$fillable** ထည့်ပေးပါ

```
class Item extends Model
{
   protected $fillable = [
                            'name',
                            'category_id',
                          ];
}
```

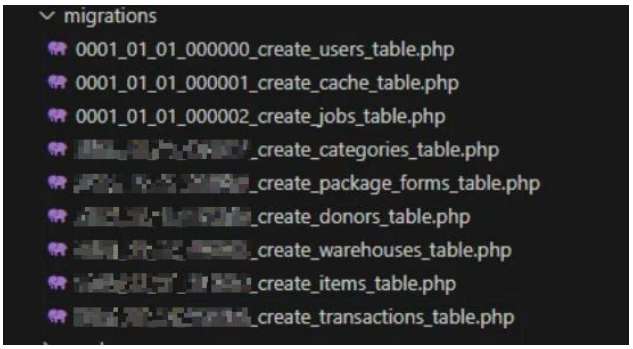
transactions migrate ဖိုင်ကိုဖွင့်ပါ။ **item column** ကို **foreignId** ပြောင်းပေးပါ။

```
Schema::create('transactions', function (Blueprint $table) {
        $table->id();
        $table->date('date');
        $table->string('type');
        $table->string('item'); // foreignId အမျိုးအစားပြောင်းရန်

        $table->foreignId('item_id')->constrained('items')->cascadeOnDelete();
        // အခြား column များ

    });
```

Transaction.php model ဖိုင်မှာလည်းပြောင်းပေးပါ။

```
   protected $fillable = [
     'date',
     'type',
     'item',
     'item_id',
     // အခြား property များ

   ];
```

ဖိုင်တွေအားလုံး **save** ပြီးပိတ်လို့ရပါပြီ။ **transactions migrate** ဖိုင်ဟာ နောက်ဆုံးမှာရှိနေရပါမယ်။ **items migrate** ဖိုင်ဟာ **categories** ရဲ့အောက်မှာရှိနေရပါမယ်။



ကျွန်တော်တို့ပြင်နေတဲ့ **migrate** ဖိုင်တွေက ဖိုင်ကိုပဲပြင်တာဖြစ်ပြီး တကယ့် **database table** မှာ ဘာမှသက်ရောက်မှုမရှိသေးပါဘူး။ လောလောဆယ် **database** ထဲမှာ **transactions table** ရှိနေပြီးသားပါ။ ရှိနေလို့လည်း **form** ဆောက်ပြီးအစမ်းသိမ်းကြည့်လို့ရတာပါ။ ဒါပေမယ့် သူမှုရင်း **migrate file** မှာသွားပြင်ပြီး **php artisan migrate command** သွား **run** ရင် ဘာမှသက်ရောက်မှုမရှိပါဘူး။ ရှိပြီးသား **table** ကိုပြန်ဖျက်ပြီးမှ **migrate file** ပြင်ထားပြီးသားဟာကို **php artisan migrate run** လို့ရမှာပါ။ ဒါကို **migrate rollback** လုပ်တယ်လို့ခေါ်ပါတယ်။ အောက်ပါ **command** ကို **run** ပါ။ အစမ်းဖြည့်ထားတဲ့ **record** နှစ်ခုကတော့ ပျက်သွားမှာဖြစ်ပါတယ်။

```
php artisan migrate:rollback
```

ပြီးမှ **migrate** ပြန်လုပ်ပါ

```
php artisan migrate
```

7.6 Link form with database

7.6.1 Laravel Eloquent ORM

Eloquent ORM ဆိုတာ **Laravel** မှာ **Database** နဲ့ ချိတ်ဆက်ပြီး **data** တွေကို **object oriented programming style** နဲ့ စီမံခန့်ခွဲနိုင်တဲ့စနစ်တစ်ခုဖြစ်ပါတယ်။ နေရာတိုင်းမှာ **RAW SQL** တွေရေးနေစရာမလိုတော့ဘဲ **eloquent ORM** နဲ့ပဲအမြန် **query** လုပ်နိုင်ပါတယ်။ **Eloquent ORM** မှာပါဝင်တာတွေက

7.6.2 Model

Database table တစ်ခုချင်းစီအတွက် **Model class** တစ်ခုစီ သတ်မှတ်ပေးရပါတယ်။ ဥပမာ - **users table** အတွက် **User model, products table** အတွက် **Product model** စသဖြင့်

```
class User extends Model
{
   protected $table = 'users';
}
```

7.6.3 Relationships

Table တွေကြား **relationship** တွေကို အလွယ်တကူ သတ်မှတ်နိုင်ပါတယ်။ **One to One, One to Many, Many to Many** စတဲ့ **relationships** မျိုးစုံ ရှိပါတယ်

```
class User extends Model
{
   public function posts()
   {
      return $this->hasMany(Post::class);
   }
}
```

7.6.4 Query Building

SQL query တွေကို PHP syntax နဲ့ရေးနိုင်ပါတယ်။ ဥပမာ

```
// Select all users
$users = User::all();

// Where clause
$activeUsers = User::where('status', 'active')->get();

// Order by
$sortedUsers = User::orderBy('name', 'asc')->get();
```

7.6.5 Mass Assignment

Multiple records တွေကို တစ်ခါတည်း insert လုပ်နိုင်ပါတယ်

```
User::create([
   'name' => 'John',
   'email' => 'john@example.com'
]);
```

7.6.6 Accessors & Mutators

Data တွေကို get/set လုပ်တဲ့အခါ format ပြောင်းနိုင်ပါတယ်

```
// Accessor
public function getNameAttribute($value)
{
   return ucfirst($value);
}

// Mutator
public function setPasswordAttribute($value)
{
   $this->attributes['password'] = bcrypt($value);
}
```

7.6.7 Soft Deletes

Record တွေကို အပြီးအပိုင်မဖျက်ပဲ အချိန်သတ်မှတ်ချက်နဲ့ ဖျက်ထားနိုင်ပါတယ်

```
use SoftDeletes;


protected $dates = ['deleted_at'];
```

Eloquent ORM ကိုသုံးခြင်းအားဖြင့် **SQL query** တွေကို **PHP code** နဲ့ ရေးနိုင်တာကြောင့် **Code** ပိုမိုသပ်ရပ်ပါတယ်။ **Relationship** တွေကို လွယ်ကူစွာ စီမံခန့်ခွဲနိုင်ပြီး **Security features** တွေ **built-in** ပါဝင်ပါတယ်။

အားနည်းချက်တွေကတော့ **Performance** အနည်းငယ်ကျနိုင်တာကြောင့် **Complex query** တွေအတွက် **raw SQL** က ပိုသင့်တော်ပါတယ်။

Eloquent ကို သုံးဖို့အတွက် အခြေခံ လိုအပ်ချက်တွေကတော့

Model class တည်ဆောက်ခြင်း

```
php artisan make:model User
```

Table name နဲ့ **fillable fields** တွေ သတ်မှတ်ခြင်း

```
protected $table = 'users';
protected $fillable = ['name', 'email'];
```

Relationships တွေ သတ်မှတ်ခြင်း

CRUD operations တွေ လုပ်ဆောင်ခြင်း

စတာတွေပဲဖြစ်ပါတယ်။

7.7 Relationships

Laravel မှာ **common database relationships** အမျိုးအစား (၄) မျိုးရှိပါတယ်:

7.7.1 One to One

```
// User model
public function profile()
{
   return $this->hasOne(Profile::class);
}
```

7.7.2 One to Many

```
// User model
public function posts()
{
   return $this->hasMany(Post::class);
}
```

7.7.3 Many to Many

```
// User model
public function roles()
{
   return $this->belongsToMany(Role::class);
}
```

7.7.4 Has Many Through

```
// Country model
public function posts()
{
   return $this->hasManyThrough(Post::class, User::class);
}
```

Relationship အသုံးပြုပုံ

```
// Get user's posts
$user = User::find(1);
$posts = $user->posts;

// Get post's user
$post = Post::find(1);
$user = $post->user;
```

TransactionResource.php ကိုပြန်ဖွင့်ပါ။ item field ကို update လုပ်ပေးပါ။

```
            Forms\Components\Select::make('item')
            Forms\Components\Select::make('item_id')
               ->required()
               ->options([
                  1 => 'Paracetamol',
                  2 => 'Amoxycillin',
               ])
               ->relationship(name: 'item', titleAttribute: 'name')
               ->createOptionForm([
                  Forms\Components\TextInput::make('name')
                     ->required(),
                  Forms\Components\Select::make('category_id')
                     ->required()
                     ->relationship(name: 'category', titleAttribute: 'name'),
               ])
               ->placeholder('please select item')
               ->native(false)
```

```
            ->searchable()
            ->preload(),
```

relationship() Transaction model နဲ့ **Item model** ကိုချိတ်ဆက်ပေးတဲ့ **relationship** ကိုခေါ်သုံးပြီး **Item table** ထဲက **data** တွေကို **dropdown option** ထဲပြပေးတာဖြစ်ပါတယ်။ **name:** က **relationship** ရဲ့နာမည်ဖြစ်ပြီး **titleAttribute:** က **item table** ရဲ့**name column** ကိုယူမယ်လို့ဆိုလိုပါတယ်။

createOptionForm() က **item** ရွေးစရာမလုံလောက်လို့ အသစ်ထပ်ဖြည့်ချင်တဲ့အခါ **dropdown** ကနေပဲ အသစ်ဖြည့်လို့ရအောင်လုပ်ပေးတာပါ။အပေါင်းလက္ခဏာ (+) နှိပ်လိုက်ရင် **modal dialog** ပွင့်လာပြီး **item name** ရိုက်ထည့်စရာ **TextInput** နဲ့ **Category** ရွေးချယ်စရာ **select dropdown** တစ်ခုထည့်ထားပါတယ်။ **Item Select** မှာကော ၊ **Create item modal** ထဲက **Category select** မှာကော **relationship method** ကနေ **option** တွေခေါ်ပြပေးတာဖြစ်တဲ့အတွက် **Transaction.php** နဲ့ **Item.php model class** နှစ်ခုစလုံးမှာ **relationship** တွေသွားထည့်ပေးရပါမယ်။

Transaction.php model class ကိုဖွင့်ပါ။ **item relationship method** ထည့်ပေးပါ။ **BelongsTo class** ကို **right click > import class** လုပ်ပေးပါ။

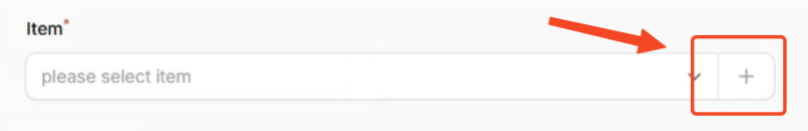
```
class Transaction extends Model
{
   protected $fillable = [
      'date',
      'type',
      'item_id',
      'category_id',
      'package_form_id',
      'exp_date',
      'batch',
      'amount',
      'donor_id',
      'source',
      'destination',
      'remarks'
   ];

   public function item(): BelongsTo
   {
      return $this->belongsTo(Item::class);
   }
}
```

Item.php model class ကိုဖွင့်ပြီး **category relationship** ထည့်ပေးပါ။

```
class Item extends Model
{
   protected $fillable = [
      'name',
      'category_id',
```

```
    ];

    public function category(): BelongsTo
    {
        return $this->belongsTo(Category::class);
    }
}
```

Browser မှာ **refresh** လုပ်ကြည့်ပါ။ **Item Select** ကိုရွေးလိုက်ရင်ဘာမှမရှိသေးပေမယ့် ညာဘက်ထောင့်မှာ (+) လက္ခဏာကိုနှိပ်လိုက်ရင် **item** အသစ်ဖန်တီးနိုင်မယ့် **modal dialog** ပွင့်လာပါလိမ့်မယ်။



modal dialog က **category select** မှာရွေးစရာမရှိသေးပါဘူး။ သူ့ကိုလည်း **createOptionForm() method** နဲ့ **category** ဖန်တီးဖို့ခလုတ်ထည့်သွင်းပေးပါမယ်။ ပြီးရင် **modal dialog** နှစ်ခုဆက်တိုက်မို့လို့ ရှုပ်ထွေးမှုမဖြစ်ရအောင် **modal heading** လေးတွေပါသတ်မှတ်ပေးပါ။

```
            Forms\Components\Select::make('item_id')
                ->required()
                ->relationship(name: 'item', titleAttribute: 'name')
                ->createOptionForm([
                    Forms\Components\TextInput::make('name')
                        ->required(),
                    Forms\Components\Select::make('category_id')
                        ->required()
                        ->relationship(name: 'category', titleAttribute: 'name')
                        ->createOptionForm([
                            Forms\Components\TextInput::make('name')
                                ->required(),
                        ])
                        ->createOptionModalHeading('Create New Category'),
                ])
                ->createOptionModalHeading('Create New Item')
                ->placeholder('please select item')
                ->native(false)
                ->searchable()
                ->preload(),
```

ဒါဆို **Category** တစ်ခုအရင်ထည့်ကြည့်ပါ။ ပြီးမှ **item** တစ်ခုထပ်ထည့်ကြည့်ပါ။ **Transaction form** မှာ **Item** ရွေးစရာ **option** တစ်ခုရှိလာပါပြီ။

Transaction form ထဲမှာရှိပြီးသား အရင်လုပ်ထားတဲ့ **Category select** ကို **update** လုပ်ပေးပါ။ **option()** နေရာမှာ **relationship()** သုံးလိုက်တဲ့အတွက် **Transaction model** မှာလည်း **category() belongsTo relationship** တစ်ခုရေးပေးရပါမယ်။ သူ့အပေါ်က **Item** ကိုရွေးလိုက်ရင် **category select** ကိုပါအလိုအလျောက်ရွေးအောင်လုပ်ပေးမှာမို့လို့ **placeholder** ကိုလည်းပြောင်းပေးပါ။ **disabled()** ကတော့

select dropdown ကို **user** လာနှိပ်ရင်အလုပ်မလုပ်အောင် ပိတ်ထားတာပါ။ **disabled()** ဆိုရင်သူ့ထဲကတန်ဖိုးကို **database** ထဲမသိမ်းပေးတော့ဘူးမို့လို့ **dehydrated(true)** ပေးထားမှသိမ်းပေးပါတယ်။

TransactionResource.php

```
            Forms\Components\Select::make('category')
            Forms\Components\Select::make('category_id')
              ->required()
              ->options([
                1 => 'Oral',
                2 => 'Injection',
                3 => 'Topical',
                4 => 'Equipment',
                5 => 'Supplies',
              ])
              ->relationship(name: 'category', titleAttribute: 'name')
              ->placeholder('please select category')
              ->placeholder('please select item first')
              ->native(false)
              ->disabled()
              ->dehydrated(true),
```

Transaction.php

```
class Transaction extends Model
{
  protected $fillable = [
    'date',
    'type',
    'item_id',
    'category_id',
    'package_form_id',
    'exp_date',
    'batch',
    'amount',
    'donor_id',
    'source',
    'destination',
    'remarks'
  ];

  public function item(): BelongsTo
  {
    return $this->belongsTo(Item::class);
  }
```

```
    public function category(): BelongsTo
    {
        return $this->belongsTo(Category::class);
    }
}
```

Browser မှာ refresh လုပ်ကြည့်လိုက်ရင် Transaction form ရဲ့ category select ကို click နှိပ်ရင်အလုပ်မလုပ်တော့ပါဘူး။ ဒီနေရာမှာ သူ့အပေါ်က Item select ကိုရွေးလိုက်တာနဲ့ category မှာ အဲဒီ item ရဲ့ category ကိုအလိုအလျောက်ရွေးအောင်လုပ်ပါမယ်။

TransactionResource.php

```
            Forms\Components\Select::make('item_id')
                ->required()
                ->relationship(name: 'item', titleAttribute: 'name')
                ->createOptionForm([
                    Forms\Components\TextInput::make('name')
                        ->required(),
                    Forms\Components\Select::make('category_id')
                        ->required()
                        ->relationship(name: 'category', titleAttribute: 'name')
                        ->createOptionForm([
                            Forms\Components\TextInput::make('name')
                                ->required(),
                        ])
                        ->createOptionModalHeading('Create New Category'),
                ])
                ->createOptionModalHeading('Create New Item')
                ->placeholder('please select item')
                ->native(false)
                ->searchable()
                ->preload()
                ->live()
                ->afterStateUpdated(function(Set $set, $state){
                    $set('category_id', Item::find($state)->category->id ?? null);
                }),
```

live() က select field ကို reactive ဖြစ်အောင်လုပ်ပေးတာပါ။ user က တခုခုရွေးလိုက်တာနဲ့ ဖြစ်စေချင်တဲ့ action တွေ နောက်ကွယ်မှာ run သွားအောင်လုပ်ပေးပါတယ်။ dependent or cascading dropdown တွေမှာသုံးပါတယ်။

afterStateUpdated() မှာ $state parameter က လက်ရှိရွေးထားတဲ့တန်ဖိုးဖြစ်ပါတယ်။ Set object ဖြစ်တဲ့ $set parameter က တခြား form field တစ်ခုကို တန်ဖိုးသတ်မှတ်ပေးပါတယ်။ $set('category_id', Item::find($state)->category->id ?? null); အလုပ်လုပ်ပုံကတော့ တခြား form field ဖြစ်တဲ့ category_id ရဲ့တန်ဖိုးကို အလိုအလျောက်ဖြည့်ပေးဖို့ပါ။ ဘာတန်ဖိုးလဲဆိုတော့ Item select မှာရွေးလိုက်တဲ့တန်ဖိုး id နံပါတ်ကို $state parameter ကတဆင့် Item::find($state) နဲ့ရှာလိုက်ပါတယ်။ ရှာတွေ့တဲ့ Item object ရဲ့ category relationship ကတဆင့် id နံပါတ်ကို return ပြန်ပေးပါတယ်။ ရှာမတွေ့ရင်တော့ null တန်ဖိုးပဲ return ပြန်ပါမယ်။ ဖိုင်တွေ Save လုပ်ပြီး browser refresh ကြည့်ပါ။ Item ကိုရွေးလိုက်ရင် Category ပါအလိုအလျောက်ပြောင်းသွားပါလိမ့်မယ်။

Package Form select ကိုလည်း createOptionForm ထည့်ကြည့်ပါမယ်။ လုပ်ပြီးသားအဆင့်တွေမို့လို့ code ကိုပဲတိုက်ရိုက်ဖော်ပြပါမယ်။

TransactionResource.php

```
            Forms\Components\Select::make('package_form')
            Forms\Components\Select::make('package_form_id')
              ->required()
              ->options([
                1 => 'Tablet',
                2 => 'Capsule',
                3 => 'Ampoule',
                4 => 'Bottle',
                5 => 'Tube',
                6 => 'Sheet',
                7 => 'Piece',
                8 => 'Pair',
              ])
              ->relationship(name: 'packageForm', titleAttribute: 'name')
              ->createOptionForm([
                Forms\Components\TextInput::make('name')
                    ->required(),
              ])
              ->createOptionModalHeading('Create New Package Form')
              ->placeholder('please select package form')
              ->native(false),
```

Transaction.php

```
class Transaction extends Model
{
  protected $fillable = [
    'date',
    'type',
    'item_id',
    'category_id',
    'package_form_id',
    'exp_date',
    'batch',
    'amount',
    'donor_id',
    'source',
    'destination',
    'remarks'
  ];

  public function item(): BelongsTo
  {
```

```
    return $this->belongsTo(Item::class);
  }

  public function category(): BelongsTo
  {
    return $this->belongsTo(Category::class);
  }

  public function packageForm(): BelongsTo
  {
    return $this->belongsTo(PackageForm::class);
  }
}
```

Donor နဲ့ Source Warehouse အတွက်ပါတလက်စတည်း ဆက်လုပ်သွားပါမယ်။

TransactionResource.php

```
        Forms\Components\Select::make('donor_id')
          ->required()
          ->relationship(name: 'donor', titleAttribute: 'name')
          ->createOptionForm([
            Forms\Components\TextInput::make('name')
              ->required(),
          ])
          ->createOptionModalHeading('Create New Donor')
          ->placeholder('please select donor')
          ->native(false),
        Forms\Components\Select::make('source')
          ->required()
          ->relationship(name: 'sourceWarehouse', titleAttribute: 'name')
          ->createOptionForm([
            Forms\Components\TextInput::make('name')
              ->required(),
          ])
          ->createOptionModalHeading('Create New Source')
          ->placeholder('please select source')
          ->native(false),
```

Transaction.php

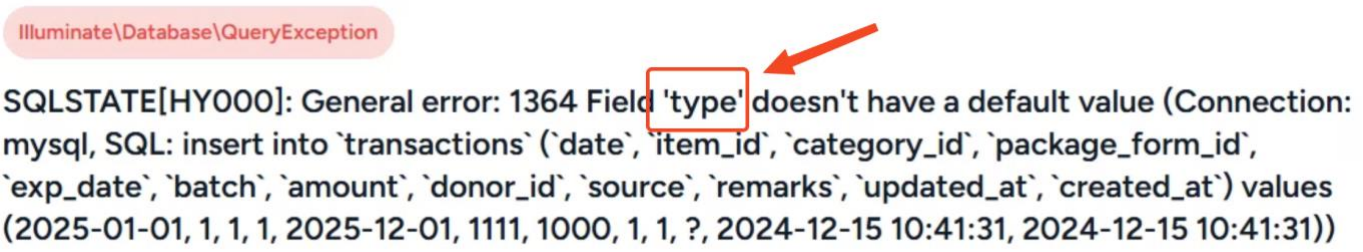
```
  public function donor(): BelongsTo
  {
    return $this->belongsTo(Donor::class);
  }
```

```
  public function sourceWarehouse(): BelongsTo
  {
    return $this->belongsTo(Warehouse::class, 'source');
  }
```

sourceWarehouse relationship က တခြား **BelongsTo** တွေနဲ့မတူတာကို သတိထားမိမှာပါ။ အရင် **relationship** တွေမှာ **model** နာမည်နဲ့ **relationship** နာမည်တူပါတယ်။ **sourceWarehouse relationship** မှာ နာမည်မတူတဲ့ **Warehouse model** ကိုသွားချိတ်တဲ့အတွက် လက်ရှိ **Transaction model** ထဲက **foreign key source** နဲ့ ချိတ်မယ်ဆိုတာကို သီးသန့်ကြေငြာပေးရပါတယ်။

7.8 Mutate Form Data

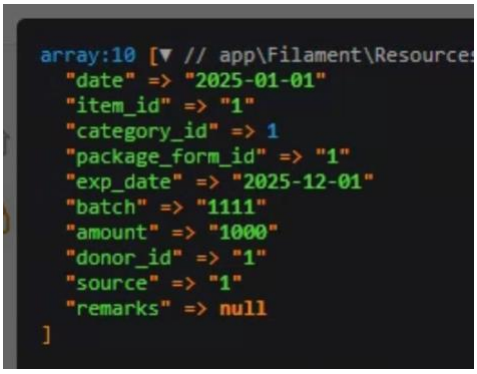
ဒီအဆင့်မှာ ပစ္စည်းအဝင်မှတ်တမ်းတစ်ခု အစမ်းထည့်သွင်းကြည့်ပါ။ **Create** နှိပ်လိုက်ရင် **error** တွေ့ရပါမယ်။



transactions table ထဲက **type column** ဟာ **nullable()** မဟုတ်လို့ **type** မှာဘာမှမထည့်ဘဲ **save** လိုက်ရင် **error** ပေါ်ပါလိမ့်မယ်။ **user** ကိုလည်း **type** မှာ **IN** ဆိုပြီး အမြဲမရိုက်ထည့်ခိုင်းချင်ပါဘူး။ ဒီ **form** ကိုသုံးပြီး **data** သိမ်းတိုင်း **type** မှာ **IN** ဆိုတဲ့တန်ဖိုးအလိုအလျောက်ထည့်သွင်းချင်ပါတယ်။ ဒါကို **mutate** လုပ်တယ်လို့ခေါ်ပါတယ်။ **CreateTransaction.php** ဖိုင်ကိုဖွင့်ပြီး **mutate function** ရေးပေးပါ။

```
class CreateTransaction extends CreateRecord
{
  protected static string $resource = TransactionResource::class;

  protected function getRedirectUrl(): string
  {
    return $this->getResource()::getUrl('index');
  }
  protected function getCreatedNotificationTitle(): ?string
  {
    return 'Transaction successfully added.';
  }

  protected function mutateFormDataBeforeCreate(array $data): array
  {
    dd($data);
    return $data;
  }
}
```

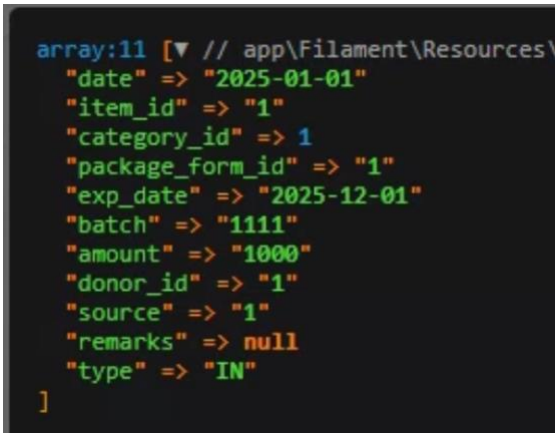
mutateFormDataBeforeCreate() function က **form** မှာဖြည့်လိုက်တဲ့တန်ဖိုးတွေကို **$data parameter** နဲ့သယ်လာပါတယ်။ **return $data;** မလုပ်ခင်မှာ **dump and die function** ဖြစ်တဲ့ **dd($data)** နဲ့ **variable dump** လုပ်ကြည့်ပါ။ **Save** လုပ်ပါ။ **browser** မှာ **error** ပေါ်နေတဲ့ **dialog** အပြင်ဘက်ကို **click** နှိပ်ပါ။ **error** ပြန်ပျောက်သွားပါလိမ့်မယ်။ **Create** ခလုတ်ထပ်နှိပ်ကြည့်ပါ။ **error** မပျောက်သေးရင် **refresh** လုပ်ပြီး **data** ပြန်ဖြည့်၊ **Create** ခလုတ်နှိပ်ပါ။ **form** က ဖြည့်လိုက်တဲ့တန်ဖိုးတွေကို **array** ပုံစံနဲ့တွေ့ရပါမယ်။



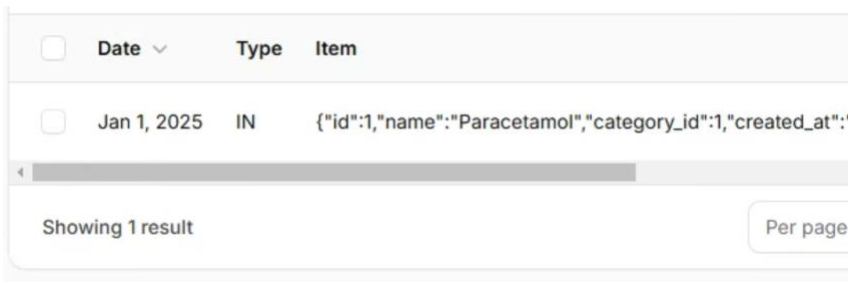
dd() မတိုင်ခင် နောက်တစ်ကြောင်းထည့်ကြည့်ပါ။ **$data array** ထဲကို **type** ဆိုတဲ့ **key** နဲ့ **IN** ဆိုတဲ့ **value** တစ်ခုထပ်ထည့်ပါမယ်။

```
protected function mutateFormDataBeforeCreate(array $data): array
{
    $data['type'] = 'IN';
    dd($data);
    return $data;
}
```

Save လုပ်ပြီး **dd dialog** အပြင်ဘက် **click** နှိပ်ပါ။ **dd** ပျောက်သွားရင် **Create** ခလုတ်ထပ်နှိပ်ကြည့်ပါ။ **type key value** တိုးလာပါလိမ့်မယ်။



dd($data); တစ်ကြောင်းပြန်ဖျက်ပြီး **Save** ပါ။ **browser** မှာ **dd dialog** အပြင်ဘက် **click** နှိပ်ပြီး **Create** ခလုတ်ထပ်နှိပ်ကြည့်ပါ။ **create** အောင်မြင်ပြီး **table view** ကိုရောက်ပါမယ်။



Item column စတာတွေမှာ object တစ်ခုလုံးရဲ့ data တွေပြပေးနေတာကို ပြင်ဖို့လိုပါတယ်။ TransactionResource.php ကိုဖွင့်ပါ။ public static function table() ကိုသွားပါ။ foreignId ပြောင်းလိုက်တဲ့ column တွေကို relationship + . + name format နဲ့ပြင်ရေးပေးပါ။

```
public static function table(Table $table): Table
{
    return $table
        ->columns([
            // ... ရှိပြီးသား column များ

            Tables\Columns\TextColumn::make('item.name')
                ->searchable(),
            Tables\Columns\TextColumn::make('category.name')
                ->searchable(),
            Tables\Columns\TextColumn::make('packageForm.name')
                ->searchable(),
            // ... ရှိပြီးသား column များ

            Tables\Columns\TextColumn::make('donor.name')
                ->searchable(),
            Tables\Columns\TextColumn::make('sourceWarehouse.name')
                ->searchable(),
            // ... ရှိပြီးသား column များ

        ])
        // ... အခြား properties

        ;
}
```

Save ပြီး refresh လုပ်ပါ။

ဒီလောက်ဆိုရင် ပစ္စည်းအဝင်မှတ်တမ်း Transaction CRUD အပိုင်းပြီးပါပြီ။ နောက်တပိုင်းမှာတော့ အဝင်အထွက်မှတ်တမ်းတွေကို SUM() နဲ့ aggregate လုပ်ပြီးပြပေးမယ့် စာမျက်နှာဖန်တီးပါမယ်။

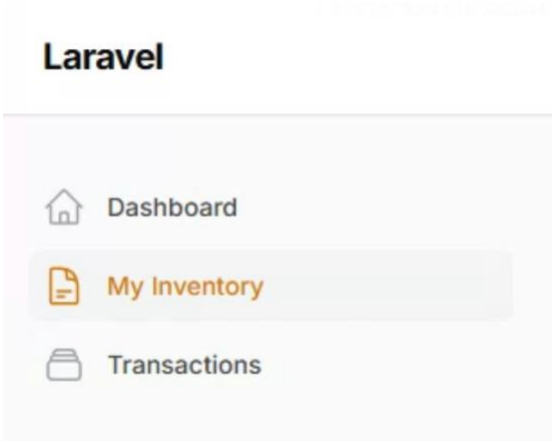
8 Creating Custom Page

ဆေးစတိုမှာ လက်ရှိပစ္စည်းအရေအတွက်ကို ဖော်ပြတဲ့ table က ပုံမှန် CRUD resource page လိုမဟုတ်ဘဲ aggregated data ပြဖို့ရယ်၊ custom action တွေလည်းထည့်ဖို့ရှိလို့ custom page အနေနဲ့ဖန်တီးပါမယ်။

page create command ကိုအောက်ပါအတိုင်း run ပါ

```
php artisan make:filament-page MyInventory
```

Which resource would you like to create this in? လို့မေးလာရင် blank အတိုင်းထားပြီး enter နှိပ်ပါ။ brower မှာ လာကြည့်ရင် My Inventory စာမျက်နှာသစ်ပေါ်လာပါမယ်။



MyInventory.php ကိုဖွင့်ပါ။ **custom page** မို့လို့ အထဲမှာဘာ **component** မှမရှိပါဘူး။ **table** နဲ့အတူ **table action** တွေပါထည့်အလုပ်လုပ်ခိုင်းဖို့ **HasTable** နဲ့ **HasForms trait** နှစ်ကိုထည့်ပေးပါ။

InteractsWithTable နဲ့ **InteractsWithForms** တွေကိုလည်း **use** လုပ်ပေးရပါမယ်။ လိုအပ်တဲ့ **class** တွေကို **right click > import class** လုပ်ပေးပါ။

```php
<?php

namespace App\Filament\Pages;

use Filament\Forms\Concerns\InteractsWithForms;
use Filament\Forms\Contracts\HasForms;
use Filament\Pages\Page;
use Filament\Tables\Contracts\HasTable;
use Filament\Tables\Concerns\InteractsWithTable;

class MyInventory extends Page implements HasTable, HasForms
{
  use InteractsWithTable, InteractsWithForms;
  //...
}
```

table ထည့်ဖို့အတွက် **table function** ရေးပေးပါ။ အဲဒီ **table** ကို **data** ပေးရမှာမို့လို့ အောက်မှာ **tableQuery function** တစ်ခုထပ်ရေးပေးပါ။ တစ်ခုသတိထားရမှာက **tableQuery** က **return hint Builder class** ကို **import** လုပ်တဲ့အခါ **Illuminate\Database\Eloquent\Builder;** ဖြစ်ရပါမယ်။ တခြား **class** တွေမရှိသေးရင်လည်း **right click > import class** လုပ်ပေးပါ။

```php
class MyInventory extends Page implements HasTable, HasForms
{
  use InteractsWithTable, InteractsWithForms;
  protected static ?string $navigationIcon = 'heroicon-o-document-text';

  protected static string $view = 'filament.pages.my-inventory';

  public function table(Table $table): Table
```

```
  {
    return $table
      ->query(self::tableQuery())
      ->columns([]);
  }

  private static function tableQuery(): Builder
  {
    return Transaction::query();
  }
}
```

Save လုပ်ပြီး **my-inventory.blade.php** ဖိုင်ကိုဆက်ဖွင့်ပါ။ **table function** လှမ်းခေါ်ပေးပါ။

```
<x-filament-panels::page>
{{ $this->table }}
</x-filament-panels::page>
```

save and refresh လုပ်ကြည့်ပါ။ **table view** တစ်ခုအကြမ်းထည်ပေါ်နေပါမယ်။ **table function** ထဲမှာ **column component** တွေမထည့်ရသေးလို့ **data** တွေမြင်ရမှာမဟုတ်သေးပါဘူး။

ဒီ **table** မှာ **aggregate data** တွေပြမှာမို့လို့ **Transaction model** အတွက် ပုံမှန် **table query** နဲ့မရဘဲ **customized query** ရေးပေးရပါမယ်။ **tableQuery() function** ထဲမှာအောက်ပါအတိုင်းဖြည့်ပေးပါ။

```
  private static function tableQuery(): Builder
  {
    return Transaction::query()
      ->selectRaw('min(id) as id, item_id, category_id, SUM(amount) as total_amount')
      ->havingRaw('SUM(amount) > 0')
      ->groupBy('item_id', 'category_id')
      ->orderBy('item_id');
  }
```

selectRaw() က **Laravel** ရဲ့ **Eloquent ORM** မသုံးဘဲ **Raw SQL query** သုံးမယ်လို့ကြေငြာပါတယ်။ **SUM(amount) as total_amount** က **amount column** တောက်လျောက် အပေါင်းအနုတ်တန်ဖိုးတွေကို **total_amount** ဆိုတဲ့ခေါင်းစဉ်အောက်မှာ ပေါင်းချလိုက်ပါမယ်။ **havingRaw()** ကတော့ ပေါင်းခြင်း 0 ထက်ပိုတဲ့တန်ဖိုးတွေကိုပဲ ယူသုံးပါမယ်။ ဆေးပစ္စည်းအားလုံးကုန်အောင်သုံးပြီးလို့ **balance** 0 ဖြစ်နေတာတွေကို ခေါ်မပြတော့ပါဘူး။ **item_id** နဲ့ **category_id** ကို **group** လုပ်လိုက်ခြင်းအားဖြင့် **amount column** ထဲကတန်ဖိုးတွေအကုန်ပေါင်းချလိုက်မှာမဟုတ်ဘဲ **item_id** နဲ့ **category_id** နှစ်ခုလုံးတူနေတဲ့ **row** တွေကို **group** လုပ်၊ တခါပေါင်းနဲ့သွားပါတယ်။ ဥပမာ **Paracetamol** နဲ့ **Oral** ပါနေတဲ့ **row** အားလုံး **group** လုပ်ချလိုက်ပြီး **amount** တန်ဖိုးအားလုံး ပေါင်းပေးပါတယ်။ အဲဒီနှစ်မျိုးအနက် တစ်မျိုးမတူလိုက်တာနဲ့ နောက်ထပ် **row** တစ်ခု ထပ် **group** လုပ်ပြီး ထပ်ပေါင်းပေးပါတယ်။ ဒီနည်းနဲ့ ဆေးစတိုမှာလက်ရှိပစ္စည်းအရေအတွက် ရရှိမှာဖြစ်ပါတယ်။

orderBy() က ပစ္စည်းစာရင်းကို **sort** လုပ်တာပါ။

ဥပမာ နမူနာပြထားတဲ့ **table** ဟာ **transaction table** ရဲ့ဖွဲ့စည်းပုံဖြစ်ပါတယ်။

id	item_id	category_id	amount
1	101	1	50
2	101	1	30

id	item_id	category_id	amount
3	102	1	20
4	101	2	40
5	103	1	10
6	102	1	60
7	101	1	25
8	104	2	90
9	103	2	15

ကျွန်တော်တို့ရေးထားတဲ့ **Eloquent query** ကို **RAW SQL query** နဲ့ရေးမယ်ဆိုရင် ဒီဖြစ်ပါမယ်။

```
SELECT
   MIN(id) AS id,
   item_id,
   category_id,
   SUM(amount) AS total_amount
FROM transactions
GROUP BY item_id, category_id
HAVING SUM(amount) > 0
ORDER BY item_id;
```

8.1 Grouping by item_id and category_id

နမူနာ table ကို item_id နဲ့ category_id ကို group လုပ်လိုက်မယ်ဆိုရင် အောက်ပါအတိုင်း **group** တွေရလာပါမယ်။

Group 1: item_id = 101, category_id = 1

- Transactions: (id 1, 2, 7)
- Total amount: 50 + 30 + 25 = 105
- min(id): 1 (1 က group ထဲမှာ အသေးဆုံး id နံပါတ်မို့လို့ပါ)

Group 2: item_id = 101, category_id = 2

- Transactions: (id 4)
- Total amount: 40
- min(id): 4 (ဒီ group ထဲမှာ တစ်ကြောင်းတည်းရှိတာမို့ သူ့ min(id) ကလည်း 4 ပါပဲ)

Group 3: item_id = 102, category_id = 1

- Transactions: (id 3, 6)
- Total amount: 20 + 60 = 80
- min(id): 3 (group ထဲမှာ အနည်းဆုံး id နံပါတ်က 3 မို့လို့ပါ)

Group 4: item_id = 103, category_id = 1

- Transactions: (id 5)
- Total amount: 10
- min(id): 5 (id နံပါတ် 5 ရှိတဲ့ transaction တစ်ကြောင်းတည်းရှိပါတယ်)

Group 5: item_id = 103, category_id = 2

- Transactions: (id 9)
- Total amount: 15

- min(id): 9 (id နံပါတ် 9 ရှိတဲ့ transaction တစ်ကြောင်းတည်းရှိပါတယ်)

Group 6: item_id = 104, category_id = 2

- Transactions: (id 8)
- Total amount: 90
- min(id): 8 (id နံပါတ် 8 ရှိတဲ့ transaction တစ်ကြောင်းတည်းရှိပါတယ်)

query လုပ်ပြီးတဲ့အခါမှာတော့ အောက်ပါ table အတိုင်းရရှိပါမယ်။

id	item_id	category_id	total_amount
1	101	1	105
4	101	2	40
3	102	1	80
5	103	1	10
9	103	2	15
8	104	2	90

id column က group လုပ်တဲ့အထဲမှာ ဘယ်တော့မှထည့်လို့မရပါဘူး။ တန်ဖိုးတူတာတွေကိုပဲ group လုပ်လို့ရတာမို့လို့ unique primary key တွေဖြစ်တဲ့ id နဲ့ group လုပ်တာဟာ group မလုပ်ထားတဲ့ table အတိုင်းပဲပြန်ရမှာပါ။ amount column ကို SUM(amount) နဲ့ aggregate လုပ်မယ်၊ ကျန်တဲ့ column တွေကိုတော့ group() နဲ့ aggregate လုပ်မယ်၊ အဲဒီမှာကျန်ခဲ့တဲ့id column က group ထဲက ကြိုက်တဲ့ နံပါတ်တစ်ခုကို ကိုယ်စားပြုရလို့ရပါတယ်။ ဒီတော့ group ထဲက အနည်းဆုံး id နံပါတ်ကိုပဲ group row ရဲ့id အဖြစ်ယူသုံးလိုက်တဲ့သဘောပါ။ ဒီနေရာမှာ min(id) ရေးရေး၊ max(id) ရေးရေး သဘောတရားချင်းအတူတူပါပဲ။

id column ကို query ထဲလုံးဝမထည့်ဘဲ ကျန်တာတွေပဲ query လုပ်လို့မရဘူးလားဆိုရင် ရပါတယ်။ read only ပဲဖော်ပြမယ့် table တွေအတွက်ဆိုရင်တော့ပြဿနာမရှိပါဘူး။ အခု Batch table မှာတော့ table row တစ်ခုချင်းစီကိုနှိပ်လိုက်ရင် modal dialog တစ်ခုပွင့်လာပြီး database transaction တွေ ဆက်လုပ်စရာရှိတဲ့အတွက် unique ဖြစ်တဲ့ primary key column တစ်ခုပါမှဖြစ်မယ်လို့မှတ်ထားပေးပါ။

8.2 Table Components

MyInventory မှာလောလောဆယ် ဘာမှမမြင်ရသေးပါဘူး။ public function table() column([]) ထဲကို TransactionResource.php ထဲက item name နဲ့ category name column တွေကူးထည့်ပေးပါ။ total_amount အတွက် column တစ်ခုရေးပေးပါ။

```
public function table(Table $table): Table
{
    return $table
        ->query(self::tableQuery())
        ->columns([
            TextColumn::make('item.name')
                ->searchable()
                ->sortable(),
            TextColumn::make('category.name')
                ->searchable()
                ->sortable(),
            TextColumn::make('total_amount')
                ->numeric(thousandsSeparator: ',')
                ->alignEnd()
                ->badge()
                ->sortable(),
```

```
        ]);
    }
```

numeric(thousandsSeparator: ',') က ပစ္စည်းအရေအတွက်တန်ဖိုးတွေကို ကိန်းဂဏန်းအနေနဲ့ပြပေးပြီး ထောင်ဂဏန်းမှာ **comma** ဖြတ်ပေးပါတယ်။

alignEnd - ကိန်းဂဏန်းမို့လို့ **align right** လုပ်ပါမယ်

badge() - ဒါကတော့ **badge style** သတ်မှတ်ပေးလိုက်တာပါ

use Filament\Tables\Columns\TextColumn; ဆိုတဲ့ **use statement** ပါမှ **TextColumn class** ကိုခေါ်သုံးလို့ရပါတယ်။ ဒါကြောင့် **TextColumn** လို့ရိုက်ထည့်ပြီး **Right click > import class** လုပ်လိုက်ရင် **use statement** ထည့်ပေးသွားမှာပါ။ နောက်ပိုင်းမှာ **import class** လုပ်ဖို့ကိုအမြဲတမ်းထည့်ရေးတော့မှာမဟုတ်တဲ့အတွက် **class** အသစ်တွေထည့်သုံးတိုင်းမှာ **import class** လုပ်ပေးဖို့သတိပြုရပါမယ်။

TextColumn::make() က **table query field** ကို **text column** အဖြစ်ဖန်တီးပါတယ်။**searchable()** က ဒီ **column** ကို **search** လုပ်လို့ရအောင် လုပ်ပေးပါတယ်။ ဆိုလိုတာက **table** ရဲ့ **search box** မှာ ဒီ **field** ကို ရှာလို့ရပါမယ်။ **sortable() - column header** ကိုနှိပ်ပြီး **ascending/descending sort** စီလို့ရအောင် လုပ်ပေးပါတယ်။

8.3 Item Detail Page

My Inventory စာမျက်နှာမှာ ပစ္စည်းတစ်ခုချင်းစီရဲ့ စုစုပေါင်း **data** ကို **table** နဲ့ ပြပေးပါတယ်။ **table row** ကိုနှိပ်လိုက်ရင် သက်ဆိုင်ရာဆေးပစ္စည်းရဲ့ အသေးစိတ်စာမျက်နှာကိုဆက်သွားပြီး **batch** အလိုက် စုစုပေါင်းအရေအတွက်၊ အသေးစိတ်အဝင်/အထွက်မှတ်တမ်း စသည်ဖြင့်ကြည့်လို့ရအောင် **Item Detail** စာမျက်နှာတစ်ခုဖန်တီးပါမယ်။

```
php artisan make:filament-page ItemDetail
```

ဘယ် **Resource** မှာလုပ်မှာလဲလို့မေးလာရင် ဘာမှမထည့်ဘဲ **enter** နှိပ်ပါ။ **browser** မှာကြည့်လိုက်ရင် **Item Detail** ဆိုတဲ့ **menu item** ပေါ်လာပါလိမ့်မယ်။ ဒီစာမျက်နှာဟာ **menu** ကနေတိုက်ရိုက်သွားရတာမဟုတ်ဘဲ **My Inventory** စာမျက်နှာရဲ့ **table row** ကို **click** နှိပ်တဲ့အခါမှ သွားစေချင်တာမို့လို့ **navigation menu** မှာ ဖျောက်ပေးဖို့လိုပါတယ်။ **ItemDetail.php** ကိုဖွင့်ပါ။ **$shouldRegisterNavigation boolean** တန်ဖိုးကို **false** ပေးပါ။ **Page class** ရဲ့ **default variable** ဖြစ်တဲ့ **$shouldRegisterNavigation** ကို **false** ပေးပြီး **override** လုပ်လိုက်တာဖြစ်ပါတယ်။

```
protected static bool $shouldRegisterNavigation = false;
```

browser မှာ ကြည့်ရင် **Item Detail menu item** မရှိတော့ပါဘူး။ ပြီးရင်အဲဒီစာမျက်နှာကိုခေါ်သွားပေးမယ့် **table action** တစ်ခုပြုလုပ်ပါမယ်။

table actions တွေဟာ **row** တစ်ခုချင်းစီအတွက် **action** ဖြစ်ပါတယ်။ သက်ဆိုင်ရာ **row** ကို **CRUD operation** လုပ်ချင်တဲ့အခါ အသုံးပြုပါတယ်။ အသုံးများတဲ့ **table action** တွေက **ViewAction, EditAction** နဲ့ **DeleteAction** ပါ။ တခြား **action** တွေလည်းရေးယူလို့ရပါတယ်။

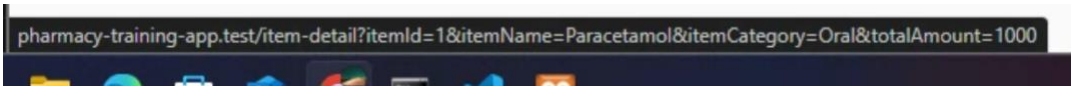
default ပါပြီးသား **ViewAction** ကိုမသုံးဘဲ **View** လုပ်တဲ့ **action** တစ်ခု **customize** ရေးကြည့်ပါမယ်။ **Action::make('view')** က **View** နာမည်နဲ့ **action** ခလုတ်တစ်ခုလုပ်ပေးပါတယ်။ **url() method** က နှိပ်လိုက်ရင်ဘယ်လိပ်စာကိုသွားချင်လဲ သတ်မှတ်ပေးပါတယ်။ **url() method** ထဲမှာ လက်ရှိ **table row** ရဲ့ **data** ကိုခေါ်သုံးဖို့ **function($record){}** နဲ့ **$record parameter** ကိုလက်ခံပေးပါတယ်။ အဲဒီကနေမှ အသစ်ဖန်တီးထားတဲ့ **ItemDetail class** ကို **getUrl() method** နဲ့ **url** ထုတ်ယူလိုက်ပါတယ်။ ဒီ **action** ခလုတ်ကိုနှိပ်လိုက်ရင် **Item Detail** စာမျက်နှာကိုသွားမယ်လို့ဆိုလိုပါတယ်။ ဒါပေမယ့်ဒီတိုင်းမသွားပါဘူး။ လိုအပ်တဲ့ **parameter** တွေပါသယ်ဆောင်ပေးရပါမယ်။

getUrl() method ထဲမှာ [] ထည့်ပြီး သယ်ယူချင်တဲ့ **parameter** တွေကို **key => value assignment** နဲ့ထည့်ပေးလို့ရပါတယ်။

```
    public function table(Table $table): Table
    {
        return $table
            ->query(self::tableQuery())
            ->columns([
```

```
                // ...
        ])
        ->actions([
            Action::make('view')
                ->url(function ($record) {
                    return ItemDetail::getUrl([
                        'itemId' => $record->item_id, // Transaction query မှ item_id

                        'itemName' => $record->item->name, // Transaction model ရဲ့ item relationship ကနေ item name ကိုလှမ်းယူ

                        'itemCategory' => $record->category->name, // Transaction model ရဲ့ category relationship ကနေ categor name လှမ်းယူ

                        'itemTotalAmount' => $record->total_amount, // Transaction query မှ total_amount

                    ]);
                }),
        ]);
    }
```

Save လုပ်ပြီး **browser** မှာ ကြည့်ပါ။ **View** ခလုတ်ကိုမနှိပ်သေးဘဲ **mouse pointer** တင်ကြည့်ပါ။ **browser** ရဲ့ဘယ်ဘက်အောက်ထောင့်မှာ **url preview** မြင်ရမှာဖြစ်ပါတယ်။



နောက်စာမျက်နှာကို **url** ကတဆင့် **parameter** တွေလှမ်းပို့ပေးပါတယ်။ **Item Detail** စာမျက်နှာမှာ အဲဒီ **parameter** တွေကိုလက်ခံပေးရပါမယ်။

ItemDetail.php ကိုဖွင့်ပြီး လက်ခံမယ့် **property variable** တွေထည့်ပေးပါ။

```
class ItemDetail extends Page
{
    public int $itemId;
    public string $itemName;
    public string $itemCategory;
    public int $itemTotalAmount;

    // ...
}
```

ItemDetail class မှာ ကြေညာထားတဲ့ **property** တွေကို **private** အဖြစ် သတ်မှတ်လို့ မရပါဘူး။ **Filament** က ဒီ **property** တွေကို အသုံးပြုပြီး **page** ကို **render** လုပ်တဲ့အခါ လိုအပ်တဲ့ **data** တွေကို **pass** လုပ်ပါတယ်။**public property** တွေက **view** ထဲမှာ တိုက်ရိုက် **access** လုပ်လို့ရအောင် ခွင့်ပြုပေးပါတယ်။ ဥပမာ - **Blade template** ထဲမှာ {{ **$itemName** }} လို့ သုံးနိုင်ပါတယ်

public function mount() ကနေ **page** ရဲ့**property** တွေထဲကို **request parameter** တွေ လာထည့်ပေးပါ။

public function mount()method ဟာ **page** တစ်ခု **load** လုပ်တဲ့အခါ အရင်ဆုံး **run** တဲ့ **method** ဖြစ်ပါတယ်။ **constructor** လို သဘောမျိုးနဲ့ အသုံးပြုကြပါတယ်။ **mount() method** ထဲမှာ **page** နဲ့ ပတ်သက်တဲ့ **initial setup** တွေ၊ **data** တွေ **load** လုပ်တာတွေ၊ **permission check** လုပ်တာတွေ စတဲ့ လုပ်ငန်းတွေကို ထည့်ရေးလေ့ ရှိပါတယ်။ ဥပမာအားဖြင့် **URL** ကနေ လက်ခံရရှိတဲ့ **parameter** တွေကို **property** တွေထဲ **assign** လုပ်တာမျိုး၊ **database** ကနေ လိုအပ်တဲ့ **data** တွေကို **query** လုပ်ပြီး **load** လုပ်တာမျိုးတွေကို ဒီ **method** ထဲမှာ ရေးလေ့ရှိပါတယ်။ **user** ရဲ့ **permission** ကို စစ်ဆေးပြီး လုပ်ပိုင်ခွင့်မရှိရင် **redirect** လုပ်တာမျိုးတွေကိုလည်း ဒီနေရာမှာ လုပ်ဆောင်နိုင်ပါတယ်။ **mount() method** က **page** ရဲ့ **lifecycle** မှာ အစောဆုံး **run** တဲ့ **method** ဖြစ်တဲ့အတွက် **page render** မလုပ်ခင် လုပ်ဆောင်ရမယ့် အရေးကြီးတဲ့ **setup** တွေအားလုံးကို ဒီ **method** ထဲမှာ ရေးသားလေ့ရှိပါတယ်။

```
class ItemDetail extends Page
{
   public int $itemId;
   public string $itemName;
   public string $itemCategory;
   public int $itemTotalAmount;

   public function mount()
   {
      $this->itemId = request('itemId') ?? 0;
      $this->itemName = request('itemName') ?? '';
      $this->itemCategory = request('itemCategory') ?? '';
      $this->itemTotalAmount = request('itemTotalAmount') ?? 0;
   }

   // ...
}
```

itemId, itemName, itemCategory, နဲ့ **itemTotalAmount** ဆိုတဲ့ **parameter** တွေကို **request** ကနေ ဆွဲယူပြီး **page property** တွေထဲကိုထည့်ပေးပါတယ်။ **Null coalescing operator ??** ကို သုံးထားတာကတော့ **request** မှာ သက်ဆိုင်ရာ **parameter** မပါလာခဲ့ရင် **default value** တစ်ခု သတ်မှတ်ပေးဖို့ ဖြစ်ပါတယ်။ ဥပမာ - **itemId** မပါလာရင် 0 ကို **assign** လုပ်မယ်၊ **itemName** နဲ့ **itemCategory** မပါရင် **empty string** ကို **assign** လုပ်မယ်၊ **itemTotalAmount** မပါရင်လည်း 0 ကို **assign** လုပ်မယ် ဆိုတဲ့ သဘောပါ။

infolist method ထည့်ပြီး **property** တွေကိုပြပေးပါမယ်။ **infolist() method** မှာ **state()** ထဲမှာ **itemId, itemName, itemCategory,** နဲ့ **itemTotalAmount** တို့ကို ထည့်သွင်းထားပါတယ်။ **schema() method** က **Infolist** ရဲ့ **layout** ကို ကြေငြာပေးတာပါ။ **Layout** မှာ **Split component** ကို သုံးထားပြီး၊ **Section** တစ်ခုထည့်ပါမယ်။ **Section** ထဲမှာတော့ **TextEntry** သုံးခု ပါဝင်ပြီး **itemName, itemCategory,** နဲ့ **itemTotalAmount** တို့ကို ပြပေးပါမယ်။ **itemTotalAmount** အတွက် **badge()** နဲ့ **numeric() method** တွေကို သုံးထားတာက **badge** ပုံစံနဲ့ ပြသပြီး **number** အနေနဲ့ ထောင်ဂဏန်းမှာ , ထည့်မယ်လို့ဆိုလိုပါတယ်။ နောက်ဆုံးမှာ **grow(false)** ဆိုတာက **Section** ရဲ့ အရွယ်အစားကို ပုံသေထားမယ်လို့ ဆိုလိုပါတယ်။

```
class ItemDetail extends Page
{
   public int $itemId;
   public string $itemName;
   public string $itemCategory;
   public int $itemTotalAmount;
```

```
    public function mount()
    {
        $this->itemId = request('itemId') ?? 0;
        $this->itemName = request('itemName') ?? '';
        $this->itemCategory = request('itemCategory') ?? '';
        $this->itemTotalAmount = request('itemTotalAmount') ?? 0;
    }
    protected static ?string $navigationIcon = 'heroicon-o-document-text';

    protected static string $view = 'filament.pages.item-detail';

    protected static bool $shouldRegisterNavigation = false;

    public function infolist(Infolist $infolist): Infolist
    {
        return $infolist
            ->state([
                'itemId' => $this->itemId,
                'itemName' => $this->itemName,
                'itemCategory' => $this->itemCategory,
                'itemTotalAmount' => $this->itemTotalAmount,
            ])
            ->schema([
                Split::make([
                    Section::make([
                        TextEntry::make('itemName')
                            ->label('Item Name'),
                        TextEntry::make('itemCategory')
                            ->label('Category'),
                        TextEntry::make('itemTotalAmount')
                            ->label('Total Amount')
                            ->badge()
                            ->numeric(),
                    ])->grow(false),
                ]),
            ]);
    }
}
```

save လုပ်ပြီး **item-detail.blade.php** ဖိုင်ကိုဖွင့်ပါ။ **component** အဖွင့်နဲ့ အပိတ် **tag** ကြားထဲမှာ {{ **$this->infolist** }} ထည့်ပေးပါ။ **Filament page layout** ထဲမှာ **infolist** ကို **render** လုပ်ပြီး ပြသပေးမှာဖြစ်ပါတယ်။ **save** ပြီး **browser** မှာကြည့်ပါ။

```
<x-filament-panels::page>
{{ $this->infolist }}
</x-filament-panels::page>
```

9 Creating a Livewire component

Split component ထဲမှာ **table** ထည့်ပါမယ်။ **table** ကို **infolist** ထဲမှာ တိုက်ရိုက်ထည့်လို့မရပါဘူး။ **Livewire component** အရင်ဆောက်ပြီး **table render** လုပ်၊ ပြီးမှ **Livewire component** ကို **infolist** ထဲလာထည့်လို့ရပါတယ်။ **Livewire component** ဖန်တီးတဲ့ **command** က -

```
php artisan make:livewire ItemBatchList
```

app/Livewire/ItemBatchList.php ဖိုင်နဲ့ **resources/views/livewire/item-batch-list.blade.php** ဖိုင်တွေရလာပါမယ်။ သူတို့ကိုမဖွင့်သေးဘဲ **ItemDetail.php** ဖိုင်ကိုပဲဆက်ပြင်ပါမယ်။ **Split::make([])** အလုပ်လုပ်တဲ့ပုံစံက သူ့ထဲမှာရှိတဲ့ **component** တွေကို **column layout** တစ်ခုစီခွဲပြပေးတာပါ။ **Split** ရဲ့ **first child** က **Section** တစ်ခုပဲရှိပါသေးတယ်။ **Section** အောက်ထဲက **child** တွေကို **Split** က သက်ရောက်မှုမရှိပါဘူး။ **first child** ရဲ့နောက်ဆုံးမှာ **comma** ခံပြီး နောက် **component** တစ်ခုဆက်ရေးလိုက်တာနဲ့ **layout** မှာ **split** လုပ်ပေးမှာဖြစ်ပါတယ်။ **first child** ဖြစ်တဲ့ **Section** ဟာ **->grow(false)**, မှာဆုံးပါတယ်။ ဒါကြောင့် သူ့နောက်မှာအသစ်ဖန်တီးလိုက်တဲ့ **Livewire component** ကို ထည့်ပေးလိုက်ပါမယ်။

```
    ->schema([
      Split::make([
        Section::make([
          TextEntry::make('itemName')
            ->label('Item Name'),
          TextEntry::make('itemCategory')
            ->label('Category'),
          TextEntry::make('itemTotalAmount')
            ->label('Total Amount')
            ->badge()
            ->numeric(),
        ])->grow(false),

        Livewire::make(ItemBatchList::class, ['itemId' => $this->itemId]) // use Filament\Infolists\Components\Livewire;
      ]),
    ]);
```

ItemBatchList class ရဲ့နောက်ကနေ **['itemId' => $this->itemId] parameter** ကိုပါ **pass** လုပ်ပေးထားပါတယ်။ **arrow operator =>** ရှေ့မှာ လက်ခံမယ့် **page** က **property** ကိုရေးရပြီး နောက်မှာပေးပို့မယ့်စာမျက်နှာက **property** ကိုရေးရပါတယ်။

ItemBatchList.php ကိုဖွင့်ပါ။ ဒီထဲမှာ **table** ထည့်မှာမို့လို့ **HasTable** နဲ့ **HasForms trait** တွေကို **inherit** လုပ်ရပါမယ်။ **InteractsWithTable** နဲ့ **InteractsWithForms** တို့ကိုလည်းသုံးပေးရပါမယ်။

```
<?php

namespace App\Livewire;
```

```
use Filament\Forms\Concerns\InteractsWithForms;
use Filament\Forms\Contracts\HasForms;
use Filament\Tables\Concerns\InteractsWithTable;
use Filament\Tables\Contracts\HasTable;
use Livewire\Component;

class ItemBatchList extends Component implements HasTable, HasForms
{
  use InteractsWithTable, InteractsWithForms;

  public function render()
  {
    return view('livewire.item-batch-list');
  }
}
```

table စတင်တည်ဆောက်ပါတော့မယ်။ **public function table()** ကို အောက်ပါအတိုင်းထည့်ပေးပါ။ **table** ရှိရင် **query** ရှိရမှာမို့လို့ **table query method** လည်းဆက်ရေးပေးဖို့လိုပါတယ်။

```
class ItemBatchList extends Component implements HasTable, HasForms
{
  use InteractsWithTable, InteractsWithForms;

  public function render()
  {
    return view('livewire.item-batch-list');
  }

  public function table(Table $table): Table
  {
    return $table
      ->query(self::tableQuery())
      ->columns([]);
  }

  private function tableQuery(): Builder
  {
    return Transaction::query();
  }
}
```

Livewire class မှာရေးထားတဲ့ **component** တွေကို **blade** မှာပါရေးပေးမှ **render** လုပ်ပေးမှာဖြစ်ပါတယ်။

item-batch-list.blade.php

```
<div>
  {{ $this->table }}
</div>
```

Browser မှာ refresh ကြည့်ရင် table အလွတ်တစ်ခုတွေ့ရပါမယ်။ ဒီ table ထဲမှာ column တွေအနေနဲ့ Item, Category, Exp date, package form, batch number, donor, total amount တွေထည့်ပေးပါမယ်။ အစောပိုင်းတုန်းက လုပ်ခဲ့တဲ့ TransactionResource.php ဖိုင်ကိုပြန်ဖွင့်ပါ။ လိုအပ်တဲ့ TextColumn တွေကို ကော်ပီကူးယူပြီး table အသစ်ရဲ့ column array ထဲမှာ paste လုပ်ပါမယ်။

```
  public function table(Table $table): Table
  {
    return $table
      ->query(self::tableQuery())
      ->columns([
        TextColumn::make('item.name')
          ->searchable(),
        TextColumn::make('category.name')
          ->searchable(),
        TextColumn::make('packageForm.name')
          ->searchable(),
        TextColumn::make('exp_date')
          ->date()
          ->sortable(),
        TextColumn::make('batch')
          ->searchable(),
        TextColumn::make('amount')
          ->numeric()
          ->sortable(),
        TextColumn::make('donor.name')
          ->searchable(),
        TextColumn::make('sourceWarehouse.name')
          ->searchable(),
      ]);
  }
```

Browser မှာ refresh ကြည့်ရင် table ထဲက row တွေက item ပေါင်းစုံရဲ့ transaction ကိုပြနေမှာပါ။ ကျွန်တော်တို့အခုကြည့်နေတဲ့ item နဲ့ဆိုင်တဲ့ transaction record ကိုပဲ filter လုပ်ချင်ပါတယ်။ လက်ရှိစာမျက်နှာပိုင်ရင် item ရဲ့ id တန်ဖိုးကိုအသုံးချပြီး transaction table ထဲက အဲဒီ item_id ရှိတဲ့ record တွေပဲရွေးကြည့်ချင်တာပါ။ ဒါကြောင့် Livewire class ကို import လုပ်တဲ့အခါ page property ဖြစ်တဲ့ $this→itemId ကို pass လုပ်ပေးရတာဖြစ်ပါတယ်။ pass လုပ်ပေးထားတဲ့ $itemId ကို Livewire class ထဲမှာ လက်ခံပေးဖို့လိုပါတယ်။ public variable ကြေငြာပြီး mount method နဲ့ parameter ကို assign လုပ်ပေးပါ။

```
class ItemBatchList extends Component implements HasTable, HasForms
{
  use InteractsWithTable, InteractsWithForms;
```

```
    public int $itemId;

    public function mount($itemId)
    {
        $this->itemId = $itemId;
    }
    public function render()
    {
        return view('livewire.item-transaction-list');
    }

    //...
}
```

page property အနေနဲ့ **itemId** တန်ဖိုးရှိသွားပြီဖြစ်တဲ့အတွက် ဒီတန်ဖိုးကိုသုံးပြီး **database query** လုပ်လို့ရပါပြီ။ **tableQuery() method** ထဲက **Transaction::query()** မှာ **filter** လုပ်ပေးပါ။

```
    private function tableQuery(): Builder
    {
        $itemId = $this->itemId;
        return Transaction::query()
            ->where('item_id', '=', $itemId);
    }
```

Transaction query ကို ထပ်ပြီးအသေးစိတ်ရေးကြည့်ပါမယ်။ ဒီ **table** မှာ အဝင်အထွက်မှတ်တမ်းအားလုံးကို မပြချင်ပါဘူး။ **Item, Category, Exp date, Batch number, Donor, Source** ဆိုတဲ့ **column** ခြောက်ခုတူရင်တူသလောက် **group** လုပ်ပြီးပြချင်ပါတယ်။ တစ်ခုခုမတူတာနဲ့ အုပ်စုမတူတဲ့ **batch** တစ်ခုအဖြစ်ခွဲပြချင်ပါတယ်။ ဒီလိုလေးဆက်ရေးကြည့်ပါ။

```
    private function tableQuery(): Builder
    {
        $itemId = $this->itemId;
        return Transaction::query()
            ->selectRaw('
            min(id) as id,
            item_id,
            category_id,
            package_form_id,
            exp_date,
            batch,
            SUM(amount) AS total_amount,
            donor_id,
            source')
            ->where('item_id', '=', $itemId)
            ->groupBy('item_id', 'category_id', 'package_form_id', 'exp_date', 'batch', 'donor_id', 'source')
            ->havingRaw('SUM(amount) > 0')
```

```
        ->orderBy('exp_date', 'ASC');
    }
```

selectRaw() method က **transactions table** ထဲက **record** တွေကို **raw SQL query** သုံးပြီးဆွဲထုတ်ပေးပါတယ်။ **Laravel** ရဲ့ **eloquent ORM** ရဲ့ကောင်းတဲ့အချက်တစ်ခုက လိုအပ်ရင် **raw sql query** တွေလည်းထည့်ရေးလို့ရပါတယ်။ ဘာလို့ **raw query** သုံးရလဲဆိုတော့ **SUM(amount)** သုံးချင်လို့ပါ။ ပြီးမှ ရလာတဲ့ **column** တွေကို **groupBy() method** နဲ့တူရင်တူသလောက် ပေါင်းလိုက်ပါတယ်။ **groupBy()** ထဲမှာ **SUM(amount) column** ထည့်ဖို့မလိုပါဘူး။ သူကနဂိုကတည်းက **aggregated column** ဖြစ်နေပြီမို့လို့ **group** လုပ်လို့မရပါဘူး။ **havingRaw() method** ကတော့ ပုံမှန် **having()** ကိုမှ **raw query** သုံးချင်လို့ပါ။ **SUM(amount) 0** ထက်ကျော်တဲ့ **item** တွေပဲပြပါမယ်။ ဟိုးရှေးရှေးကတည်းက သုံးလို့ကုန်ပြီး **balance 0** ဖြစ်နေတဲ့ဟာမှန်သမျှပေါ်နေတာမျိုး မလိုချင်လို့ပါ။ **orderBy()** မှာတော့ **exp_date** အနီးဆုံးပစ္စည်းတွေကို အပေါ်ဆုံးတင်လိုက်ပါတယ်။

date တွေကို **ascending** စီရင် အစောဆုံးကနေ နောက်အကျဆုံးကိုစီပါတယ်။ ဥပမာ -

Date Ascending

2024-01-15
2024-02-10
2024-03-05
2024-04-20
2024-05-30

Descending စီရင်တော့ နောက်အကျဆုံးကနေ အစောဆုံးကိုစီပါတယ်။ အနီးဆုံး **expiry date** ကနေစီချင်ရင် **ascending** ကိုသုံးရပြီး နောက်ဆုံးရသတင်းများစာမျက်နှာမှာလို့ သုံးဖို့ဆိုရင်တော့ **descending** စီရမှာဖြစ်ပါတယ်။

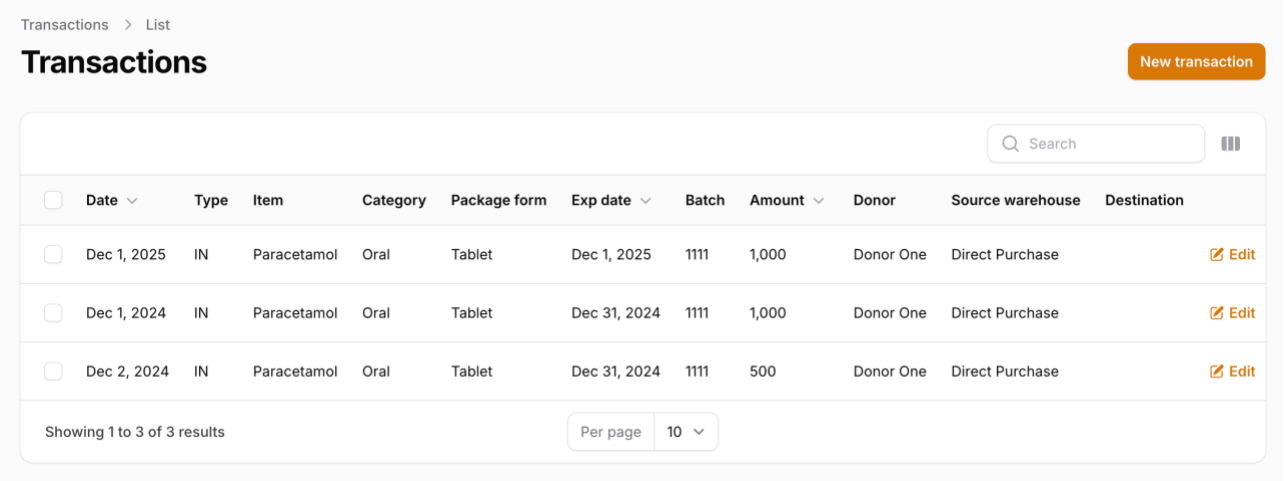
RAW SQL query တစ်ခုရဲ့ဖွဲ့စည်းပုံဟာ အောက်ပါအစီအစဉ်အတိုင်းလိုက်နာရပါတယ်။ **Laravel** ရဲ့ **eloquent ORM** နဲ့ရေးရင်လည်း အဲဒီအစီအစဉ်ကနေ သွေဖည်လို့မရပါဘူး။

1. **SELECT**
2. **FROM**
3. **WHERE**
4. **GROUP BY**
5. **HAVING**
6. **ORDER BY**

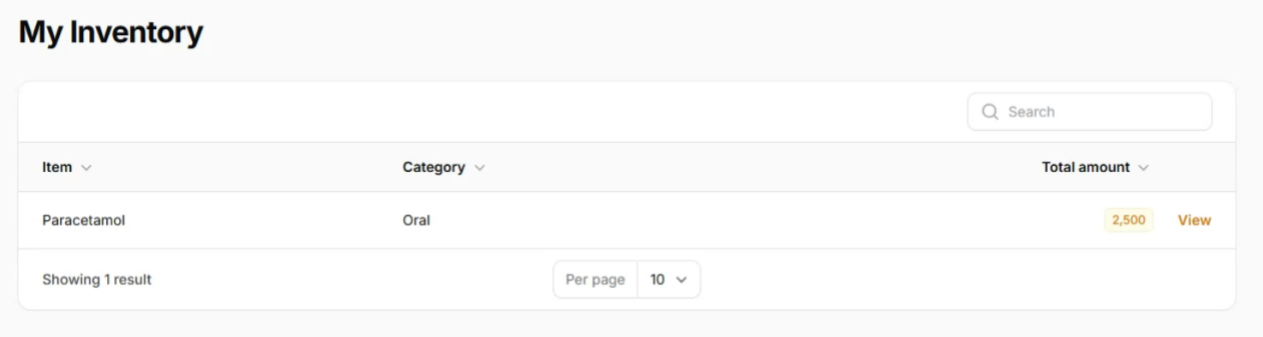
min(id) as id အကြောင်းကိုတော့ **Item Inventory** စာမျက်နှာတုန်းက ရှင်းပြခဲ့ပြီးဖြစ်ပါတယ်။

raw query မှာ **total_amount** ဆိုတဲ့ **alias** နဲ့ **column** ထုတ်ထားတာမို့လို့ **TextColumn::make('amount')** ကို **total_amount** လို့ပြောင်းပေးပါ။ **table** ကောင်းကောင်းအလုပ်လုပ်လားစမ်းကြည့်ဖို့အတွက် ရှိပြီးသား **item** နဲ့ **exp_date, package_form, batch, donor, source** အားလုံးတူတဲ့ **record** တစ်ခုနဲ့ ဒီ **item** ကိုပဲ **expiry date** ပဲမတူတဲ့ နောက်ထပ် **record** တစ်ခုထပ်ထည့်ကြည့်ပါ။

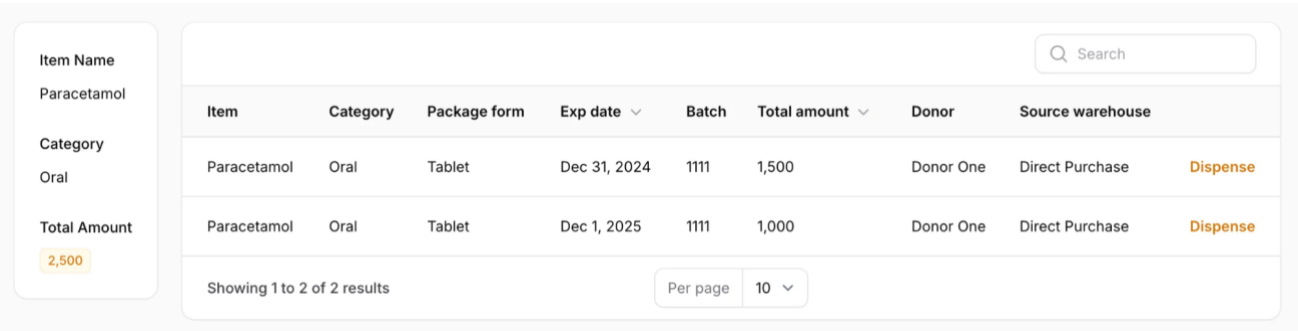
ဒီ **example** မှာဆိုရင် **Transaction Resource table** ရဲ့တစ်ကြောင်းချင်းကြည့်လို့ရတဲ့ စာမျက်နှာမှာ ဆေးပစ္စည်းတစ်မျိုးတည်းရဲ့ **record** သုံးခုရှိပါတယ်။ အားလုံးတူတဲ့ **record** က နှစ်ကြောင်းနဲ့ အားလုံးတူပြီး **expiry date** လေးပဲကွဲသွားတဲ့ **record** ကတစ်ကြောင်းပါ။



ဒါကိုပဲ **My Inventory table** မှာသွားကြည့်ရင်ပစ္စည်းတစ်မျိုးတည်းမို့လို့ တစ်ကြောင်းတည်းပေါ်ပါမယ်။ ပစ္စည်းတူတာနဲ့ ကျန်တာတူတူ/မတူတူ အကုန်လုံးပေါင်းထားတဲ့ **balance** ကိုပြပေးပါတယ်။



ဆေးပစ္စည်းရဲ့ညာဘက်အစွန်က **View** ကိုနှိပ်ရင် သူ့ကိုယ်ပိုင်စာမျက်နှာ **Item Detail page** ကိုရောက်ပါမယ်။ **Item batch list table** မှာတော့ **expiry date** တူညီတဲ့ **record** နှစ်ခုကိုပေါင်းပြပြီး တစ်ခုခုမတူတာနဲ့ နောက်ထပ် **record** တစ်ခုအဖြစ်ခွဲပြပေးမှာပါ။



ဒီလောက်ဆိုရင် **ItemBatchList table** ကလုံလောက်ပါပြီ။ ဒီ **table** ရဲ့အောက်မှာ နောက်ထပ်စာမျက်နှာအပြည့် **table** တစ်ခုထပ်ထည့်ပါမယ်။ **Transaction Resource** မှာလိုပဲ အဝင်အထွက်မှတ်တမ်းအားလုံးကို တစ်ကြောင်းချင်းပြချင်တာပါ။ တစ်ခုပဲကွာသွားမှာက လက်ရှိရောက်နေတဲ့စာမျက်နှာပိုင်ရှင် ဆေးပစ္စည်းရဲ့မှတ်တမ်းတွေချည်းပဲပါချင်ပါတယ်။ **Livewire component** ဖန်တီးတဲ့လုပ်ငန်းစဉ်ကို အစကနေနောက်တစ်ခေါက်ထပ်လုပ်ကြည့်ပါမယ်။

```
php artisan make:livewire ItemTransactionList
```

ItemTransactionList.php နဲ့ **item-transaction-list.blade.php** နှစ်ဖိုင်ရပါမယ်။ **ItemTransactionList.php** ကိုဖွင့်ပါ။ **$itemId property** ရေးပေးပါ။ **mount() method** နဲ့ သူ့ဆီပို့ပေးတဲ့ **parameter** ကိုလက်ခံပြီး **itemId property** ထဲကို **assign** လုပ်ပေးပါမယ်။

table ဆောက်မှာဖြစ်တဲ့အတွက် HasTable နဲ့ HasForm trait တွေကို inherit လုပ်ပေးပါ။ InteractsWithTable နဲ့ InteractsWithForms class တွေကိုလည်း use ပေးပါ။

```php
<?php

namespace App\Livewire;

use Filament\Forms\Concerns\InteractsWithForms;
use Filament\Forms\Contracts\HasForms;
use Filament\Tables\Concerns\InteractsWithTable;
use Filament\Tables\Contracts\HasTable;
use Filament\Tables\Table;
use Livewire\Component;

class ItemTransactionList extends Component implements HasTable, HasForms
{
                    use InteractsWithTable, InteractsWithForms;
  public int $itemId;

  public function mount($itemId)
  {
    $this->itemId = $itemId;
  }

  public function render()
  {
    return view('livewire.item-transaction-list');
  }
}
```

ပြီးရင် table နဲ့ tableQuery method ရေးပေးပါ။ ဒီ table မှာက transaction record တွေကို aggregate မလုပ်ဘူးမို့လို့ row တိုင်းက unique transaction id နဲ့ရမှာပါ။ ဒါကြောင့် min(id) as idနဲ့ primary key column မလိုပါဘူး။

```php
class ItemTransactionList extends Component implements HasTable, HasForms
{
  // ...

  public function table(Table $table): Table
  {
    return $table
      ->query(self::tableQuery())
      ->columns([]);
  }
```

```
  private function tableQuery()
  {
    return Transaction::query()
      ->where('item_id', $this->itemId);
  }
}
```

item-transaction-list.blade.php ဖိုင်ကိုဖွင့်ပြီး **blade template** မှာ **table render** လုပ်ပေးရပါမယ်။

```
<div>
  {{ $this->table }}
</div>
```

blade template မှာ **render** လုပ်ပြီးပေမယ့် **browser** မှာသွားကြည့်ရင် မမြင်ရသေးပါဘူး။ **Item Detail page** ရဲ့ **infolist** ထဲမှာ **import** လုပ်ပေးရပါဦးမယ်။ **ItemDetail.php** ကိုဖွင့်ပါ။

```
      ->schema([
        Split::make([
          Section::make([
            TextEntry::make('itemName')
              ->label('Item Name'),
            TextEntry::make('itemCategory')
              ->label('Category'),
            TextEntry::make('itemTotalAmount')
              ->label('Total Amount')
              ->badge()
              ->numeric(),
          ])->grow(false),

          Livewire::make(ItemBatchList::class, ['itemId' => $this->itemId])
        ]),
        Livewire::make(ItemTransactionList::class, ['itemId' => $this->itemId])
      ]);
```

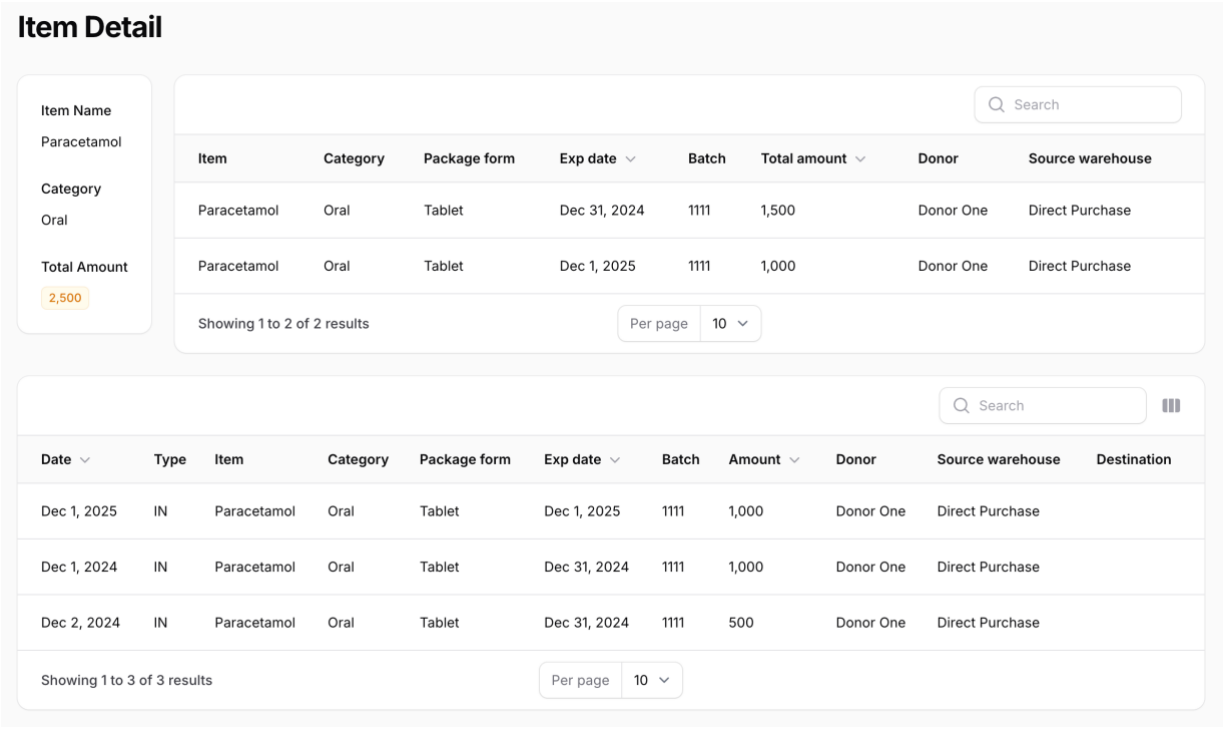
ItemTransactionList livewire component ကို **infolist schema** ထဲမှာ **Split** အပြင်ဘက်မှာ ထည့်ပေးလိုက်ပါမယ်။ လောလောဆယ် **Split** ထဲမှာ **component** နှစ်ခုရှိပါတယ်။ **Section** နဲ့ **Livewire ItemBatchList** ပါ။ အခုအသစ်လုပ်လိုက်တဲ့ **ItemTransactionList** ကို **Split** ရဲ့အပြင်ဘက်မှာထည့်ပေးလိုက်ပါမယ်။ ဒါမှ အောက်မှာသီးသန့်ပေါ်နေပါမှာ။ **Split** ထဲထည့်မိရင် ဘေးမှာ **column** တစ်ခုအနေနဲ့တိုးလာပြီး ကျဉ်းကျဉ်းကျပ်ကျပ်ဖြစ်နေပါလိမ့်မယ်။

ဒီလောက်ဆို **browser** မှာ **refresh** လုပ်ကြည့်ပါ။ **table** မြင်နေရပါမယ်။ **column** တွေဆက်ထည့်ပေးဖို့လိုပါတယ်။

ItemTransactionList table ဟာ **TransactionResource** ရဲ့ **table** နဲ့ **column** တွေလုံးဝအတူတူပဲမို့လို့ တိုက်ရိုက် **copy** ယူပြီး **paste** ချလိုက်လို့ရပါတယ်။

```
  public function table(Table $table): Table
  {
    return $table
```

```
        ->query(self::tableQuery())
        ->columns([
            TextColumn::make('date')
                ->date()
                ->sortable(),
            TextColumn::make('type')
                ->searchable(),
            TextColumn::make('item.name')
                ->searchable(),
            TextColumn::make('category.name')
                ->searchable(),
            TextColumn::make('packageForm.name')
                ->searchable(),
            TextColumn::make('exp_date')
                ->date()
                ->sortable(),
            TextColumn::make('batch')
                ->searchable(),
            TextColumn::make('amount')
                ->numeric()
                ->sortable(),
            TextColumn::make('donor.name')
                ->searchable(),
            TextColumn::make('sourceWarehouse.name')
                ->searchable(),
            TextColumn::make('destination')
                ->searchable(),
            TextColumn::make('created_at')
                ->dateTime()
                ->sortable()
                ->toggleable(isToggledHiddenByDefault: true),
            TextColumn::make('updated_at')
                ->dateTime()
                ->sortable()
                ->toggleable(isToggledHiddenByDefault: true),
        ]);
    }
```



ဒါဆိုရင် လက်ရှိကြည့်နေတဲ့ ဆေးပစ္စည်းရဲ့ **transaction** တစ်ကြောင်းစီကိုတွေ့ရမှာဖြစ်ပါတယ်။ **table query** ကတော့ **Transaction resource** ကပစ္စည်းအားလုံးကိုပြပေးနေတဲ့ **query** နဲ့တူတူပါပဲ။ **item_id** ကိုပဲ **page property** ဖြစ်တဲ့ **$itemId** နဲ့ **filter** လုပ်လိုက်တာပါ။

9.1.1 Add Back button

ဒါပြီးရင် **Item Detail** စာမျက်နှာကနေ **My Inventory** စာမျက်နှာကိုပြန်သွားတဲ့ **Back button** ထည့်ပါမယ်။ မထည့်ရင် **My Inventory** စာမျက်နှာကိုဘယ်လိုပြန်သွားရမှန်း **user** ကသိမှာမဟုတ်ပါဘူး။

ItemDetail.php မှာ **getHeaderActions method** ထည့်ပေးပါ။

```
protected function getHeaderActions(): array
{
    return [
        Action::make('back-action')
            ->label('Back to My Inventory')
            ->outlined()
            ->color('gray')
            ->url(MyInventory::getUrl())
    ];
}
```

outlined() ဆိုတာက **button** ကိုအပြင်မှာမျဉ်းတစ်လိုင်းအုပ်ထားတဲ့ပုံစံနဲ့ပြပေးနေမှာဖြစ်ပါတယ်။ **color('gray')** ကတော့ **default color** မသုံးဘဲအရောင်မထည့်ထားတဲ့ပုံစံပါ။ **url()** ထဲမှာပြန်သွားချင်တဲ့စာမျက်နှာရဲ့ **class** ကို **getUrl() method** နဲ့ထည့်ပေးလို့ရပါတယ်။

ဒီစာမျက်နှာမှာပဲ **batch** တစ်ခုချင်းစီကို နှိပ်ပြီးပစ္စည်းအထွက်မှတ်တမ်းတွေထည့်လို့ရအောင် ဆက်လုပ်ပါမယ်။

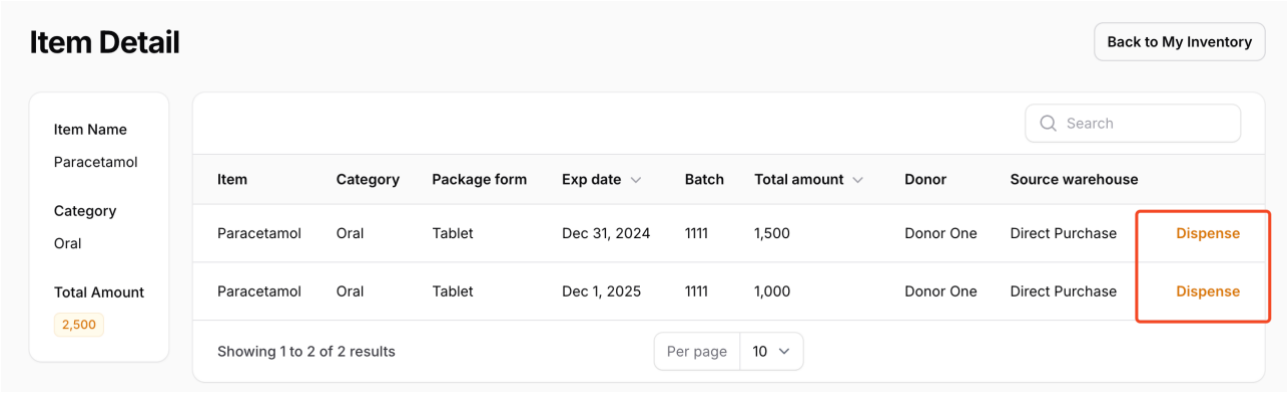
9.1.2 Add table action

ပစ္စည်းအထွက်မှတ်တမ်းတွေထည့်သွင်းတဲ့အခါ အဝင်တုန်းကလို ပစ္စည်းနာမည်၊ အမျိုးအစား၊ သက်တမ်းလွန်ရက်စွဲ စသည်ဖြင့် အစကနေပြန်မထည့်သင့်တော့ပါဘူး။ ရှိပြီးသား **batch** တစ်ခုချင်းစီကို ဒီထဲကဘယ်လောက်ထုတ်မယ်၊ ထုတ်မယ့်နေ့ကဘယ်နေ့လဲ၊

ဘယ်ကိုထုတ်ပေးမှာလဲ စသည်ဖြင့် **user** ဘက်ကနေထည့်ပေးဖို့ပဲလိုပါတော့တယ်။ ဒါကြောင့် ပစ္စည်းအထွက်မှတ်တမ်းအတွက် **form** တည်ဆောက်တဲ့အခါ **form field** အကုန်လုံးကိုဆောက်ပေးဖို့မလိုတော့ပါဘူး။ နောက်တစ်ခေါက်ပြန်ထည့်ပေးဖို့မလိုတော့တဲ့ **field** တွေကို **read only** နဲ့ပဲပြပေးရင်ရပါပြီ။

ဒီတစ်ခါတော့ **page** အသစ်မဆောက်တော့ဘဲ **modal dialog** တစ်ခုလုပ်ပေးပါမယ်။ **batch** တစ်ခုချင်းစီရဲ့ဘေးမှာ **Dispense** ခလုတ်ကိုနှိပ်လိုက်ရင် **modal dialog** ပွင့်လာမယ်။ ဆေးပစ္စည်းထုတ်မယ့်ရက်စွဲ၊ အရေအတွက်၊ ဘယ်ကိုထုတ်ပေးမလဲဆိုတာ **input** ထည့်ပေးရပါမယ်။

ပုံမှာပြထားသလို **button** ထည့်ပါမယ်။



ခလုတ်ထည့်ချင်တဲ့ **table** က **BatchList** မို့လို့ **ItemBatchList.php** ဖိုင်ကိုဖွင့်ပါ။

table method မှာ **column** ရဲ့ကွင်းပိတ်အဆုံးမှာ **action** ထည့်ပေးပါ။ **action** ထဲမှာ **dispense** နာမည်နဲ့ **action** တစ်ခုထည့်ပေးပါ။

```
public function table(Table $table): Table
{
    return $table
        ->query(self::tableQuery())
        ->columns([
            // ...
        ])
        ->actions([
            Action::make('dispense') // use Filament\Tables\Actions\Action;
        ]);
}
```

Action ကိုနှိပ်လိုက်ရင် **modal dialog** ပွင့်လာပြီး **form** ကနေ **user input** လက်ခံစေချင်တဲ့အတွက် **form([])** ထည့်ပေးပါ။ ဆက်လုပ်စေချင်တဲ့ **action** အတွက် **function** ရေးပေးပါ။

```
        ->actions([
            Action::make('dispense')
                                ->form([])
                                ->action(function(array $data, Transaction $record){
                                                dd($data);
                                })
        ]);
```

form([]) ထဲမှာ **Form component** တွေဆက်ရေးပေးပါမယ်။ **action** ထဲမှာ **function closure** ထပ်သုံးခြင်းအားဖြင့် **modal** ထဲထည့်ထားတဲ့ **form** ကနေ **user** ထည့်သွင်းလိုက်တဲ့ **data** တွေကို **array $data parameter** ထဲမှာလက်ခံပါတယ်။ ပြီး **Transaction model** ကို ကိုယ်စားပြုတဲ့ **$record parameter** ကိုသုံးပြီး **create, update** တွေဆက်လုပ်ဆောင်နိုင်ပါတယ်။ လောလောဆယ် **function** ထဲမှာ **dd($data);** ပဲရေးထားပါ။ **browser** မှာ **refresh** လုပ်ပြီး **dispense** ခလုတ်ကိုနှိပ်ကြည့်ရင် **[] empty array** ပဲပေါ်နေပါမယ်။ **Transaction outgoing record** ဟာ အဲဒီ **array** နဲ့သွားသိမ်းမှာဖြစ်ပါတယ်။ ရှေ့ဆက်ပြီး **array** ထဲကို **data** တွေထည့်ပါမယ်။

```
    ->actions([
      Action::make('dispense')
                        ->form([])
                        ->action(function(array $data, Transaction $record){
                                $data['type'] = 'OUT';
                                dd($data);
                        })
    ]);
```

Outgoing record မို့လို့ **type value** ကို **'OUT'** လို့ပုံသေပေးထားပါမယ်။ **save** လုပ်ပြီး **browsre** မှာ **refresh** လုပ်ကြည့်ပါ။ **type => 'OUT'** လို့ပြနေပါမယ်။

Transaction model ကိုသွားကြည့်ရအောင်။ **Transaction model** ထဲက **$fillable** ထဲမှာ ဖြည့်ခဲ့တဲ့ **property** တွေအတိုင်း **form** ဆောက်ပေးရပါမယ်။ တချို့က **read only field** ဖြစ်ပြီး တချို့က **user input** လက်ခံရမယ့် **form field** တွေပါ။

app/Models/Transaction.php

```
protected $fillable = [
    'date', // ပစ္စည်းထုတ်မယ့်နေ့ input လက်ခံရန်
    'type', // 'OUT' တန်ဖိုးပုံသေပေးရန်
    'item_id', // readonly ပြရန်
    'category_id', // readonly ပြရန်
    'package_form_id', // readonly
    'exp_date', // read only
    'batch', // read only
    'amount', // ထုတ်မယ့် အရေအတွက် input လက်ခံရန်
    'donor_id', // read only
    'source', // read only
    'destination', // ပို့မယ့်နေရာ input လက်ခံရန်
    'remarks' // မှတ်ချက် input လက်ခံရန်
];
```

action ထဲက **form** မှာ **Section** နှစ်ခုထည့်ပေးပါ

```
        ->form([
          Section::make('Item Details'),
```

```
            Section::make('Dispense Details'),
        ])
```

ဒါဆို **browser** မှာ **dispense** နှိပ်လိုက်ရင် **modal dialog** ပေါ်လာပြီး **section** နှစ်ခုကိုတွေ့ရပါမယ်။
အပေါ်က **Item Detail section** က **read only field** တွေထည့်ဖို့ဖြစ်ပြီး **Dispense Detail section** ကတော့ **user input** လက်ခံရမယ့် **form field** တွေထည့်ဖို့ဖြစ်ပါတယ်။ **filament form component** ထဲမှာ **read only component** အနေနဲ့ **Placeholder** ကိုသုံးလို့ရပါတယ်။ **Item Detail Section** ထဲမှာ **Item name** အတွက် **placeholder** တစ်ခုထည့်ပေးပါ

```
            Section::make('Item Details')
                ->schema([
                    Placeholder::make('item')
                        ->content(fn($record) => $record->item->name)
                        ->inlineLabel(),
                ]),
```

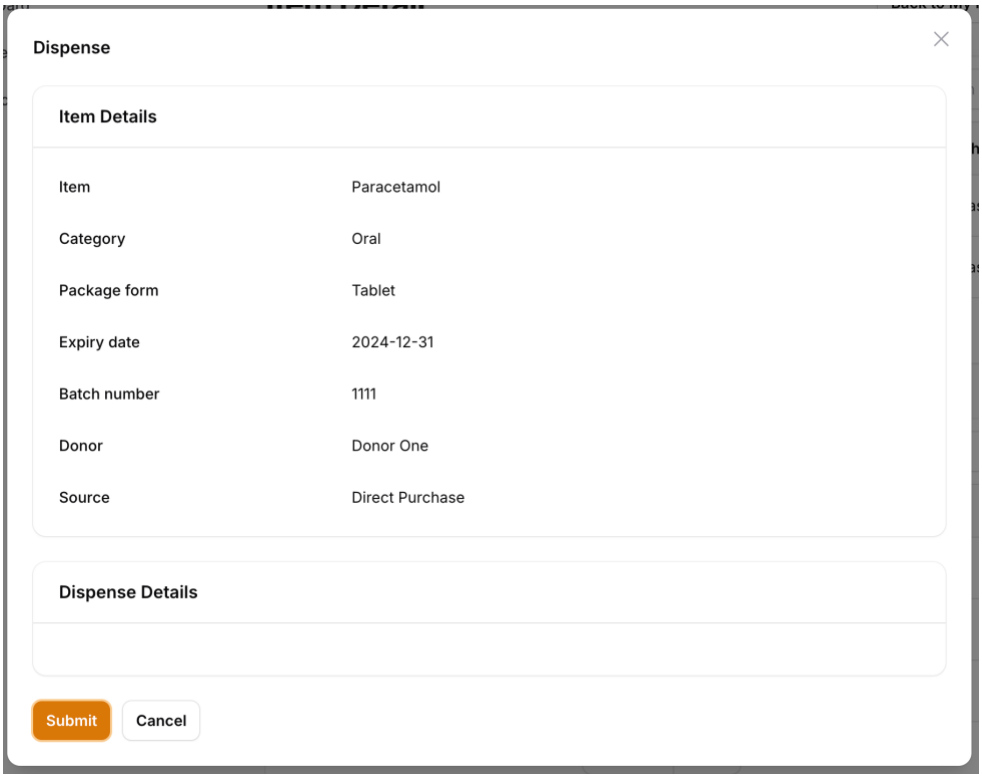
Section ရဲ့ **schema** ထဲမှာ **placeholder** ထည့်ပါမယ်။ **placeholder::make()** ထဲမှာထည့်တဲ့ တန်ဖိုးဟာ သူ့ရဲ့ **label** အနေနဲ့ပြပေးပါတယ်။ **content** ထဲမှာတော့ ပြစေချင်တဲ့တန်ဖိုးကို သတ်မှတ်ပေးရပါတယ်။ ကျွန်တော်တို့ **Dispense** ခလုတ်ကို **click** နှိပ်လိုက်တဲ့ **table row** ရဲ့ **record** ကို ယူချင်တာမို့လို့ **fn($record) closure** ထည့်ပေးရပါတယ်။ ဒီလိုထည့်ပေးပြီးတာနဲ့ **table row** ရဲ့ **item relationship** ထဲက **name** တန်ဖိုးကို **access** လုပ်ပြီး ပြလို့ရသွားပါပြီ။ **inlineLabel()** ထည့်ထားတာကတော့ **Item name label** နဲ့ တန်ဖိုးကို အထက်အောက်မဟုတ်ဘဲ တတန်းတည်းပြစေချင်လို့ပါ။

ဒီတိုင်းပဲ **category, package form, exp date, batch, donor** တွေကိုတန်းစီထည့်သွားပါမယ်။

```
            Section::make('Item Details')
                ->schema([
                    Placeholder::make('item')
                        ->content(fn($record) => $record->item->name)
                        ->inlineLabel(),
                    Placeholder::make('category')
                        ->content(fn($record) => $record->category->name)
                        ->inlineLabel(),
                    Placeholder::make('package form')
                        ->content(fn($record) => $record->packageForm->name)
                        ->inlineLabel(),
                    Placeholder::make('expiry date')
                        ->content(fn($record) => $record->exp_date)
                        ->inlineLabel(),
                    Placeholder::make('batch number')
                        ->content(fn($record) => $record->batch)
                        ->inlineLabel(),
                    Placeholder::make('donor')
                        ->content(fn($record) => $record->donor->name)
                        ->inlineLabel(),
                    Placeholder::make('source')
                        ->content(fn($record) => $record->sourceWarehouse->name)
```

```
            ->inlineLabel(),
        ]),
```

save လုပ်ပြီး **browser** မှာပြန်ကြည့်ပါ။ **modal dialog** မှာ **read only** ပြပေးမယ့်တန်ဖိုးတွေအကုန်မြင်ရပါပြီ။



ပုံမှာမြင်ရတဲ့အတိုင်းဆိုရင် တန်ဖိုးတွေကိုအထက်အောက်ပြပေးနေတာကြောင့် ညာဘက်အခြမ်းမှာနေရာလွတ်တွေပိုနေပါတယ်။ ဒါကြောင့် **Item Detail Section** ကို **column** နှစ်ခုခွဲပြီး တန်ဖိုးတွေကို **column** နှစ်ခုနဲ့ခွဲပြပါမယ်။ အောက်က **section** မှာ **form field** တွေထပ်ထည့်ရမှာမို့လို့ **user** ကို **scroll down** လုပ်စရာမလိုပဲ **data** ဖြည့်စေချင်လို့ပါ။ **Item Detail section** ကို **column** နှစ်ခုသတ်မှတ်ပေးလိုက်ပါ

```
        Section::make('Item Details')
          ->schema([
            // ...
          ])->columns(2),
        Section::make('Dispense Details'),
```

ဒါဆို **Section** ထဲမှာ **column** နှစ်ခုနဲ့ပြပေးမှာဖြစ်ပါတယ်။ ဒါပြီးရင်တော့ **Dispense Details section** ထဲမှာ **date, amount, destination, remarks form field** တွေဆက်ထည့်ပေးပါမယ်။

```
        Section::make('Dispense Details')
          ->schema([
            DatePicker::make('date')
              ->label('Date of Dispense')
              ->required()
              ->native(false),
            TextInput::make('amount')
              ->label('Amount')
              ->required()
```

```
            ->numeric()
            ->maxValue(fn($record) => $record->total_amount),
        Select::make('destination')
            ->label('Destination')
            ->required()
            ->placeholder('Please select destination')
            ->native(false)
            ->relationship('destinationWarehouse', 'name'),
        Textarea::make('remarks')
    ])->columns(2),
```

amount form field မှာ **maxValue() method** ထည့်ထားပါတယ်။ **user** က **amount field** မှာလာထည့်မယ့်တန်ဖိုးကို **validate** လုပ်တဲ့ **method** ဖြစ်ပါတယ်။ **maxValue** ဆိုတဲ့အတိုင်းပဲ အများဆုံးထည့်နိုင်တဲ့ **maximum value** ကိုသတ်မှတ်ပေးပါတယ်။ **user click** နှိပ်လိုက်တဲ့ ဆေးပစ္စည်းရဲ့ **batch** မှာ **total_amount column** တန်ဖိုးရှိပါတယ်။ အဲဒီ **batch** ရဲ့စုစုပေါင်းတန်ဖိုးထက် ပိုထုတ်လို့မရပါဘူး။ ထုတ်လိုက်လို့ရင် **balance** ကို **SUM(amount)** နဲ့တွက်တဲ့အခါမှာ အနုတ်တန်ဖိုး **minus** တွေပြကုန်မှာပါ။ ဒါကြောင့် ထုတ်ချင်တဲ့ **batch** ရဲ့စုစုပေါင်းအရေအတွက်ဟာ ဥပမာအားဖြင့် 1000 ရှိမယ်ဆိုရင် ထုတ်လို့ရတဲ့အရေအတွက်ဟာလည်း 1000 အများဆုံးရှိရပါမယ်။ 1000 ထက်ပေးကျော်လို့မရပါဘူး။
maxValue() ရဲ့ **closure** ထဲမှာ **$record parameter** လက်ခံပါတယ်။ ထုတ်ချင်တဲ့ **batch** ရဲ့ **record** တစ်ခုလုံးပါ။ **return** ပြန်တဲ့ တန်ဖိုးကတော့ **$record** ထဲက **total_amount** ဖြစ်ပါတယ်။ ဒီတော့ **batch** ရဲ့ **total_amount** က 1000 ဆိုရင် **maxValue(1000)** လို့ **return** ပြန်ပါမယ်။ **total_amount** က 500 ဆိုရင် **maxValue(500)** အနေနဲ့ **dynamic value** အဖြစ်အမြဲ **return** ပြန်ပေးနေမှာဖြစ်ပါတယ်။

destination အတွက် **destinationWarehouse relationship** သုံးထားပါတယ်။ **sourceWarehouse relationship** ရှိပြီးသားမို့လို့ ဒါပဲဆက်သုံးမယ်ဆိုလည်းရပါတယ်။ ဒါပေမယ့် **code** ကိုဖတ်ရင် **destination dropdown** မှာ **sourceWarehouse relationship** သုံးထားတာ အဓိပ္ပါယ်ကောက်လွဲချင်စရာမို့လို့ **Transaction model** မှာ **destinationWarehouse** ဆိုတဲ့ **BelongsTo relationship** တစ်ခုဆက်ရေးပေးပါမယ်။
Transaction.php

```
class Transaction extends Model
{
    protected $fillable = [
        // ...
    ];

    //...

    public function destinationWarehouse(): BelongsTo
    {
        return $this->belongsTo(Warehouse::class, 'destination');
    }
}
```

ဒါပေမယ့်လက်ရှိမှာ ကျွန်တော်တို့ရဲ့ **transactions table** ရဲ့ **destination column** ဟာ **string column** ဖြစ်ပါတယ်။ **foreign key** လက်ခံနိုင်တဲ့ **column** မဟုတ်ပါဘူး။ ဒီအချိန်မှာ **destination column** ကို **foreignId** အမျိုးအစားလို့ **transaction migration file** ထဲမှာသွားပြင်လိုက်ရင် တကယ့် **database table** ထဲမှာလိုက်ပြောင်းမှာမဟုတ်ပါဘူး။ **php artisan migrate:rollback command** နဲ့ **database table** ကို ပြန်ဖျက်၊ **migrate file** မှာပြင်ထားတဲ့ **schema** အသစ်နဲ့ **php artisan migrate command** ပြန် **run** ရမှာဖြစ်ပါတယ်။ အခုတခေါက်မှာ အဲဒီလုပ်ငန်းစဉ်အတိုင်းဆက်မလုပ်တော့ဘဲ **table column** ပြင်တဲ့ **migrate file** သီးသန့်ထပ်ရေးကြည့်ပါမယ်။ ဘာလို့လဲဆိုတော့ ကျွန်တော်တို့အစမ်းဖြည့်ထားတဲ့ **transaction record** တွေရှိပြီးသားဖြစ်နေတော့ **php artisan migrate:rollback command run** လိုက်ရင် ဖြည့်ထားတဲ့ **data**

တွေ့ပျက်ကုန်မှာမို့လို့ပါ။ လောလောဆယ် **transaction table** ထဲက **destination column** ထဲမှာ ဘာ **data** မှမရှိသေးတာမို့လို့ **column modify** လုပ်တဲ့ **migrate file** သပ်သပ်ထပ်ရေးပေးလို့ရပါတယ်။
terminal မှာ ဒီ **command run** ကြည့်ပါ။

```
php artisan make:migration alter_destination_column_in_transactions_table
```

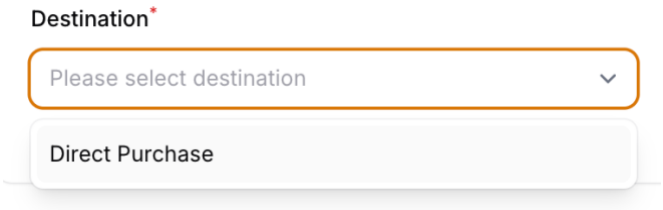
ရလာတဲ့ဖိုင်ကို **Ctrl + click** နှိပ်ပြီးဖွင့်ပါ။ အောက်ပါအတိုင်းရေးပေးပါ။

```
public function up(): void
{
    Schema::table('transactions', function (Blueprint $table) {
        $table->dropColumn('destination');
    });

    Schema::table('transactions', function (Blueprint $table) {
        $table->foreignId('destination')->nullable()->constrained('warehouses')->cascadeOnDelete();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::table('transactions', function (Blueprint $table) {
        $table->dropForeign(['destination']);
        $table->dropColumn('destination');
        $table->string('destination')->nullable();
    });
}
```

public function up() က **migrate** လုပ်မယ့် **method** ဖြစ်ပြီး **public function down()** က **migrate:rollback** လုပ်မယ့် **method** ဖြစ်ပါတယ်။
အရင်ဆုံး **public function up()** ထဲမှာ **transactions table** ထဲက **destination column** တစ်ခုကို **dropColumn()** နဲ့ ဖျက်လိုက်ပါမယ်။
ကျွန်တော်တို့ **database table** ထဲက **destination column** ထဲမှာ ဘာ **value** မှမထည့်ရသေးဘူးမို့လို့ ဒီနေရာမှာ စိတ်ချလက်ချဖျက်လို့ရပါတယ်။
ပြီးရင်တော့ **up() function** ထဲမှာ ပုံမှန်ရေးနေကျအတိုင်း **destination column** ကို **foreignId** အမျိုးအစားနဲ့ **warehouses table** နဲ့ချိတ်ဆက်ပြီးရေးပေးလိုက်ပါမယ်။
အသစ်မှတ်စရာတိုးလာတာက **down() function** ပါ။ **undo** ပြန်လုပ်ပေးရမယ့်လုပ်ငန်းအစဉ်လိုက်အတိုင်းရေးပေးရပါတယ်။ **undo** ပြန်လုပ်ချင်တဲ့အခါမှာ **destination** နာမည်နဲ့ **foreign key constraint** ကိုပြန်ဖျက်ပေးရပါတယ်။ ပြီးမှ **destination column** ကို **drop column** လုပ်ပေးရပါတယ်။ နောက်ဆုံးမှ **original** အတိုင်းရှိခဲ့တဲ့ **string** အမျိုးအစား **destination column** ကိုထည့်ပေးလိုက်ပါတယ်။
down function က မရေးလည်းရပေမယ့် အလေ့အကျင့်ကောင်းတစ်ခုမွေးမြူတဲ့အနေနဲ့ ရေးတဲ့အကျင့်လေးလုပ်ထားသင့်ပါတယ်။
ဒါပြီးရင်တော့ **migrate** လုပ်ပါ။

```
php artisan migrate
```

Save ပြီး **browser** မှာဖွင့်ကြည့်ရင် နမူနာ **destination dropdown** မှာ **Direct purchase** ဆိုပြီး **option** တစ်ခုအနေနဲ့တွေ့ရပါမယ်။



ဘာလို့လဲဆိုတော့ ဆေးပစ္စည်းအဝင်မှတ်တမ်းအတွက် **source dropdown** မှာ **option** တိုးတုန်းက နမူနာအနေနဲ့ **"Direct Purchase"** လို့ တိုးပေးခဲ့လို့ဖြစ်ပါတယ်။ **source warehouse** အတွက်ရွေးစရာတန်ဖိုးမှာ တိုက်ရိုက်ဝယ်တယ်ဆိုတဲ့ **"Direct Purchase"** တန်ဖိုးကိုမြင်နေရတာပြဿနာမရှိပေမယ့် **destination** အတွက်ရွေးတဲ့နေရာမှာ **"Direct Purchase"** ကိုမြင်နေရတာ အဓိပ္ပါယ်မရှိပါဘူး။ ဒါကြောင့် မိမိ **warehouse** စနစ်မှာ **Direct Purchase** လုပ်ရတာရှိလို့ ထည့်ထားတဲ့သူတွေအတွက် **destination option** ထဲမှာ ဒီ **option** ကိုဖယ်ထုတ်ပေးဖို့လိုပါတယ်။ **Direct Purchase option** မလိုလို့ မထည့်ထားတဲ့သူတွေကတော့ ဒီအပိုင်းကို ကျော်သွားလို့ရပါတယ်။

9.1.3 Modifying dropdown options query

destination select ရဲ့ **relationship** ထဲမှာ **modifyQueryUsing() method** တစ်ခုထပ်ထည့်ပေးပါ။

```
Select::make('destination')
    ->label('Destination')
    ->required()
    ->placeholder('Please select destination')
    ->native(false)
    ->relationship(
        name: 'destinationWarehouse',
        titleAttribute: 'name',
        modifyQueryUsing: fn(Builder $query) => $query->where('name', '!=', 'Direct Purchase')
    ),
```

နောက်ကွယ်မှာအလုပ်လုပ်တဲ့ပုံစံကတော့
destinationWarehouse() relationship ဟာ **SQL** နဲ့ဆိုရင်

```
SELECT id, name FROM warehouses;
```

ဒါကို **modifyQueryUsing() method** နဲ့ **filter** ထပ်လုပ်ပါမယ်။ **closure** ထဲမှာ **Builder object** ဖြစ်တဲ့ **$query parameter** ထည့်ပေးပါမယ်။ ဒီတော့ **$query parameter** ထဲမှာ မူရင်း **query** ဖြစ်တဲ့ **SELECT id, name FROM warehouses;** ကိုသယ်လာပေးပါတယ်။ အဲဒါကိုမှ **where('name', '!=', 'Direct Purchase')** နဲ့ **filter** ထပ်လုပ်ပါတယ်။ **modify** လုပ်ပြီးတဲ့ နောက်ကွယ်မှာအလုပ်လုပ်တဲ့ **SQL** က

```
SELECT id, name FROM warehouses WHERE name != 'Direct Purchase';
```

ဒီနေရာမှာ **'Direct Purchase'** စာလုံးပေါင်းက **database table** ထဲမှာ မိမိသိမ်းခဲ့တဲ့ စာလုံးပေါင်းနဲ့ စာလုံးအကြီးအသေး **(case sensitive)** မှန်ကန်မှသာ အလုပ်လုပ်မှာဖြစ်ပါတယ်။
Save လုပ်ပြီး **browser** မှာကြည့်မယ်ဆိုရင် **Destination dropdown** မှာ **'Direct Purchase' option** မပါတော့ပါဘူး။ ရှေ့ဆက်ပြီး **option** တွေထည့်လို့ရအောင် **createOptionForm() method** နဲ့ **name input field** ထည့်ပေးပါမယ်။

```
Select::make('destination')
    ->label('Destination')
    ->required()
    ->placeholder('Please select destination')
```

```
            ->native(false)
            ->relationship(
                name: 'destinationWarehouse',
                titleAttribute: 'name',
                modifyQueryUsing: fn(Builder $query) => $query->where('name', '!=', 'Direct Purchase')
            )
            ->createOptionForm([
                TextInput::make('name')
                    ->label('Destination Warehouse')
                    ->required(),
            ]),
```

Save လုပ်ပြီး **browser refresh** လုပ်ပါ။ **Destination select** ရဲ့ဘေးက (+) ခလုတ်ကိုနှိပ်ပြီး **destination warehouse** တန်ဖိုးတွေထည့်ကြည့်ပါ။ ဒီမှာထည့်တဲ့ **data** နဲ့ **Incoming transaction form** မှာ **source** အတွက်ထည့်ခဲ့တဲ့ **data** တိုင်းဟာ **warehouses table** ထဲကိုပဲ ဝင်မှာဖြစ်ပါတယ်။

9.1.4 Mutating form data

အခုနေ **Submit** ခလုတ်ကိုနှိပ်လိုက်ရင် **dd** ခံထားတဲ့အတွက် **form** ကနေတက်လာတဲ့ **data array** ကို အခုလိုမြင်နိုင်ပါတယ်။

```
[
  "date" => "2024-12-01"
  "amount" => "1000"
  "destination" => 2
  "remarks" => null
  "type" => "OUT"
]
```

ဒီ **data** ဟာ **database table** ထဲသွားသိမ်းလို့မရသေးပါဘူး။ **read only field** ထဲက **data** တွေကိုလည်း **$data array** ထဲထည့်ပေးဖို့ကျန်ပါသေးတယ်။ ထပ်ထည့်ပေးရမယ့် **field** တွေက **item_id, category_id, package_form_id, exp_date, batch, donor_id, source** တို့ဖြစ်ပါတယ်။ **form data** တွေကို အောက်ပါအတိုင်း **mutate** လုပ်ကြည့်ပါမယ်။

```
            ->action(function (array $data, Transaction $record) {
                $data['type'] = 'OUT';
                $data['item_id'] = $record->item_id;
                $data['category_id'] = $record->category_id;
                $data['package_form_id'] = $record->package_form_id;
                $data['exp_date'] = $record->exp_date;
                $data['batch'] = $record->batch;
                $data['donor_id'] = $record->donor_id;
                $data['source'] = $record->source;
                dd($data);
            })
```

action ရဲ့ **closure** ထဲမှာ **$record** တစ်ခုလုံးကို **access** ရနေပြီးသားမို့လို့ **$record** ထဲက ကြိုက်တဲ့ **data** ကိုဆွဲထုတ်ရုံပါပဲ။ **Save** လုပ်ပြီး **submit** ခလုတ်ပြန်နှိပ်ကြည့်ပါ။ နမူနာအနေနဲ့ ဒီလို **array** တစ်ခုရပါမယ်။

```
[
  "date" => "2024-12-01"
  "amount" => "1000"
  "destination" => 2
  "remarks" => null
  "type" => "OUT"
  "item_id" => 1
  "category_id" => 1
  "package_form_id" => 1
  "exp_date" => "2024-12-31"
  "batch" => "1111"
  "donor_id" => 1
  "source" => 1
]
```

ဒီအဆင့်ဟာ **form validation** အဆင့်ကိုကျော်လာပါပြီ။ **amount field** မှာ **maxValue** ပေးထားတာကြောင့် ကိုယ်ထုတ်မယ့် **batch** ရဲ့တကယ်ရှိနေတဲ့အရေအတွက်ထက် ပိုထုတ်လို့မရအောင် **form** ကို **validate** လုပ်ထားပါတယ်။ အခု **dd** နဲ့ဖော်ထားတဲ့ **data** ဟာ **form validation** အောင်မြင်ပြီး **database** ကို သွားမ **save** ရသေးခင်လမ်းမှာဖြတ်စစ်ထားတာပါ။ ဒီနေရာမှာကျွန်တော်တို့ **amount** ကို **mutate** လုပ်ပါမယ်။ အခု **form** ဟာ **outgoing data** တွေကိုပဲသိမ်းတာမို့လို့ ဒီ **form** ထဲကတက်လာတဲ့ **amount** တန်ဖိုးတွေဟာ အနုတ်လက္ခဏာကိန်းပြည့် **(negative integer)** အနေနဲ့ သိမ်းပေးရပါမယ်။ ဒါမှ **aggregate table** တွေမှာ **SUM(amount)** လုပ်ရင် **balance** ထွက်မှာပါ။

amount field ကတက်လာတဲ့တန်ဖိုးတွေကို အနုတ်လက္ခဏာထည့်ရတာလွယ်ပါတယ် **-1** နဲ့မြှောက်ပေးလိုက်ရုံပါပဲ။

```
        ->action(function (array $data, Transaction $record) {
            $data['type'] = 'OUT';
            $data['item_id'] = $record->item_id;
            $data['category_id'] = $record->category_id;
            $data['package_form_id'] = $record->package_form_id;
            $data['exp_date'] = $record->exp_date;
            $data['batch'] = $record->batch;
            $data['donor_id'] = $record->donor_id;
            $data['source'] = $record->source;
            $data['amount'] *= -1; //$data['amount'] = $data['amount'] * -1; နဲ့တူ

            dd($data);
        })
```

$data['amount'] *= -1;** ဟာ **$data['amount'] = $data['amount'] * -1; နဲ့အတူတူပါပဲ။ နဂိုတန်ဖိုးကို **-1** နဲ့မြှောက်လိုက်တာပါ။ **Save** လုပ်ပြီး **submit** ပြန်နှိပ်ကြည့်ပါ။ အားလုံးအဆင်ပြေတယ်ဆိုရင် **dd** လုပ်ထားတဲ့ **line** ကိုပြန်ဖျက်လို့ရပါပြီ။ **$record→create($data)** နဲ့ **save** လုပ်ပါမယ်။

```
        ->action(function (array $data, Transaction $record) {
            $data['type'] = 'OUT';
            $data['item_id'] = $record->item_id;
            $data['category_id'] = $record->category_id;
            $data['package_form_id'] = $record->package_form_id;
```

```
            $data['exp_date'] = $record->exp_date;
            $data['batch'] = $record->batch;
            $data['donor_id'] = $record->donor_id;
            $data['source'] = $record->source;
            $data['amount'] *= -1;
            dd($data);
            $record->create($data);
        })
```

Submit လုပ်လိုက်တဲ့အခါမှာ မူရင်း **table** မှာ ဘာမှမထူးခြားသွားပါဘူး။ ဘာလို့လဲဆိုတော့ **refresh** မဖြစ်လို့ပါ။ **manual refresh** လုပ်ကြည့်လိုက်မယ်ဆိုရင် **table** နှစ်ခုစလုံးမှာ **update** ဖြစ်သွားပါလိမ့်မယ်။ ဒါပေမယ့် ဘယ်ဘက်အစွန်က **Total amount** ထဲက တန်ဖိုးကတော့ **Refresh** လုပ်လည်းမပြောင်းပါဘူး။ ဘာလို့လဲဆိုတော့သူ့ထဲကတန်ဖိုးဟာ **database** ထဲမှာ လက်ရှိဖြစ်နေတဲ့တန်ဖိုးကိုပြပေးတာမဟုတ်ဘဲ **My Inventory page** က လွဲပြောင်းပေးလိုက်တဲ့ **url parameter** တန်ဖိုးအတိုင်းဆက်ပြပေးနေတာမို့လို့ပါ။

ဒါကြောင့် ဆေးပစ္စည်းအထွက်မှတ်တမ်းသိမ်းပြီးတာနဲ့ လက်ရှိ **Item Detail page** ရဲ့မူရင်း **My Inventory page** ကိုခေါ်သွားပေးရပါမယ်။ ဒါမှ **total value** ဟာမူရင်း **page** မှာ **refresh** ဖြစ်မှာပါ။

```
        ->action(function (array $data, Transaction $record) {
            // ...
            $record->create($data);
            redirect()->route('filament.user.pages.my-inventory');
        })
```

save လုပ်ပြီး **browser** မှာ အထွက်မှတ်တမ်းတစ်ခုစမ်းဖြည့်ကြည့်လိုက်ရင် **submit** ခလုတ်နှိပ်လိုက်တာနဲ့ **My Inventory** စာမျက်နှာကိုရောက်သွားပါလိမ့်မယ်။

Destination column ဟာ အခုမှစဖြည့်တာမို့လို့ **Item Transaction List table** က **destination column** မှာ **id** နံပါတ်ကြီးပဲပေါ်နေပါလိမ့်မယ်။ ဒါကို **relationship** ရေးထားတဲ့နာမည်အတိုင်းပြောင်းပေးလိုက်ပါမယ်။

```
        TextColumn::make('destinationWarehouse.name')
            ->searchable(),
```

ဒီအတိုင်းပဲ **TransactionResource.php** ထဲက **destination column** ကို လိုက်ပြောင်းပေးပါ။ ပြီးသွားရင်တော့ **destination warehouse** နာမည်အတိုင်းပြပေးပါလိမ့်မယ်။

ဒီနေရာမှာ **CRUD operation** တွေလုံလောက်ပါပြီ။ တကယ့် **production level application** မှာတော့ အများကြီးဆက်လုပ်ဖို့ကျန်သေးပေမယ့် လေ့ကျင့်ခန်းအတွက်ဆိုရင်တော့ ဒီလောက်နဲ့တင်လုံလောက်ပြီထင်ပါတယ်။ နောက်တစ်ပိုင်းမှာ **Dashboard visualization** တွေဆက်လုပ်ဖို့ရှိတာကြောင့် **Data** တွေများလာတဲ့အခါ ဘယ်လိုအနေအထားမျိုးရှိမယ်ဆိုတာ ကြိုသိနိုင်အောင်ရယ်၊ **Visualize** လုပ်ဖို့ဆိုရင် **data** များမှ ပြလို့အဆင်ပြေမှာကြောင့် နမူနာ **data** တွေဖန်တီးတဲ့အပိုင်းကို ဖော်ပြသွားပါမယ်။

10 Generate Dummy data

data တွေအများကြီးကိုယ်တိုင်ဖြည့်ဖို့ အချိန်ယူရမှာကြောင့် နမူနာ **data** တွေကို အလိုအလျောက်ဖန်တီးဖို့ **Laravel Seeder** ကိုသုံးရပါတယ်။ အထူးသဖြင့် **test data** တွေကို စမ်းသပ်ထည့်သွင်းဖို့အတွက် အဓိကသုံးပါတယ်။ တချို့လိုအပ်ချက်တွေက **data** များလာမှသိသာတာမို့လို့ပါ။

Seeder ဖန်တီးတဲ့အဆင့်တွေကတော့

```
php artisan make:seeder <Model>Seeder // <Model> ကို မိမိပြုလုပ်ချင်တဲ့ Model နာမည်နဲ့အစားထိုးရန်
```

ပြီးရင် **Seeder class** မှာ ဖန်တီးချင်တဲ့ပုံစံကိုသတ်မှတ်ပေးရပါမယ်။

```
php artisan db:seed --class=<Model>Seeder
```

10.1 CategorySeeder

ပထမဆုံး **Category** တွေဖန်တီးဖို့ **command run** ပါမယ်။

```
php artisan make:seeder CategorySeeder
```

CatgorySeeder.php ရဲ့ **up() function** ထဲမှာ ထည့်ရေးပေးပါ။

```
  public function run(): void
  {
    $categories = ['Oral', 'Injection', 'Topical', 'Equipment', 'Supplies'];

    foreach ($categories as $category) {
      Category::firstOrCreate(['name' => $category]);
    }
  }
```

$categories array ထဲမှာ **database** ထဲထည့်ချင်တဲ့ **category** တွေကို **string** တန်ဖိုးတွေနဲ့ သိမ်းထားပါတယ်။ **foreach loop** နဲ့ **array item** တစ်ခုချင်းစီကို ပတ်ပေးပါတယ်။ **Category::firstOrCreate(['name' => $category])** မှာ **first** က **database** ထဲမှာ ရှိပြီးသား **category** ဟုတ်မဟုတ် စစ်ဆေးပါတယ်။ မရှိသေးဘူးဆိုရင် **orCreate** နဲ့ အသစ်တစ်ခု **create** လုပ်ပါတယ်။ ရှိပြီးသားဆိုရင်တော့ ထပ်ပြီး **create** မလုပ်တော့ပါဘူး။ ဒီ **class** ကို **run** မယ်ဆိုရင် -

```
php artisan db:seed --class=CategorySeeder
```

နောက် **Seeder** တွေကိုလည်း ဒီအဆင့်တွေအတိုင်းပဲ ဆက်လုပ်သွားပါမယ်။

10.2 PackageFormSeeder

```
php artisan make:seeder PackageFormSeeder

  public function run(): void
  {
    $packageForms = ['Tablet', 'Capsule', 'Ampoule', 'Bottle', 'Tube', 'Sheet', 'Piece', 'Pair'];

    foreach ($packageForms as $packageForm) {
      PackageForm::firstOrCreate(['name' => $packageForm]);
    }
  }

php artisan db:seed --class=PackageFormSeeder
```

10.3 DonorSeeder

```
php artisan make:seeder DonorSeeder
```

```
  public function run(): void
  {
    $donors = ['Donor One', 'Donor Two', 'Donor Three', 'Donor Four', 'Donor Five'];

    foreach ($donors as $donor) {
      Donor::firstOrCreate(['name' => $donor]);
    }
  }
```

```
php artisan db:seed --class=DonorSeeder
```

10.4 WarehouseSeeder

```
php artisan make:seeder WarehouseSeeder
```

```
  public function run(): void
  {
    $warehouses = ['Warehouse One', 'Warehouse Two', 'Warehouse Three', 'Warehouse Four', 'Warehouse Five'];

    foreach ($warehouses as $warehouse) {
      Warehouse::firstOrCreate(['name' => $warehouse]);
    }
  }
```

```
php artisan db:seed --class=WarehouseSeeder
```

10.5 ItemSeeder

```
php artisan make:seeder ItemSeeder
```

Items table မှာတော့ **item name** နဲ့ **category_id** ကိုပါ **generate** လုပ်ရတာဖြစ်ပါတယ်။ ဒီနေရာမှာအသေးစိတ်မရှင်းတော့ပါဘူး။ ဆေးနာမည်တွေနဲ့ အနီးစပ်ဆုံးဆင်တူအောင် **generate** လုပ်နေတယ်လို့ပဲမှတ်ထားပေးပါ။

```
  public function run(): void
  {
    $categories = Category::all();

    foreach ($categories as $category) {
      for ($i = 0; $i < 10; $i++) { // category တစ်ခုမှာ ဆေး (၁၀) မျိုးဖန်တီးမယ်

        Item::create([
          'name' => $this->generateMedicineName(),
          'category_id' => $category->id,
        ]);
      }
    }
  }
```

```
  private function generateMedicineName(): string
  {
    $prefixes = ['Neo', 'Oxy', 'Phen', 'Hydro', 'Meth', 'Dex', 'Levo'];
    $roots = ['cort', 'derm', 'cillin', 'cycline', 'dryl', 'zole', 'prin'];
    $suffixes = ['ol', 'in', 'ate', 'ide', 'one', 'ine', 'il'];

    $name = $prefixes[array_rand($prefixes)] .
        $roots[array_rand($roots)] .
        $suffixes[array_rand($suffixes)];

    return ucfirst(strtolower($name));
  }
```

```
php artisan db:seed --class=ItemSeeder
```

10.6 TransactionSeeder

```
php artisan make:seeder TransactionSeeder
```

```
public function run(): void
  {
    // all() method နဲ့ database ထဲက record အားလုံးဆွဲထုတ်မယ်

    $items = Item::all();
    $donors = Donor::all();
    $warehouses = Warehouse::all();
    $packageForms = PackageForm::all();

    foreach ($items as $item) {
      // property အတွဲတစ်စုံနဲ့ ဆေးပစ္စည်းတစ်မျိုးယာယီတည်ဆောက်မယ်

      $tempItem = [
        'item_id' => $item->id,
        'category_id' => $item->category_id,
        'package_form_id' => $packageForms->random()->id,
        'exp_date' => now()->addMonths(rand(6, 24)),
        'batch' => strtoupper(substr(str_shuffle('ABCDEFGHIJKLMNOPQRSTUVWXYZ'), 0, 3)) . rand(1000, 9999),
        'donor_id' => $donors->random()->id,
        'source' => $warehouses->random()->id,
      ];

      // 'IN' transactions အရင်လုပ်မယ်

      $inCount = rand(1, 3);  // 'IN' transactions ၁ ကြိမ်မှ ၃ ကြိမ် ကျပန်းဖြစ်နိုင်

      for ($i = 0; $i < $inCount; $i++) {
        Transaction::create([
          'date' => now()->subDays(rand(30, 365)),
```

```
            'type' => 'IN',
            'item_id' => $tempItem['item_id'],
            'category_id' => $tempItem['category_id'],
            'package_form_id' => $tempItem['package_form_id'],
            'exp_date' => $tempItem['exp_date'],
            'batch' => $tempItem['batch'],
            'amount' => rand(100, 1000),
            'donor_id' => $tempItem['donor_id'],
            'source' => $tempItem['source'],
            'destination' => null,
            'remarks' => 'Stock in for ' . $item->name,
        ]);
    }

    // 'OUT' transactions ဆက်လုပ်မယ်

    $outCount = rand(2, 5);  // 'OUT' transactions ၂ ကြိမ် ၅ ကြိမ်အတွင်း ကျပန်းဖြစ်နိုင်

    for ($i = 0; $i < $outCount; $i++) {
        Transaction::create([
            'date' => now()->subDays(rand(1, 29)),  // 'IN' transactions တွေထက် နောက်ကျမှဖြစ်ဖို့လို

            'type' => 'OUT',
            'item_id' => $tempItem['item_id'],
            'category_id' => $tempItem['category_id'],
            'package_form_id' => $tempItem['package_form_id'],
            'exp_date' => $tempItem['exp_date'],
            'batch' => $tempItem['batch'],
            'amount' => -rand(10, 50),  // 'OUT' မို့လို့ အနုတ်လက္ခဏာလို၊ 'IN' တုန်းက amount ထက်နည်းရမယ်

            'donor_id' => $tempItem['donor_id'],
            'source' => $tempItem['source'],
            'destination' => $warehouses->where('id', '!=', $tempItem['source'])->random()->id,
            'remarks' => 'Stock out for ' . $item->name,
        ]);
    }
  }
}
```

```
php artisan db:seed --class=TransactionSeeder
```

အားလုံး **generate** လုပ်ပြီးရင်တော့ **MyInventory** စာမျက်နှာမှာ ဆေးပစ္စည်းတွေအစုံရောက်နေမှာဖြစ်ပါတယ်။

11 Dashboard Widgets

CRUD အပိုင်းပြီးသွားတဲ့နောက်မှာ **Dashboard** မှာ **Chart** တွေနဲ့ **data visualization** လုပ်ပါမယ်။ **Filament dashboard** ရဲ့ **widget** တွေအသုံးပြုပြီးထည့်နိုင်ပါတယ်။

Nearest Expiry - သက်တမ်းကုန်ရက်အနီးဆုံးဆေးပစ္စည်းတွေကို အစဉ်လိုက်စီပြီးပြပါမယ်။ **table widget** တည်ဆောက်ပေးပါမယ်။
Donor percentage - ဆေးပစ္စည်းတွေကို ဘယ်အလှူရှင်အဖွဲ့အစည်းရဲ့ **budget** အောက်ကဝယ်တာလဲဆိုတာကို **Pie chart** လေးနဲ့ **proportion** ပြချင်ပါတယ်။
Distribution percentage - ဆေးပစ္စည်းတွေကို ဘယ် **warehouse** တွေဆီအများဆုံးပေးသလဲဆိုတာကို **Pie chart** နဲ့ **proportion** ပြပါမယ်။
Top 10 Items - မိမိ **warehouse** ထဲမှာ အရေအတွက် အများဆုံးပစ္စည်း (၁၀) မျိုးကို **bar chart** နဲ့ပြချင်ပါတယ်။

Dashboard မှာ **widget** တွေအသစ်မထည့်ခင် **default widget** တွေကိုဖျောက်ပေးပါမယ်။ **Ctrl + P > UserPanelProvider.php** ဖွင့်ပါ။ **panel configuration** ထဲက **widgets([])** အပိုင်းကိုရှာပါ။

```
public function panel(Panel $panel): Panel
{
    return $panel
        // ...
        ->widgets([
            Widgets\\AccountWidget::class,
            Widgets\\FilamentInfoWidget::class,
        ])
        // ...
        ;
}
```

default widget နှစ်ခုကို **select** လုပ်ပြီး **Ctrl + /** နှိပ်ပါ။ **comment** ပိတ်လိုက်တဲ့သဘောပါ

```
        ->widgets([
            // Widgets\\AccountWidget::class,
            // Widgets\\FilamentInfoWidget::class,
        ])
```

UserPanelProvider.php ကို **save** လုပ်ပြီးပိတ်လို့ရပါပြီ

11.1 Nearest Expiry

Table widget လုပ်တဲ့ **command**

```
php artisan make:filament-widget NearestExpiry --table
```

ဘယ် **resource** မှာလုပ်မလဲမေးရင် **blank** ထားပြီး **enter** နှိပ်ပါ။ ဘယ် **panel** မှာလုပ်မလဲမေးရင် **user panel** မှာပဲမို့လို့ **enter** နှိပ်ပါ။
NearestExpiry.php ဖိုင်ဖွင့်ပါ။ **Transaction::query()** ထည့်ပေးပါ။

```
class NearestExpiry extends BaseWidget
{
    public function table(Table $table): Table
    {
        return $table
            ->query(
                Transaction::query()
            )
```

```
        ->columns([
            // ...
        ]);
    }
```

Save လုပ်ပြီး **browser** မှာကြည့်လိုက်ရင် **Dashboard** မှာ **table widget** ပေါ်လာပါလိမ့်မယ်။

Transaction query မှာ **modify** လုပ်ပေးပါ။

```
    public function table(Table $table): Table
    {
        return $table
            ->query(
                Transaction::query()
                ->selectRaw('
                min(id) as id,
                item_id,
                category_id,
                package_form_id,
                exp_date,
                SUM(amount) as total_amount
                ')
                ->groupBy('item_id', 'category_id', 'package_form_id', 'exp_date')
                ->havingRaw('sum(amount) > 0')
                ->whereNotNull('exp_date')
                ->orderBy('exp_date', 'asc')
                ->limit(5)
            )
            ->columns([
                // ...
            ]);
    }
```

Aggregate လုပ်မှာမို့လို့ **selectRaw()** နဲ့ **query** ဆက်လုပ်ပါမယ်။ **primary key** ပါသွားအောင် **min(id)** ပါထည့်ပေးပါမယ်။ **groupBy()** ကတော့ **SUM()** နဲ့ **aggregate** မလုပ်တဲ့ ကျန် **column** တွေကိုလုပ်ပေးရပါမယ်။ **havingRaw()** မှာ **SUM(amount) > 0** ထည့်ပေးထားတာက တကယ်ရှိနေတဲ့ပစ္စည်းတွေကိုပဲလိုချင်လို့ပါ။ သုံးရင်ကုန်သွားလို့ **balance 0** ဖြစ်နေတဲ့ပစ္စည်းတွေကို ထည့်မတွက်ချင်ပါဘူး။ **whereNotNull('exp_date')** ကတော့ **exp_date** တန်ဖိုးထည့်ထားတဲ့ပစ္စည်းကိုပဲပြချင်ပါတယ်။ တချို့ **exp_date** မပါတဲ့ ကတ်ကြေး၊ စတီးခွက်စတာတွေက သက်တမ်းလွန်မှာမဟုတ်လို့ ထည့်မပြချင်ပါဘူး။ **orderBy('exp_date', 'asc')** နဲ့ **earliest to latest** စီပါတယ်။ **Expiry date** လို အနီးဆုံးကစစီမယ်ဆိုရင် **earliest to latest** စီတဲ့ **ascending** သုံးရပြီး **latest news, latest posts** စတာတွေမှာတော့ **latest to earliest** စီတဲ့ **descending** သုံးရပါတယ်။ **limit(5)** က **query result** ကို ငါးခုအထိပဲပြပေးပါတယ်။

ပြီးရင် **column** တွေဆက်ထည့်ပေးပါ။

```
        ->columns([
            TextColumn::make('item.name'),
            TextColumn::make('category.name'),
```

```
            TextColumn::make('exp_date'),
            TextColumn::make('total_amount'),
            TextColumn::make('packageForm.name'),
        ])
```

save ပြီး **browser** မှာကြည့်ရင် **expiry** အနီးဆုံးပစ္စည်းတွေ မြင်ရပါမယ်။ **limit 5** လုပ်ထားပေမယ့် **table pagination** အတိုင်း **10 row** ပြနေပါလိမ့်မယ်။ **table pagination** မှာ ပြောင်းရွေးရင်ရွေးသလောက် **row** အရေအတွက်ပြပေးပါလိမ့်မယ်။
limit(5) လုပ်ထားတဲ့အတိုင်း ငါးခုပဲပြပေးအောင် **table pagination** ကိုပိတ်ပေးရပါမယ်။

```
        ->columns([
            TextColumn::make('item.name'),
            TextColumn::make('category.name'),
            TextColumn::make('exp_date'),
            TextColumn::make('total_amount'),
            TextColumn::make('packageForm.name'),
        ])
        ->paginated(false);
```

ဒါဆိုရင် **5 row** ပဲပြပေးမှာပါ။
number တန်ဖိုး **total_amount** ကို **align end** လုပ်ပေးပါမယ်။ **thousand seperator comma** ထည့်ပေးပါမယ်။

```
            TextColumn::make('total_amount')
                ->numeric(thousandsSeparator:',')
                ->alignEnd(),
```

exp_date ကို **badge style** နဲ့ **conditional color** သတ်မှတ်ပေးပါမယ်

```
            TextColumn::make('exp_date')
                ->badge()
                ->color(function ($state) {
                    if ($state >= today()->addMonths(3)) {
                        return 'info';
                    } else if ($state < today()->addMonths(3) && $state >= today()->addMonth()) {
                        return 'warning';
                    } else {
                        return 'danger';
                    }
                }),
```

badge() က **badge style** ပြောင်းပေးပြီး **color()** မှာ တန်ဖိုးသတ်မှတ်ပေးလို့ရပါတယ်။ **Filament** မှာ **tailwind css color** တွေသုံးထားပြီး **main color** တွေက
success - အစိမ်းရောင်
info - အပြာရောင်
warning - အဝါရောင်
danger - အနီရောင် ဖြစ်ပါတယ်။ **color** တန်ဖိုးကို **'success', 'danger'** စသည် အသေသတ်မှတ်ပေးလို့ရသလို **function closure** နဲ့ **dynamic** ပေးလို့လည်းရပါတယ်။

exp_date သုံးလကျော်ကျန်သေးရင် **$state >= today()->addMonths(3)** အပြာရောင်ပြပါမယ်။

exp_date တစ်လမှသုံးလအတွင်းကျန်သေးရင် **$state < today()->addMonths(3) && $state >= today()->addMonth()** အဝါရောင်ပြပါမယ်။

exp_date တစ်လအောက်ဆိုရင်တော့ အနီရောင်ပြပါမယ်။

table ခေါင်းစဉ်ကို **Top 5 nearest expiry** လို့ပြောင်းသွားအောင် **heading()** ထည့်ပေးပါ။

```
public function table(Table $table): Table
{
    return $table
        ->heading('Top 5 Nearest Expiry')
        ->query(
            // ...
        )
        ->columns([
            // ...
        ])
        // ...
        ;
}
```

11.2 Donor contribution

Chart widget ဖန်တီးဖို့ **command run** ပါ

```
php artisan make:filament-widget DonorContributionChart --chart
```

resource မှာ **blank** ထားပြီး **enter** နှိပ်ပါ။ **user panel** မှာ **enter** နှိပ်ပါ။ **chart** အမျိုးအစားမေးရင် **up or down arrow key** နဲ့ **pie chart** ရွေး **enter** နှိပ်ပါ။

DonorContributionChart.php ဖွင့်ပါ။ **getData function** ကို **update** လုပ်ပေးပါ။

```
protected function getData(): array
{
    $data = [];
    $labels = [];
    return [
        'datasets' => [
            [
                'label' => 'Donor contribution',
                'data' => $data,
            ]
        ],
        'labels' => $labels,
    ];
}
```

function ရဲ့အစမှာကတည်းက **$data** နဲ့ **$labels empty array** တွေဖန်တီးလိုက်ပါတယ်။ **$data array** က **chart** မှာပြမယ့်တန်ဖိုးတွေအတွက်ဖြစ်ပြီး **$labels array** က အညွှန်းစာသားတွေဖြစ်ပါတယ်။ **return** ပြန်မယ့် **array** ထဲမှာ **'datasets' key** နဲ့ **'labels' key** ထည့်ပေးရပါတယ်။ **Filament**

မှာသုံးတဲ့ **visualization** ဟာ **Chart.js library** ကိုယူသုံးထားတာမို့လို့ ဒီ **array format** ဖွဲ့စည်းပုံအကြောင်း အသေးစိတ်ကို **Chart.js documentation** ထဲမှာဖတ်ကြည့်လို့ရပါတယ်။

'datasets' key ထဲက **'label' key** ဟာ **chart** ရဲ့ **label** မို့လို့ ပုံသေပေးထားလို့ရပါတယ်။ **$data key** ကတော့ **dynamic** ဖြစ်လို့ **database query result** လာထည့်ပေးရမှာပါ။

'labels' key ကတော့ **chart** ထဲက **data** တစ်ခုစီရဲ့ **label** မို့လို့ **donor name** တွေကို **database** ထဲက **result** တွေထည့်ပေးရမှာပါ။

transactions table ထဲက **donor** တန်ဖိုးတွေဆွဲထုတ်ပေးပါ။

```
protected function getData(): array
{
    $data = [];
    $labels = [];
    $transactionsWithDonors = Transaction::query()
        ->selectRaw('
        donor_id,
        SUM(amount) as total_amount
        ')
        ->groupBy('donor_id')
        ->where('type', 'IN')
        ->orderBy('total_amount', 'desc')
        ->get();
    return [
        // ...
    ];
}
```

Donor နာမည်တွေပြဖို့ဆိုပေမယ့် **donors table** ထဲကဆွဲထုတ်ရင် နာမည်ပဲရမှာပါ။ ကိုယ့်ဆီဝင်လာတဲ့ ပစ္စည်းတွေထဲက ဘယ် **donor** ကပေးတာလဲဆိုတဲ့ **data** ကိုဆွဲထုတ်ရမှာမို့လို့ **transactions table** ထဲက ကိုယ့်ဆီကိုဝင်လာတဲ့ **'IN' record** အားလုံးကို **where('type', 'IN')** နဲ့ထုတ်ပါတယ်။ **selectRaw()** နဲ့ **donor_id,** သူတို့ဆီက **'IN' record** တွေရဲ့ **SUM(amount)** ယူပါတယ်။ **groupBy()** မှာတော့ **SUM()** မလုပ်ဘဲကျန်တဲ့ **column** ကိုထည့်ပေးရပါတယ်။ နောက်ဆုံးမှာတော့ **'IN' amount** အများဆုံးကနေ အနည်းဆုံးကို **'desc'** နဲ့စီလိုက်ပါတယ်။ **Builder query** အမျိုးအစားမို့လို့ နောက်ဆုံးမှာ **get()** နဲ့ပိတ်ပေးပါ။

$transactionsWithDonors ထဲက တန်ဖိုးတွေကို **loop** ပတ်ပေးပါတော့မယ်။

```
protected function getData(): array
{
    $data = [];
    $labels = [];
    $transactionsWithDonors = Transaction::query()
        ->selectRaw('
        donor_id,
        SUM(amount) as total_amount
        ')
        ->groupBy('donor_id')
        ->where('type', 'IN')
        ->orderBy('total_amount', 'desc')
```

```
        ->get();

    foreach ($transactionsWithDonors as $transaction){
        $data[] = $transaction->total_amount;
        $labels[] = $transaction->donor->name;
    }
    return [
        // ...
    ];
}
```

$transactionsWithDonors ထဲမှာဝင်နေတဲ့ **row** တစ်ခုချင်းစီကို **foreach loop** ထဲထည့်ပတ်လိုက်ပါတယ်။ **empty** ဖြစ်နေတဲ့ **$data array** ထဲကို **row** တစ်ခုချင်းစီရဲ့ **total_amount** တွေလာထည့်ပေးပြီး **$lable array** ထဲကိုတော့ **donor relationship** ကနေတဆင့် **name** တန်ဖိုးတွေလာထည့်ပေးလိုက်ပါတယ်။

တန်ဖိုးတစ်ခု **color** တစ်မျိုးစီသတ်မှတ်ပေးပါမယ်။

```
protected function getData(): array
{
    // ...
    return [
        'datasets' => [
            [
                'label' => 'Donor contribution',
                'data' => $data,
                'borderWidth' => 1,
                'backgroundColor' => [
                    'rgba(70, 130, 180, 0.7)',
                    'rgba(0, 128, 128, 0.7)',
                    'rgba(100, 149, 237, 0.7)',
                    'rgba(0, 206, 209, 0.7)',
                    'rgba(106, 90, 205, 0.7)',
                    'rgba(72, 209, 204, 0.7)',
                    'rgba(65, 105, 225, 0.7)',
                ],
                'borderColor' => [
                    'rgba(70, 130, 180, 1)',
                    'rgba(0, 128, 128, 1)',
                    'rgba(100, 149, 237, 1)',
                    'rgba(0, 206, 209, 1)',
                    'rgba(106, 90, 205, 1)',
                    'rgba(72, 209, 204, 1)',
                    'rgba(65, 105, 225, 1)',
                ],
            ]
```

```
        ],
        'labels' => $labels,
      ];
  }
```

ဒီမှာသတ်မှတ်ပေးထားတဲ့ **property** တွေဟာ **Chart.js** ရဲ့ **documentation** ထဲကအတိုင်းလိုက်နာထားတာဖြစ်ပါတယ်။ **borderWidth** က **pie chart** ထဲက **slice** တစ်ခုစီအကြား **border** အထူသတ်မှတ်ပေးပါတယ်။ **backgroundColor array** က **slice** တွေရဲ့ **color variation** ကိုသတ်မှတ်ပေးပြီး **borderColor** ကတော့ **width** 1 ရှိတဲ့ **border** အရောင်ဖြစ်ပါတယ်။
ဒါပြီးရင် **getOptions() method** တစ်ခုထည့်ပေးပါ။

```
class DonorContributionChart extends ChartWidget
{
  protected static ?string $heading = 'Chart';

  protected function getData(): array
  {
    // ...
  }

  protected function getType(): string
  {
    // ...
  }

  protected function getOptions(): array
  {
    return [
      'plugins' => [
        'legend' => [
          'position' => 'bottom',
        ],
      ],
      'responsive' => true,
      'maintainAspectRatio' => false,
      'scales' => [
        'x' => ['display' => false],
        'y' => ['display' => false]
      ],
    ];
  }
}
```

pie chart ကို **configuration** ထပ်သတ်မှတ်ပေးတာဖြစ်ပါတယ်။ ပုံစာ **legend** ကို အောက်မှာပေါ်မယ်။ **responsive** မှာ **true** ပေးထားတာက **screen size** ပေါ်မူတည်ပြီး **chart size** ပြောင်းလဲပေးဖို့ပါ။ **maintainAspectRatio false** ပေးထားလို့ **responsive** ပိုဖြစ်စေပါတယ်။ **scales**

ဆိုတာက **axis** ပေါ်က **tick mark** တွေဖြစ်ပါတယ်။ **pie chart** လိုမျိုးက **tick mark** တွေရှိဖို့မလိုဘူးမို့လို့ **X, Y** နှစ်ခုစလုံးမှာ **display false** ပေးထားပါတယ်။

နောက်ဆုံး **$heading** မှာပြောင်းပေးပါ။

```
protected static ?string $heading = 'Donor Contribution';
```

Save ပြီး **refresh** လုပ်ပါ။

11.3 Distribution

Distributed လုပ်ပေးလိုက်တဲ့ **warehouse** အချိုးအတွက် **pie chart** ပဲဆက်လုပ်ပေးပါမယ်။

```
php artisan make:filament-widget DistributionChart --chart
```

resource -> blank, user panel နဲ့ **pie chart** ရွေးပါ။

DistributionChart.php ကိုအောက်ပါအတိုင်းထည့်ပေးပါ။ အရင် **pie chart** နဲ့ သဘောချင်းအတူတူပဲမို့လို့ အသေးစိတ်မရေးတော့ပါဘူး

```
class DistributionChart extends ChartWidget
{
  protected static ?string $heading = 'Distributed Warehouses';

  protected function getData(): array
  {
    $data = [];
    $labels = [];

    $transactionsWithDistributions = Transaction::query()
      ->selectRaw('
      destination,
      SUM(amount) as total_amount
      ')
      ->groupBy('destination')
      ->where('type', 'OUT')
      ->orderBy('total_amount', 'desc')
      ->get();

    foreach ($transactionsWithDistributions as $transaction) {
      $data[] = $transaction->total_amount;
      $labels[] = $transaction->destinationWarehouse->name;
    }
    return [
      'datasets' => [
        [
          'label' => 'Distributed Warehouses',
          'data' => $data,
          'borderWidth' => 1,
```

```
            'backgroundColor' => [
                'rgba(255, 99, 71, 0.2)',
                'rgba(255, 165, 0, 0.2)',
                'rgba(255, 215, 0, 0.2)',
                'rgba(255, 69, 0, 0.2)',
                'rgba(255, 140, 0, 0.2)',
                'rgba(255, 127, 80, 0.2)',
                'rgba(255, 99, 71, 0.2)',
            ],
            'borderColor' => [
                'rgba(255, 99, 71, 1)',
                'rgba(255, 165, 0, 1)',
                'rgba(255, 215, 0, 1)',
                'rgba(255, 69, 0, 1)',
                'rgba(255, 140, 0, 1)',
                'rgba(255, 127, 80, 1)',
                'rgba(255, 99, 71, 1)',
            ],
        ]
    ],
    'labels' => $labels,
  ];
}

protected function getType(): string
{
  return 'pie';
}

protected function getOptions(): array
{
  return [
    'plugins' => [
      'legend' => [
        'position' => 'bottom',
      ],
    ],
    'responsive' => true,
    'maintainAspectRatio' => false,
    'scales' => [
      'x' => ['display' => false],
      'y' => ['display' => false]
    ],
  ];
```

```
    }
}
```

11.4 Top 10 Items

အရေအတွက်အများဆုံး item 10 ခုအတွက် bar chart လုပ်ပေးပါမယ်။ command ကအတူတူပါပဲ။ chart အမျိုးအစားရွေးတဲ့အခါမှ bar chart ရွေးပေးပါ။

```
php artisan make:filament-widget TopItemsChart --chart

class TopItemsChart extends ChartWidget
{
    protected static ?string $heading = 'Top 10 Stocked Items';

    protected function getData(): array
    {
        $data = [];
        $labels = [];
        $transactionsWithItems = Transaction::query()
            ->selectRaw('
            item_id,
            category_id,
            SUM(amount) as total_amount
            ')
            ->groupBy('item_id', 'category_id')
            ->havingRaw('SUM(amount) > 0')
            ->orderBy('total_amount', 'desc')
            ->limit(10)
            ->get();

        foreach ($transactionsWithItems as $transaction) {
            $data[] = $transaction->total_amount;
            $labels[] = $transaction->item->name . ' (' . $transaction->category->name . ')';
        }

        return [
            'datasets' => [
                [
                    'label' => 'Most Stocked Items',
                    'data' => $data,
                    'backgroundColor' => [
                        'rgba(70, 130, 180, 0.7)',  // Steel Blue
                        'rgba(0, 128, 128, 0.7)',   // Teal
                        'rgba(100, 149, 237, 0.7)', // Cornflower Blue
                        'rgba(0, 206, 209, 0.7)',   // Dark Turquoise
```

```
                'rgba(106, 90, 205, 0.7)',  // Slate Blue
                'rgba(72, 209, 204, 0.7)',  // Medium Turquoise
                'rgba(65, 105, 225, 0.7)',  // Royal Blue
                'rgba(30, 144, 255, 0.7)',  // Dodger Blue
                'rgba(32, 178, 170, 0.7)',  // Light Sea Green
                'rgba(95, 158, 160, 0.7)',  // Cadet Blue
            ],
            'borderColor' => [
                'rgba(70, 130, 180, 1)',    // Steel Blue
                'rgba(0, 128, 128, 1)',     // Teal
                'rgba(100, 149, 237, 1)',   // Cornflower Blue
                'rgba(0, 206, 209, 1)',     // Dark Turquoise
                'rgba(106, 90, 205, 1)',    // Slate Blue
                'rgba(72, 209, 204, 1)',    // Medium Turquoise
                'rgba(65, 105, 225, 1)',    // Royal Blue
                'rgba(30, 144, 255, 1)',    // Dodger Blue
                'rgba(32, 178, 170, 1)',    // Light Sea Green
                'rgba(95, 158, 160, 1)',    // Cadet Blue
            ],
            'borderWidth' => 1
        ]
    ],
    'labels' => $labels,
  ];
}

protected function getType(): string
{
  return 'bar';
}

protected function getOptions(): array
{
  return [
    'indexAxis' => 'y',
    'scales' => [
      'x' => [
        'grid' => [
          'display' => false
        ],
      ],
      'y' => [
        'grid' => [
          'display' => false
```

```
                ],
            ],
        ],
        'plugins' => [
            'legend' => [
                'display' => false,
            ],
        ],
    ];
  }
}
```

11.5 Sorting Widgets

Widget တွေကို **sorting** စီဖို့အတွက် **$sort property** သုံးနိုင်ပါတယ်။ သက်ဆိုင်ရာ **widget class** တွေထဲမှာ 1, 2, 3 စသည် အစဉ်လိုက်သတ်မှတ်ပေးသွားရုံပါပဲ

```
protected static ?int $sort = 1;
```

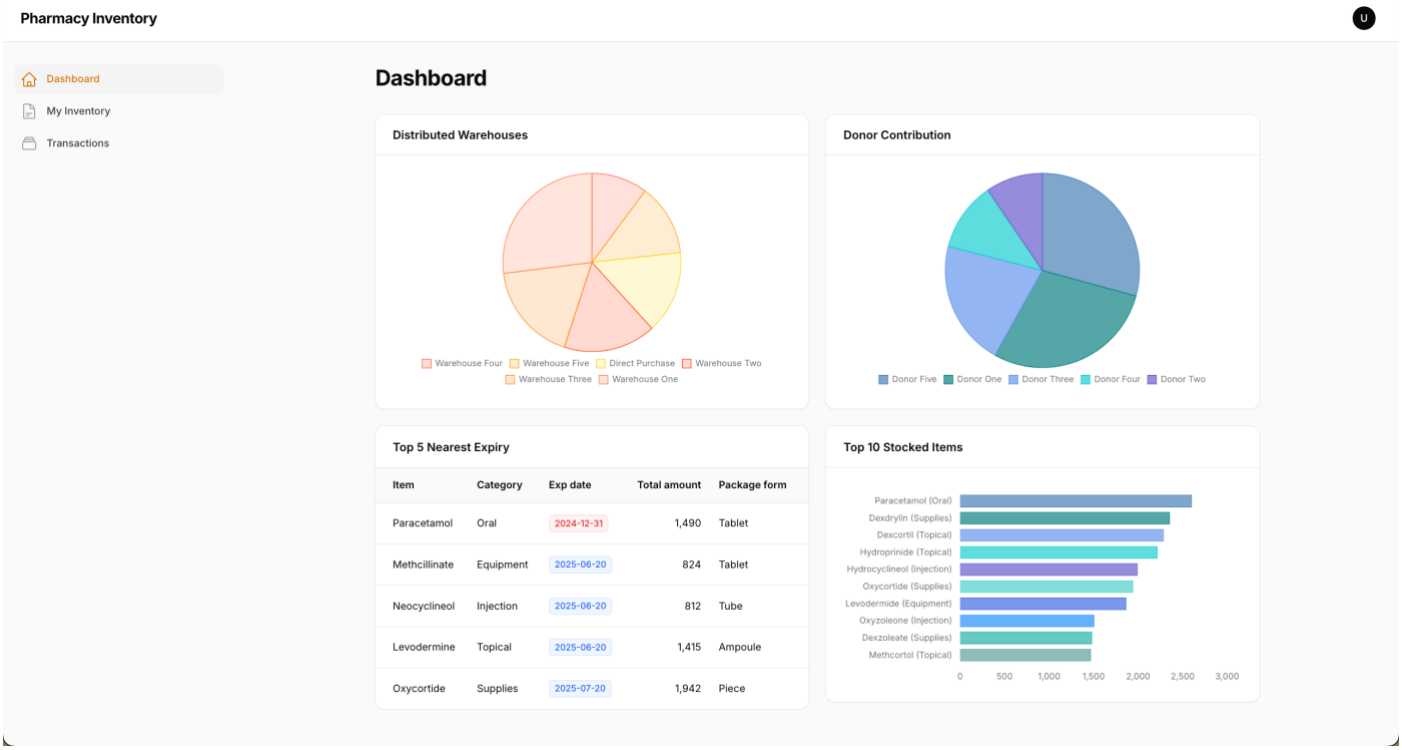
12 Conclusion

12.1 Change App Name

header မှာပေါ်နေတဲ့ **app name** က **default** အရ "**Laravel**" ဖြစ်နေတာကို **.env** မှာပြင်လို့ရပါတယ်။

```
APP_NAME=Laravel # နှစ်သက်ရာအမည်ပြောင်းရန်
```

ပေးချင်တဲ့ **app name** က **space** တွေပါမယ်ဆိုရင် **Double quote** "" ထဲမှာထည့်ပေးပါ။
.env ဖိုင်မှာပြင်ပြီး **save** လုပ်လိုက်ရင်တော့ **terminal** မှာ **npm run dev** လုပ်နေတာကော၊ **php artisan serve** နဲ့ **run** နေတာကော **Ctrl + C** နှိပ်ပြီး **stop** လုပ်ရပါတယ်။ ပြီးမှ **npm run dev** နဲ့ **php artisan serve** ကိုနောက်တစ်ခေါက်ပြန် **run** ရပါတယ်။
ဒါကတော့ **app name** လည်းပြောင်း၊ **Widget sorting** လည်းလုပ်ပြီးတဲ့အခါ တွေ့ရတဲ့ **Dashboard** ဖြစ်ပါတယ်။



လေ့ကျင့်ခန်းတွေက ဒီမှာပြီးပါပြီ။ ဒီလေ့ကျင့်ခန်းတွေဟာ **project based approach** နဲ့သွားတာမို့လို့ လက်တွေ့လုပ်လိုက်၊ သဘောတရားပြန်လေ့လာလိုက်နဲ့ ရှေ့တိုးနောက်ဆုတ်ဖြစ်ပေမယ့် သဘောတရားသီးသန့်လေ့လာတာထက် ပိုမှတ်မိလွယ်ပါတယ်။

ဒီသင်ခန်းစာတွေမှာပြုလုပ်ခဲ့တဲ့ **Pharmacy Inventory Management Application** ဟာ **production ready** ဖြစ်ဖို့အများကြီးလိုသေးပေမယ့် လေ့လာရင်းနဲ့စမ်းသပ်လိုသူများအတွက်တော့ အသုံးဝင်မယ့် **project** တစ်ခုဖြစ်ပါတယ်။

Laravel framework နဲ့ပတ်သက်ပြီးအခြေခံကျကျလေ့လာဖို့ **Laravel documentation website** ဖြစ်တဲ့ **https://laravel.com/docs** မှာဖတ်ရှုနိုင်ပါတယ်။

FilamentPHP အတွက်သေချာအချိန်ပေးပြီးလေ့လာချင်တယ်ဆိုရင် **Filament documentation https://filamentphp.com/docs** မှာဖတ်ရှုနိုင်ပါတယ်။