Implementing Database Multi-tenancy with Separate Databases per Tenant

This solution provides a comprehensive approach to multi-tenancy where each tenant has their own dedicated database, ensuring complete data isolation.

1. Setup Tenant Identification

First, establish how to identify tenants (typically via subdomain or request header):

```
// app/Providers/TenancyServiceProvider.php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Config;

class TenancyServiceProvider extends ServiceProvider
{
    public function register()
    {
        // Detect tenant from subdomain
        $host = request()->getHost();
        $subdomain = explode('.', $host)[0];

        // Or from header (for API requests)
        $tenantId = request()->header('X-Tenant-ID') ?? $subdomain;

        // Store tenant identifier
        Config::set('tenant.id', $tenantId);
    }

    public function boot()
    {
        // Register middleware to handle tenant switching
        $this->app['router']->aliasMiddleware('tenant',
\App\Http\Middleware\SetTenantDatabase::class);
    }
}
```

PROF

Don't forget to register this provider in `config/app.php`.

2. Create Tenant Database Middleware

```php
// app/Http/Middleware/SetTenantDatabase.php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Config;
use Illuminate\Support\Facades\DB;

class SetTenantDatabase
{
    public function handle($request, Closure $next)
    {
        $tenantId = Config::get('tenant.id');

        if (!$tenantId) {
            abort(403, 'Tenant not identified');
        }

        // Verify tenant exists and get database config
        $tenant = \App\Models\Tenant::where('identifier', $tenantId)-
>firstOrFail();

        // Configure the tenant's database
        Config::set('database.connections.tenant', [
            'driver' => 'mysql',
            'host' => $tenant->db_host,
            'port' => $tenant->db_port,
            'database' => $tenant->db_name,
            'username' => $tenant->db_username,
            'password' => $tenant->db_password,
            'charset' => 'utf8mb4',
            'collation' => 'utf8mb4_unicode_ci',
            'prefix' => '',
            'prefix_indexes' => true,
            'strict' => true,
            'engine' => null,
        ]);

        // Set as default connection
        DB::purge('tenant');
        Config::set('database.default', 'tenant');
        DB::reconnect('tenant');

        return $next($request);
    }
}
```

PROF

3. Tenant Model and Migration

Create a model and migration for storing tenant information:

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

```
// database/migrations/xxxx_create_tenants_table.php

public function up()
{
    Schema::create('tenants', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('identifier')->unique(); // subdomain or tenant
ID
        $table->string('db_host');
        $table->string('db_name');
        $table->string('db_username');
        $table->string('db_password');
        $table->integer('db_port')->default(3306);
        $table->json('meta')->nullable();
        $table->timestamps();
    });
}
```

4. Central Database Configuration

Configure your central database (for tenant records) in `.env`:

```
DB_CONNECTION=central
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=central_db
DB_USERNAME=root
DB_PASSWORD=
```

PROF

Add the connection to `config/database.php`:

```
'connections' => [
    'central' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', '127.0.0.1'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'central_db'),
        'username' => env('DB_USERNAME', 'root'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8mb4',
        'collation' => 'utf8mb4_unicode_ci',
        'prefix' => '',
        'prefix_indexes' => true,
        'strict' => true,
        'engine' => null,
    ],
    // ... other connections
],
```

5. Tenant Database Creation Command

Create an Artisan command to provision new tenant databases:

```
php artisan make:command CreateTenant
```

```
// app/Console/Commands/CreateTenant.php

namespace App\Console\Commands;

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

class CreateTenant extends Command
{
    protected $signature = 'tenant:create {name} {--subdomain=}';
    protected $description = 'Create a new tenant with dedicated
database';

    public function handle()
    {
        $name = $this->argument('name');
        $subdomain = $this->option('subdomain') ?? Str::slug($name);

        // Create database
        $dbName = 'tenant_' . Str::random(8);
        $dbUsername = 'tenant_' . Str::random(8);
        $dbPassword = Str::random(16);

        // Create database and user (MySQL example)
        DB::connection('central')->statement("CREATE DATABASE
{$dbName}");
        DB::connection('central')->statement("CREATE USER
'{$dbUsername}'@'%' IDENTIFIED BY '{$dbPassword}'");
        DB::connection('central')->statement("GRANT ALL PRIVILEGES ON
{$dbName}.* TO '{$dbUsername}'@'%'");
        DB::connection('central')->statement("FLUSH PRIVILEGES");

        // Create tenant record
```

PROF

```
        $tenant = Tenant::create([
            'name' => $name,
            'identifier' => $subdomain,
            'db_host' => env('DB_HOST', '127.0.0.1'),
            'db_name' => $dbName,
            'db_username' => $dbUsername,
            'db_password' => $dbPassword,
            'db_port' => env('DB_PORT', 3306),
        ]);

        // Run migrations for the new tenant
        $this->migrateTenantDatabase($tenant);

        $this->info("Tenant {$name} created successfully!");
        $this->info("Subdomain: {$subdomain}");
        $this->info("Database: {$dbName}");
    }

    protected function migrateTenantDatabase(Tenant $tenant)
    {
        config([
            'database.connections.tenant.database' => $tenant->db_name,
            'database.connections.tenant.username' => $tenant-
>db_username,
            'database.connections.tenant.password' => $tenant-
>db_password,
        ]);

        DB::purge('tenant');

        $this->call('migrate', [
            '--database' => 'tenant',
            '--path' => 'database/migrations/tenant',
            '--force' => true,
        ]);
    }
}
```

PROF

6. Tenant-Specific Migrations

Create tenant-specific migrations in a separate directory:

```
mkdir database/migrations/tenant
```

Example tenant migration:

```
// database/migrations/tenant/xxxx_create_tenant_tables.php
```

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

    // Add other tenant-specific tables
}
```

7. Tenant Model Trait

Create a trait to make models tenant-aware:

```
// app/Traits/TenantScoped.php

namespace App\Traits;

trait TenantScoped
{
    public static function bootTenantScoped()
    {
        static::addGlobalScope(new TenantScope);
    }
}
```

And the scope:

PROF

```
// app/Scopes/TenantScope.php

namespace App\Scopes;

use Illuminate\Database\Eloquent\Builder;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Scope;

class TenantScope implements Scope
{
    public function apply(Builder $builder, Model $model)
    {
        // No need for tenant_id column since we're using separate
databases
        // This is just here for consistency with other multi-tenancy
approaches
    }
}
```

8. Route Middleware

Apply tenant middleware to routes:

```
// routes/web.php

Route::middleware(['tenant'])->group(function () {
    // All tenant-specific routes
    Route::get('/', [TenantController::class, 'dashboard']);

    // Other tenant routes...
});
```

9. Tenant Switch Controller

For admin users who need to switch between tenants:

```
// app/Http/Controllers/TenantSwitchController.php

namespace App\Http\Controllers;

use App\Models\Tenant;
use Illuminate\Support\Facades\Session;

class TenantSwitchController extends Controller
{
    public function switch(Tenant $tenant)
    {
        if (!auth()->user()->isSuperAdmin()) {
            abort(403);
        }

        Session::put('tenant_id', $tenant->identifier);

        return redirect()->to('http://' . $tenant->identifier . '.' .
config('app.domain'));
    }
}
```

PROF

10. Testing the Implementation

Create tests for your multi-tenancy setup:

```php
// tests/Feature/TenancyTest.php

namespace Tests\Feature;

use App\Models\Tenant;
use Illuminate\Foundation\Testing\RefreshDatabase;
use Tests\TestCase;

class TenancyTest extends TestCase
{
    use RefreshDatabase;

    protected $tenant;

    protected function setUp(): void
    {
        parent::setUp();

        $this->tenant = Tenant::create([
            'name' => 'Test Tenant',
            'identifier' => 'test',
            'db_host' => env('DB_HOST', '127.0.0.1'),
            'db_name' => 'tenant_test',
            'db_username' => 'tenant_test',
            'db_password' => 'password',
            'db_port' => env('DB_PORT', 3306),
        ]);

        // Setup tenant database (in memory for testing)
        config([
            'database.connections.tenant.database' => ':memory:',
            'database.default' => 'tenant',
        ]);

        $this->artisan('migrate', ['--database' => 'tenant']);
    }

    public function test_tenant_database_isolation()
    {
        // Create a record in tenant database
        \App\Models\User::create(['name' => 'Tenant User', 'email' =>
'user@tenant.com', 'password' => 'secret']);

        // Verify it exists in tenant DB
        $this->assertDatabaseHas('users', ['email' =>
'user@tenant.com']);

        // Switch to central DB
        config(['database.default' => 'central']);

        // Verify it doesn't exist in central DB
        $this->assertDatabaseMissing('users', ['email' =>
```

PROF

```
'user@tenant.com']);
    }
}
```

11. Deployment Considerations

1. **Database Hosting**: Consider using cloud database services that support easy provisioning
2. **Backups**: Implement a backup strategy for all tenant databases
3. **Migrations**: Create a system to apply migrations across all tenant databases
4. **Monitoring**: Set up monitoring for all databases
5. **Connection Pooling**: Configure connection pooling for better performance

12. Scaling Considerations

1. **Database Sharding**: Distribute tenant databases across multiple servers
2. **Read Replicas**: Implement read replicas for high-traffic tenants
3. **Caching**: Use Redis or Memcached with tenant-specific prefixes
4. **Queue Workers**: Configure separate queue workers per tenant if needed

This implementation provides complete database isolation for each tenant while maintaining a manageable central system for tenant administration. The solution is scalable and can be adapted to various hosting environments.