Building a Multi-Tenant App in Laravel with Separate Databases

Building a Multi-Tenant App in Laravel with Separate Databases

In a multi-tenant architecture with separate databases, each customer gets their own schema. This provides strong data isolation, simplifies compliance, and makes backups or migrations safer. In this tutorial, we'll build a central database for tenants, dynamically switch DB connections per request, define a tenant-aware base model, run per-tenant migrations, seed both central and tenant databases, and create an admin UI + sample controller to operate on tenant data.

1 – Central Database for Tenants

Create a central `tenants` table to store each customer's domain and DB credentials. This table lives in the default connection (usually `mysql`).

```
// database/migrations/2025_08_27_000000_create_tenants_table.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up(): void {
        Schema::create('tenants', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('domain')->unique();   // e.g.
acme.example.com
```

```
            $table->string('database')->unique(); // e.g. tenant_acme
            $table->string('db_host')->default('127.0.0.1');
            $table->unsignedInteger('db_port')->default(3306);
            $table->string('db_username');
            $table->string('db_password');
            $table->timestamps();
        });
    }
    public function down(): void {
        Schema::dropIfExists('tenants');
    }
};Code language: PHP (php)
```

Each row represents one tenant. The `domain` identifies requests, and the DB credentials allow dynamically connecting to that tenant's database. Store secrets securely in production (e.g., encrypt `db_password` or use a secret manager).

2 – Models and Base Class

Define a central `Tenant` model and a tenant-aware base model. Tenant-owned models will extend the base and automatically use the tenant connection after middleware config.

```
// app/Models/Tenant.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Tenant extends Model
{
    protected $connection = 'mysql'; // central DB
    protected $fillable = [
        'name', 'domain', 'database', 'db_host', 'db_port',
'db_username', 'db_password'
```

```
    ];
}Code language: PHP (php)
```

This model reads from the central database and exposes tenant rows we'll use at runtime to configure the per-request tenant connection.

```
// app/Models/TenantModel.php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

abstract class TenantModel extends Model
{
    protected $connection = 'tenant'; // dynamic connection, set per
request
}Code language: PHP (php)
```

Any model extending `TenantModel` will query the `tenant` connection. We'll create and reconnect that connection dynamically in middleware before controllers run.

```
// app/Models/TenantUser.php
namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;

class TenantUser extends Authenticatable
{
    protected $connection = 'tenant';
    protected $fillable = ['name','email','password'];
    protected $hidden = ['password','remember_token'];
}Code language: PHP (php)
```

This tenant-scoped user model lives in each tenant DB and supports isolated authentication per customer.

3 – Middleware to Configure the Tenant Connection

Detect the tenant by the request's host, then configure and reconnect the `tenant` connection for the lifetime of the request.

```
// app/Http/Middleware/IdentifyTenant.php
namespace App\Http\Middleware;

use App\Models\Tenant;
use Closure;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;

class IdentifyTenant
{
    public function handle($request, Closure $next)
    {
        $host = $request->getHost(); // e.g. acme.example.com
        $tenant = Tenant::where('domain', $host)->firstOrFail();

        Config::set('database.connections.tenant', [
            'driver'   => 'mysql',
            'host'     => $tenant->db_host,
            'port'     => $tenant->db_port,
            'database' => $tenant->database,
            'username' => $tenant->db_username,
            'password' => $tenant->db_password,
            'charset'  => 'utf8mb4',
            'collation'=> 'utf8mb4_unicode_ci',
        ]);

        DB::purge('tenant');
```

```
        DB::reconnect('tenant');

        return $next($request);
    }
}Code language: PHP (php)
```

This middleware looks up the tenant record, injects its DB config into runtime configuration, purges any stale connection, and reconnects. All subsequent Eloquent calls against `tenant` use the correct database.

Register it in `app/Http/Kernel.php`, then protect your tenant routes with it:

```
// app/Http/Kernel.php (snippet)
protected $routeMiddleware = [
    // ...
    'tenant' => \App\Http\Middleware\IdentifyTenant::class,
];Code language: PHP (php)
```

This makes the middleware available as `tenant` for route groups and controllers.

4 – Per-Tenant Migrations

Create a command to loop tenants and run `migrate` (or `migrate:fresh`) against the `tenant` connection after configuring it (same logic as the middleware).

```
// app/Console/Commands/TenantsMigrate.php
namespace App\Console\Commands;

use App\Models\Tenant;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\Artisan;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;
```

```
class TenantsMigrate extends Command
{
    protected $signature = 'tenants:migrate {--fresh}';
    protected $description = 'Run migrations for all tenants';

    public function handle()
    {
        foreach (Tenant::all() as $tenant) {
            $this->info("Migrating: {$tenant->name}");

            Config::set('database.connections.tenant', [
                'driver' => 'mysql',
                'host' => $tenant->db_host,
                'port' => $tenant->db_port,
                'database' => $tenant->database,
                'username' => $tenant->db_username,
                'password' => $tenant->db_password,
                'charset' => 'utf8mb4',
                'collation' => 'utf8mb4_unicode_ci',
            ]);
            DB::purge('tenant');
            DB::reconnect('tenant');

            $cmd = $this->option('fresh') ? 'migrate:fresh' :
'migrate';
            Artisan::call($cmd, ['--database' => 'tenant', '--force'
=> true]);

            $this->line(Artisan::output());
        }
    }
}Code language: PHP (php)
```

Running `php artisan tenants:migrate` applies your tenant schema to every tenant DB. Use `--fresh` to rebuild from scratch (only in safe environments).

5 – Seeders: Central Tenant + Per-Tenant Data

Seed a demo tenant in the central DB, then loop over tenants to seed each tenant DB (users, demo content, etc.).

```
// database/seeders/TenantSeeder.php  (CENTRAL DB)
namespace Database\Seeders;

use App\Models\Tenant;
use Illuminate\Database\Seeder;

class TenantSeeder extends Seeder
{
    public function run(): void
    {
        Tenant::firstOrCreate(
            ['domain' => 'acme.local'],
            [
                'name'        => 'Acme Inc',
                'database'    => 'tenant_acme',
                'db_host'     => '127.0.0.1',
                'db_port'     => 3306,
                'db_username' => 'root',
                'db_password' => '',
            ]
        );
    }
}Code language: PHP (php)
```

This creates a sample tenant row in the central database pointing to a tenant database named `tenant_acme`. Create that database manually or via automation when provisioning.

```
// database/seeders/TenantDatabaseSeeder.php  (RUNS INSIDE TENANT DB)
```

```
namespace Database\Seeders;

use App\Models\TenantUser;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\Hash;

class TenantDatabaseSeeder extends Seeder
{
    public function run(): void
    {
        TenantUser::firstOrCreate(
            ['email' => 'owner@acme.local'],
            ['name' => 'Acme Owner', 'password' =>
Hash::make('password')]
        );
    }
}Code language: PHP (php)
```

This seeder targets the tenant DB and creates a default tenant user. It assumes the connection named `tenant` is already configured before running.

```
// app/Console/Commands/TenantsSeed.php
namespace App\Console\Commands;

use App\Models\Tenant;
use Illuminate\Console\Command;
use Illuminate\Support\Facades\Artisan;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;

class TenantsSeed extends Command
{
    protected $signature = 'tenants:seed';
    protected $description = 'Run seeders for all tenants';

    public function handle()
    {
        foreach (Tenant::all() as $tenant) {
```

```
            $this->info("Seeding: {$tenant->name}");

            Config::set('database.connections.tenant', [
                'driver' => 'mysql',
                'host' => $tenant->db_host,
                'port' => $tenant->db_port,
                'database' => $tenant->database,
                'username' => $tenant->db_username,
                'password' => $tenant->db_password,
                'charset' => 'utf8mb4',
                'collation' => 'utf8mb4_unicode_ci',
            ]);
            DB::purge('tenant');
            DB::reconnect('tenant');

            // Run the tenant-specific seeder class
            Artisan::call('db:seed', [
                '--database' => 'tenant',
                '--class'    =>
\Database\Seeders\TenantDatabaseSeeder::class,
                '--force'    => true,
            ]);

            $this->line(Artisan::output());
        }
    }
}Code language: PHP (php)
```

Run `php artisan tenants:seed` after `tenants:migrate` to populate each tenant DB. If you need different data per tenant, parameterize by `$tenant` values or add factories.

6 – Sample Controller That Uses the Tenant Database

Protect tenant routes with the middleware so the `tenant` connection is live. Then query tenant models as usual.

```
// routes/web.php (snippet)
use App\Http\Controllers\TenantDashboardController;

Route::middleware('tenant')->group(function () {
    Route::get('/dashboard', [TenantDashboardController::class,
'index'])
        ->name('tenant.dashboard');
});Code language: PHP (php)
```

This route group ensures that requests pass through `IdentifyTenant` first. Inside the controller, any Eloquent query against `tenant` models will hit the tenant DB.

```
// app/Http/Controllers/TenantDashboardController.php
namespace App\Http\Controllers;

use App\Models\TenantUser;
use Illuminate\Http\Request;

class TenantDashboardController extends Controller
{
    public function index(Request $request)
    {
        $usersCount = TenantUser::count();
        $latestUsers =
TenantUser::orderBy('id','desc')->limit(5)->get();

        return view('tenant.dashboard',
compact('usersCount','latestUsers'));
    }
}Code language: PHP (php)
```

This controller reads from the tenant database via `TenantUser`. The middleware made sure the `tenant` connection was configured before these queries run.

```
<!-- resources/views/tenant/dashboard.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1 class="mb-4">Tenant Dashboard</h1>

  <div class="card mb-3">
    <div class="card-body">
      <h5 class="card-title">Users</h5>
      <p class="card-text mb-0">Total Users: {{ $usersCount }}</p>
    </div>
  </div>

  <h5 class="mt-4 mb-3">Latest Users</h5>
  @foreach($latestUsers as $user)
    <div class="card mb-2">
      <div class="card-body d-flex justify-content-between">
        <span>{{ $user->name }} ({{ $user->email }})</span>
        <a href="#" class="btn btn-sm btn-theme r-04">View</a>
      </div>
    </div>
  @endforeach
</div>
@endsectionCode language: PHP (php)
```

The Blade UI shows basic stats and a recent users list from the tenant DB. You can expand this with charts, filters, or pagination as needed.

7 – Admin UI for Provisioning Tenants (Central)

Provide a minimal central admin to create tenants. After creating a row, create the DB, run tenant migrations, and optionally seed it.

```
// app/Http/Controllers/TenantAdminController.php
namespace App\Http\Controllers;

use App\Models\Tenant;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Artisan;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;

class TenantAdminController extends Controller
{
    public function index()
    {
        return view('tenants.index', ['tenants' => Tenant::all()]);
    }

    public function store(Request $request)
    {
        $tenant = Tenant::create($request->validate([
            'name'        => 'required|string|max:100',
            'domain'      =>
'required|string|max:191|unique:tenants,domain',
            'database'    =>
'required|string|max:191|unique:tenants,database',
            'db_host'     => 'required|string',
            'db_port'     => 'required|integer',
            'db_username' => 'required|string',
            'db_password' => 'nullable|string',
        ]));

        DB::statement("CREATE DATABASE `{$tenant->database}`");

        Config::set('database.connections.tenant', [
            'driver'   => 'mysql',
            'host'     => $tenant->db_host,
            'port'     => $tenant->db_port,
            'database' => $tenant->database,
            'username' => $tenant->db_username,
```

```
            'password' => $tenant->db_password,
            'charset'  => 'utf8mb4',
            'collation'=> 'utf8mb4_unicode_ci',
        ]);
        DB::purge('tenant');
        DB::reconnect('tenant');

        Artisan::call('migrate', ['--database' => 'tenant', '--force'
=> true]);
        Artisan::call('db:seed', [
            '--database' => 'tenant',
            '--class'    =>
\Database\Seeders\TenantDatabaseSeeder::class,
            '--force'    => true,
        ]);

        return redirect()->back()->with('status','Tenant created and
initialized');
    }
}Code language: PHP (php)
```

On submission, this creates the tenant DB, prepares the connection, migrates it, and seeds an initial user. Add authorization controls so only super-admins can access this endpoint.

```
<!-- resources/views/tenants/index.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1 class="mb-4">Tenants</h1>

  @if(session('status'))
    <div class="alert alert-success">{{ session('status') }}</div>
  @endif

  @foreach($tenants as $tenant)
    <div class="card mb-2">
      <div class="card-body">
```

```
        <strong>{{ $tenant->name }}</strong> — {{ $tenant->domain }}
      </div>
    </div>
  @endforeach

  <h5 class="mt-4">Create Tenant</h5>
  <form method="POST" action="{{ route('tenants.store') }}"
class="mt-2">
    @csrf
    <div class="row g-2">
      <div class="col-md-4"><input name="name" class="form-control"
placeholder="Tenant Name" required/></div>
      <div class="col-md-4"><input name="domain" class="form-control"
placeholder="Domain (e.g. acme.local)" required/></div>
      <div class="col-md-4"><input name="database" class="form-
control" placeholder="DB Name (e.g. tenant_acme)" required/></div>
      <div class="col-md-3"><input name="db_host" class="form-control"
value="127.0.0.1" required/></div>
      <div class="col-md-3"><input name="db_port" class="form-control"
value="3306" required/>></div>
      <div class="col-md-3"><input name="db_username" class="form-
control" value="root" required/></div>
      <div class="col-md-3"><input name="db_password" type="password"
class="form-control" placeholder="DB Password"/></div>
    </div>
    <button class="btn btn-theme mt-3">Create and Initialize</button>
  </form>
</div>
@endsectionCode language: PHP (php)
```

This minimal UI lists existing tenants and provides a form that provisions a new tenant end-to-end. In production, apply authorization, rate limiting, and detailed error handling.

Wrapping Up

You built a multi-tenant system with strict data isolation by using a central database for tenant metadata and a separate database per tenant. You configured the tenant connection at request-time via middleware, created a base model to route queries to the tenant DB, added migration and seeding commands for all tenants, and implemented an admin UI that provisions, migrates, and seeds new tenant databases automatically. This pattern scales well for compliance and backups, and provides clear boundaries between customer data sets.

What's Next

- Using JSON Columns in Eloquent (Practical Guide)
- How to Use Eloquent API Resources for Clean APIs
- Soft Deletes: Restore, Force Delete, and Prune Data