ChalaN – Laravel Invoice Management System

Developer Guide

Version: 1.0

Author: RokByte Technologies

Support: rokbyte@gmail.com

Introduction:

Welcome to the **ChalaN – Laravel Invoice Management System (Developer Documentation)**.

This guide is intended for **developers and technical contributors** who wish to customize, extend, or integrate the ChalaN system. It explains the internal logic, architecture, and Laravel MVC structure of the application.

It covers:

- Application folder and file structure
- MVC architecture and relationships
- Module-wise functionality
- Configuration and customization guidelines
- Developer best practices

Version: 1.0.0
Developed by: RokByte Technologies
Last updated: November 2025

System Overview:

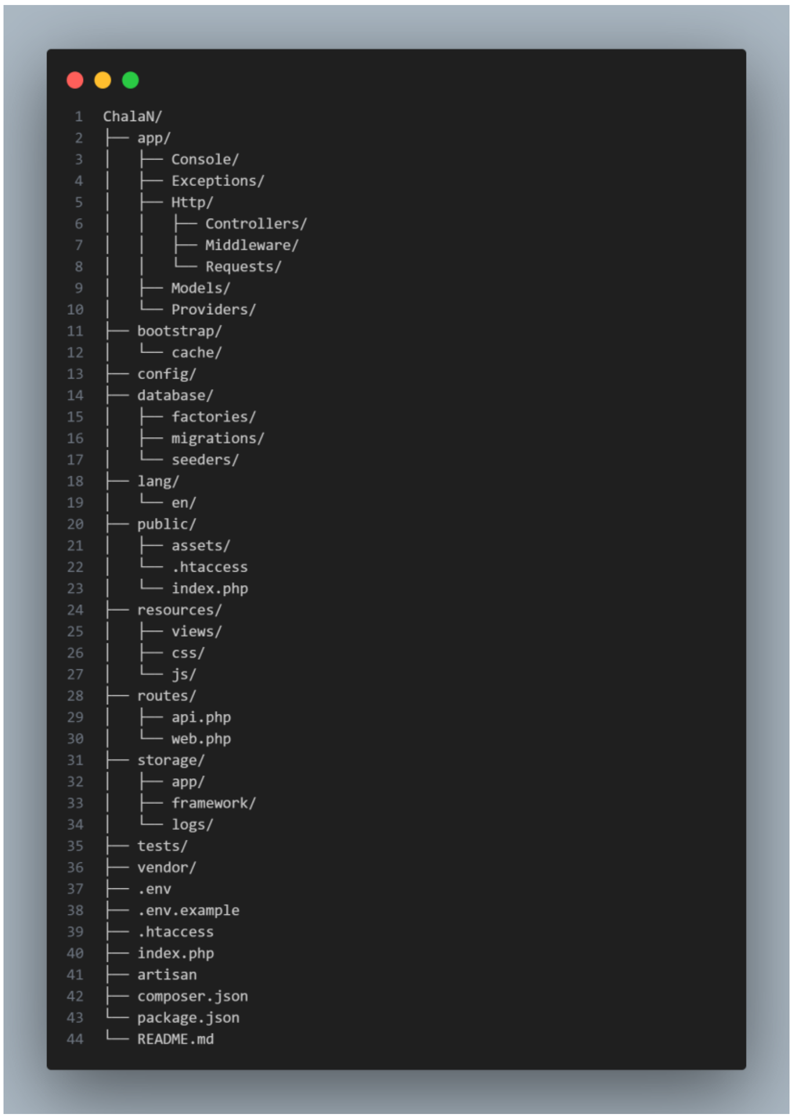
ChalaN is built using the **Laravel 12.x** framework, which follows the **MVC (Model–View–Controller)** pattern.
This architecture separates the logic, presentation, and data handling for better scalability and maintainability.

Key Technologies Used

- **Backend Framework:** Laravel 12.x
- **Frontend:** Blade Templates, Bootstrap 5, jQuery
- **Database:** MySQL
- **Language Support:** Multi-language via Database (Custom Phrase Helper)
- **Payment Integration:** Configurable gateways
- **Email Service:** Laravel Mail (SMTP)
- **Charts:** Chart.js

Folder Structure (MVC Overview):

Below is the general structure of the Laravel Project:



```
ChalaN/
├── app/
│   ├── Console/
│   ├── Exceptions/
│   ├── Http/
│   │   ├── Controllers/
│   │   ├── Middleware/
│   │   └── Requests/
│   ├── Models/
│   └── Providers/
├── bootstrap/
│   └── cache/
├── config/
├── database/
│   ├── factories/
│   ├── migrations/
│   └── seeders/
├── lang/
│   └── en/
├── public/
│   ├── assets/
│   └── .htaccess
│   └── index.php
├── resources/
│   ├── views/
│   ├── css/
│   └── js/
├── routes/
│   ├── api.php
│   └── web.php
├── storage/
│   ├── app/
│   ├── framework/
│   └── logs/
├── tests/
├── vendor/
├── .env
├── .env.example
├── .htaccess
├── index.php
├── artisan
├── composer.json
└── package.json
└── README.md
```

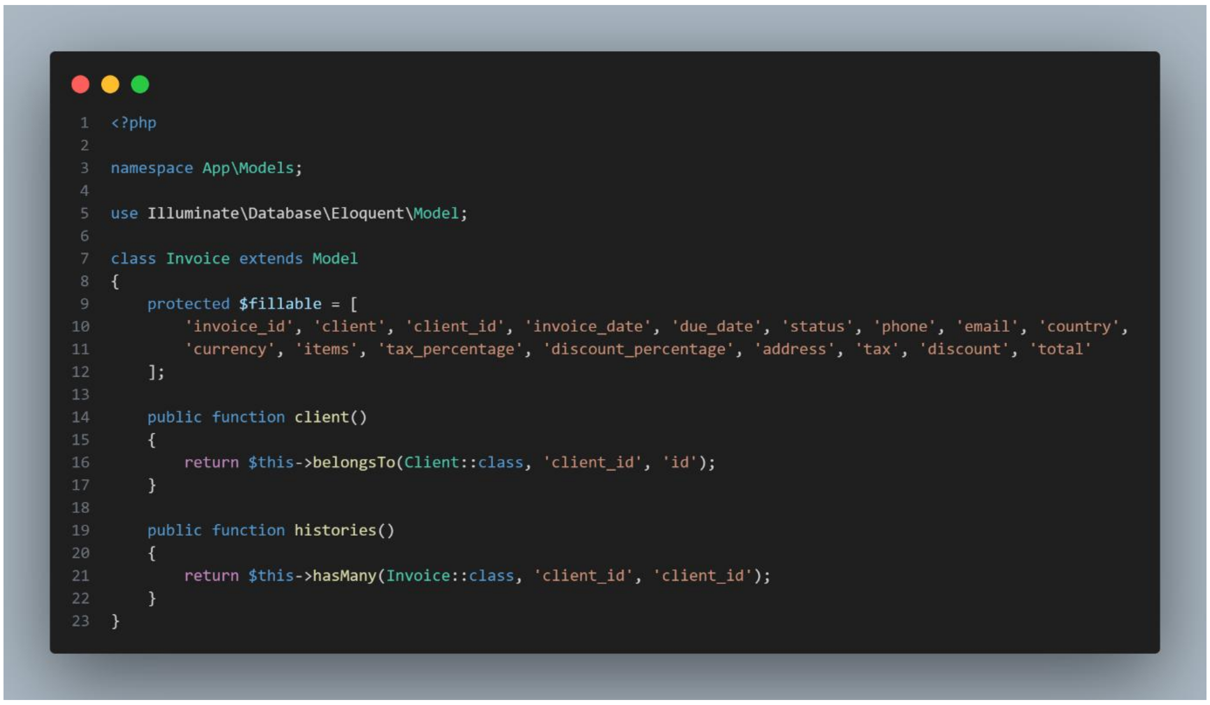
MVC Architecture Overview:

The **ChalaN – Laravel Invoice Management System** is built on the **Laravel MVC (Model–View–Controller)** architecture, ensuring a clean separation of logic, presentation, and data management.

Model:

- Located in `/app/Models/`
- Handles database interaction using **Eloquent ORM**.
- Each table has a dedicated model (e.g., `Invoice.php`, `Client.php`, `Service.php`).
- Defines **relationships** (hasMany, belongsTo, etc.) and **fillable** attributes.

Example: `Invoice.php`

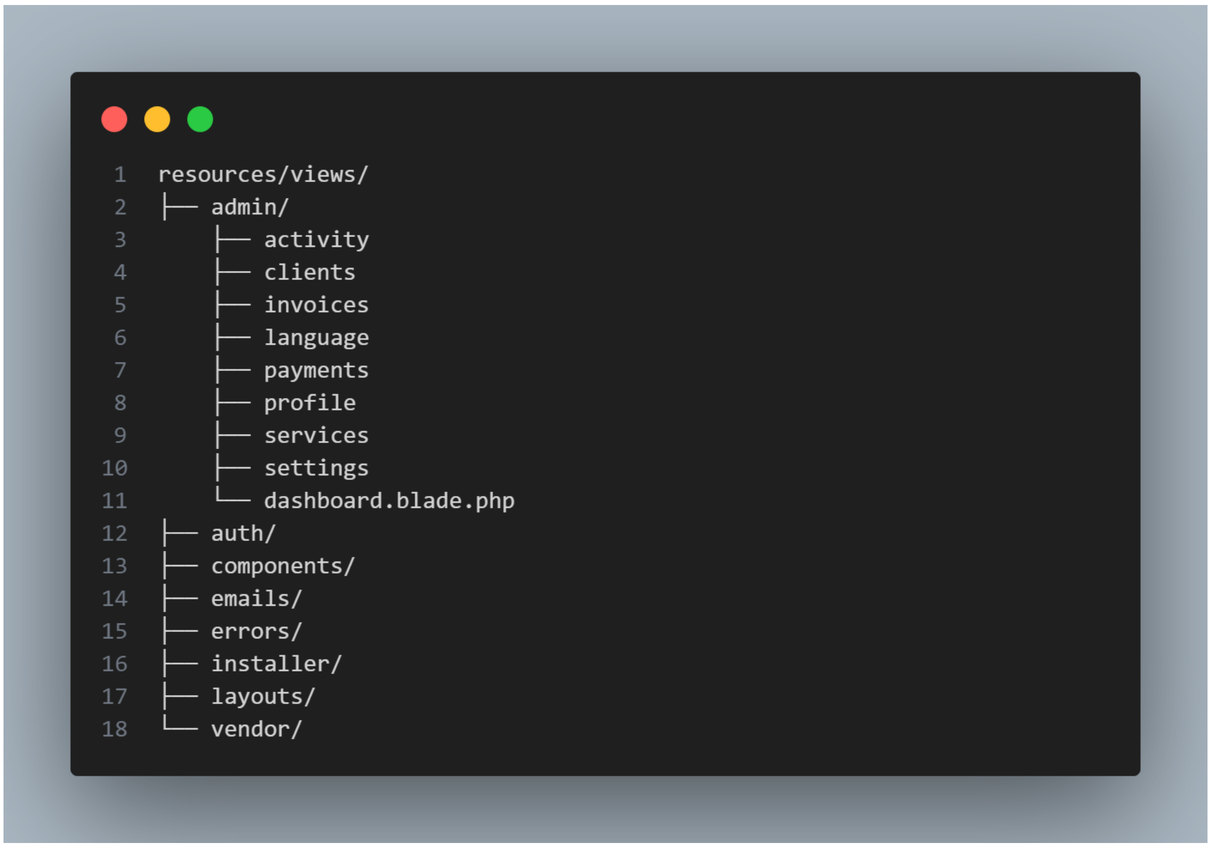


```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Invoice extends Model
{
    protected $fillable = [
        'invoice_id', 'client', 'client_id', 'invoice_date', 'due_date', 'status', 'phone', 'email', 'country',
        'currency', 'items', 'tax_percentage', 'discount_percentage', 'address', 'tax', 'discount', 'total'
    ];

    public function client()
    {
        return $this->belongsTo(Client::class, 'client_id', 'id');
    }

    public function histories()
    {
        return $this->hasMany(Invoice::class, 'client_id', 'client_id');
    }
}
```

View:

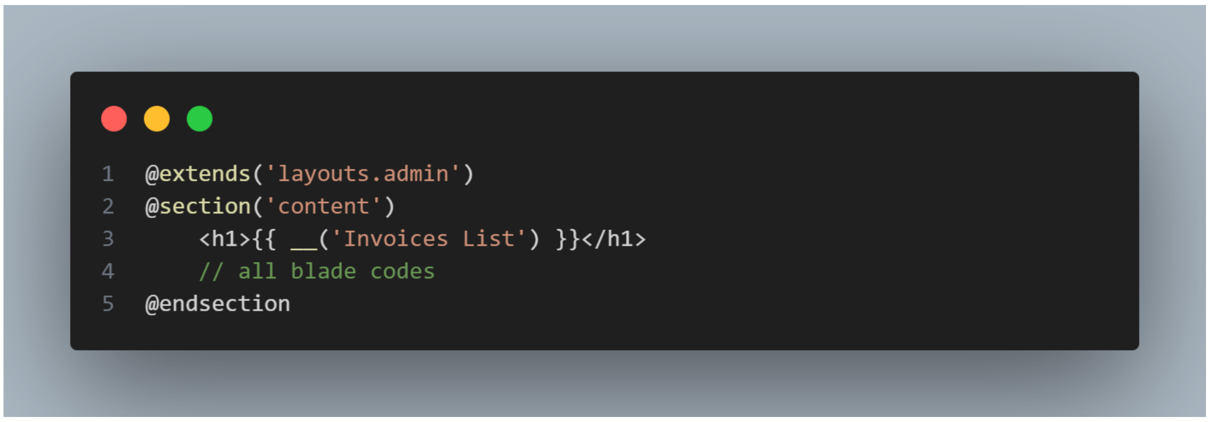
- Located in `/resources/views/`
- Contains all **Blade templates** for the frontend.

- Organized by modules for easy maintenance:



```
resources/views/
├── admin/
    ├── activity
    ├── clients
    ├── invoices
    ├── language
    ├── payments
    ├── profile
    ├── services
    ├── settings
    └── dashboard.blade.php
├── auth/
├── components/
├── emails/
├── errors/
├── installer/
├── layouts/
└── vendor/
```

- Uses Blade syntax (`@extends`, `@section`, `@yield`) for layout inheritance.



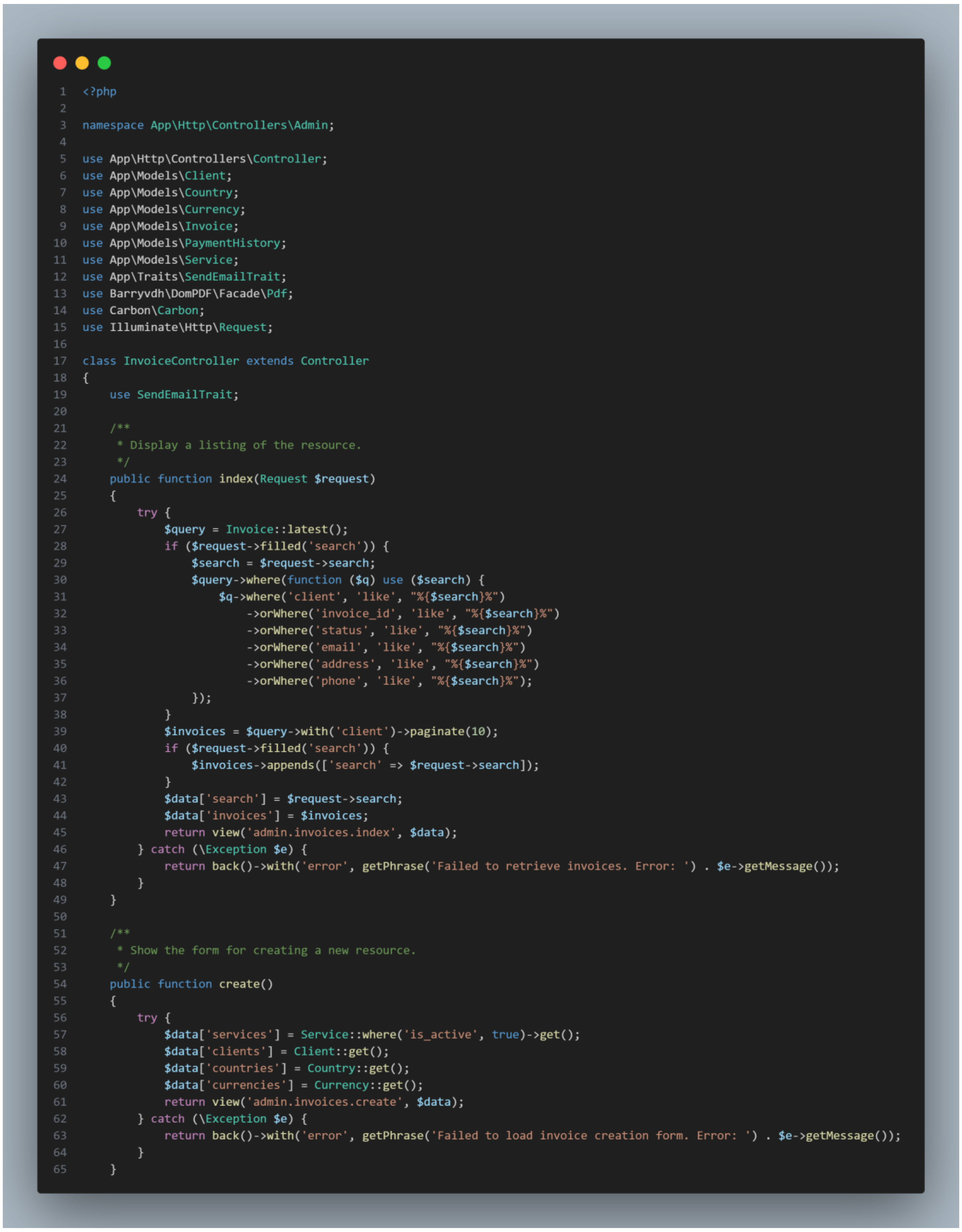
```
@extends('layouts.admin')
@section('content')
    <h1>{{ __('Invoices List') }}</h1>
    // all blade codes
@endsection
```

Controller:

- Located in `/app/Http/Controllers/`
- Contains business logic and interacts with Models and Views.

- Follows naming convention: `ModuleNameController`.

Example: `InvoiceController.php`



```php
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Models\Client;
use App\Models\Country;
use App\Models\Currency;
use App\Models\Invoice;
use App\Models\PaymentHistory;
use App\Models\Service;
use App\Traits\SendEmailTrait;
use Barryvdh\DomPDF\Facade\Pdf;
use Carbon\Carbon;
use Illuminate\Http\Request;

class InvoiceController extends Controller
{
    use SendEmailTrait;

    /**
     * Display a listing of the resource.
     */
    public function index(Request $request)
    {
        try {
            $query = Invoice::latest();
            if ($request->filled('search')) {
                $search = $request->search;
                $query->where(function ($q) use ($search) {
                    $q->where('client', 'like', "%{$search}%")
                        ->orWhere('invoice_id', 'like', "%{$search}%")
                        ->orWhere('status', 'like', "%{$search}%")
                        ->orWhere('email', 'like', "%{$search}%")
                        ->orWhere('address', 'like', "%{$search}%")
                        ->orWhere('phone', 'like', "%{$search}%");
                });
            }
            $invoices = $query->with('client')->paginate(10);
            if ($request->filled('search')) {
                $invoices->appends(['search' => $request->search]);
            }
            $data['search'] = $request->search;
            $data['invoices'] = $invoices;
            return view('admin.invoices.index', $data);
        } catch (\Exception $e) {
            return back()->with('error', getPhrase('Failed to retrieve invoices. Error: ') . $e->getMessage());
        }
    }

    /**
     * Show the form for creating a new resource.
     */
    public function create()
    {
        try {
            $data['services'] = Service::where('is_active', true)->get();
            $data['clients'] = Client::get();
            $data['countries'] = Country::get();
            $data['currencies'] = Currency::get();
            return view('admin.invoices.create', $data);
        } catch (\Exception $e) {
            return back()->with('error', getPhrase('Failed to load invoice creation form. Error: ') . $e->getMessage());
        }
    }
```

RESTful Methods:

Controllers adhere to Laravel's standard RESTful structure:

- `index()` – Displays a **list of all records** (e.g., invoices).
- `create()` – Shows the **form to create a new record**.
- `store(Request $request)` – Handles **saving new records** to the database.
- `edit($id)` – Shows the **form to edit an existing record**.
- `update(Request $request, $id)` – Handles **updating existing records** in the database.
- `destroy($id)` – Handles **deleting records** safely.

Key Application Modules:

Dashboard

- Provides summary of total revenue, clients, invoices, and activities.
- Uses AJAX and Chart.js to render revenue graphs.
- Data is fetched from multiple models via `DashboardController`.

Clients

- **Model:** `Client`
- **Controller:** `ClientController`
- **Views:** `/resources/views/admin/clients/`
- **Relations:**
  - `Client` hasMany `Invoice`
- Supports CRUD operations for managing client profiles.

Services

- **Model:** `Service`
- **Controller:** `ServiceController`
- **Views:** `/resources/views/admin/services/`
- **Fields**: `name, sku, unit_price, is_active, description`.

Invoices

- **Model:** `Invoice`
- **Controller:** `InvoiceController`
- **Views:** `/resources/views/admin/invoices`
- **Relations:**
  - `Invoice` belongsTo `Client`
  - `Invoice` hasMany `Payment`
- **Fields**: `invoice_id, client, client_id, invoice_date, due_date, status, phone, email, country, currency, items, tax_percentage, discount_percentage, address, tax, discount, total.`

Payments

- **Model:** `Payment`
- **Controller:** `PaymentController`
- **Relations:**
  - `Payment` belongsTo `Invoice`
- Allows configuring gateways, tracking transactions, and payment history.

Settings

- **Controllers:**
  - `SettingController`
  - `LanguageController`
- All settings are stored in the database and cached for performance.

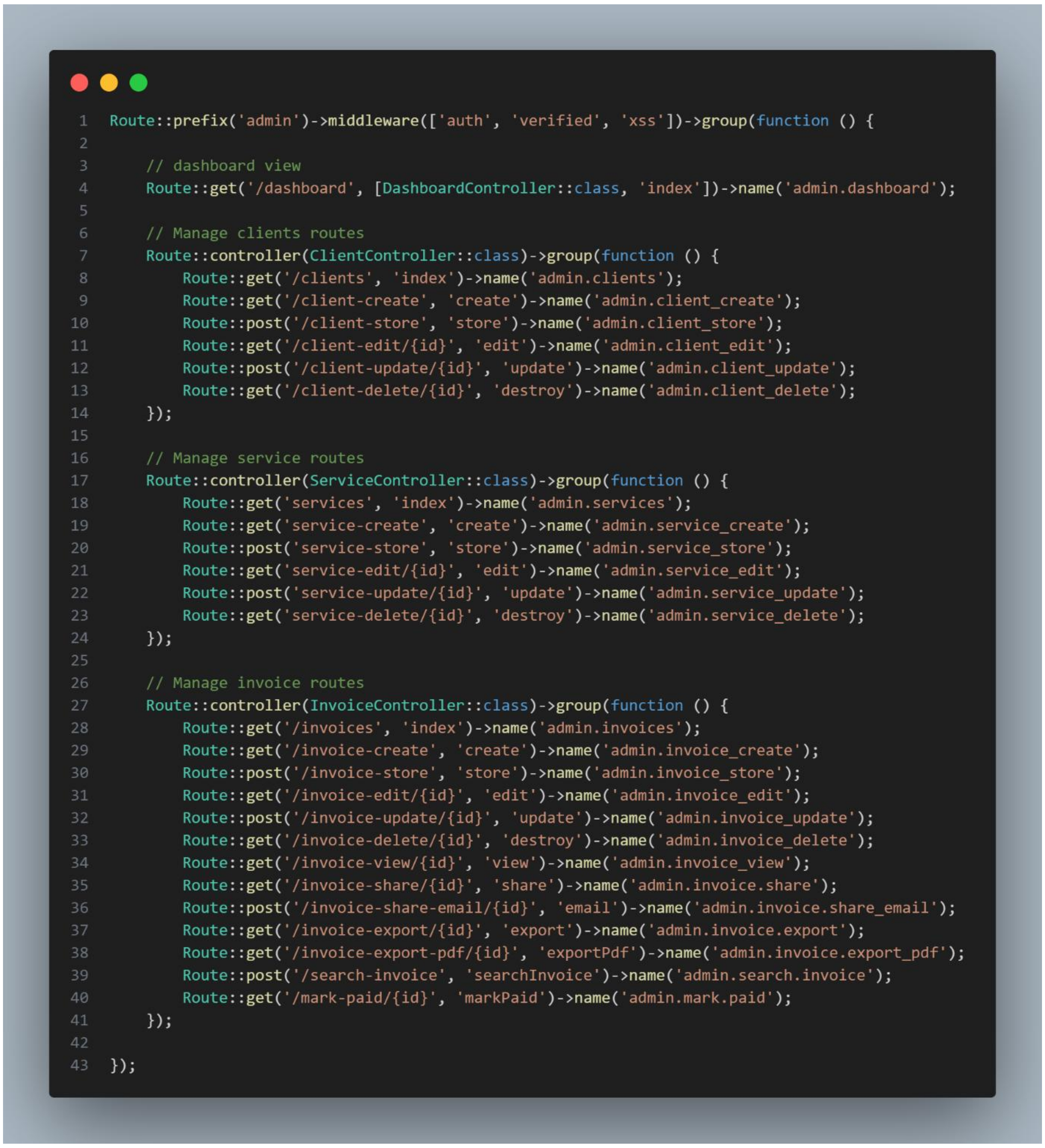
Language Management

- Languages are stored in database on language table
- You can add new languages by creating new language and translate phrase.

Routes

All routes are defined inside `/routes/web.php`

Example (`web.php`):



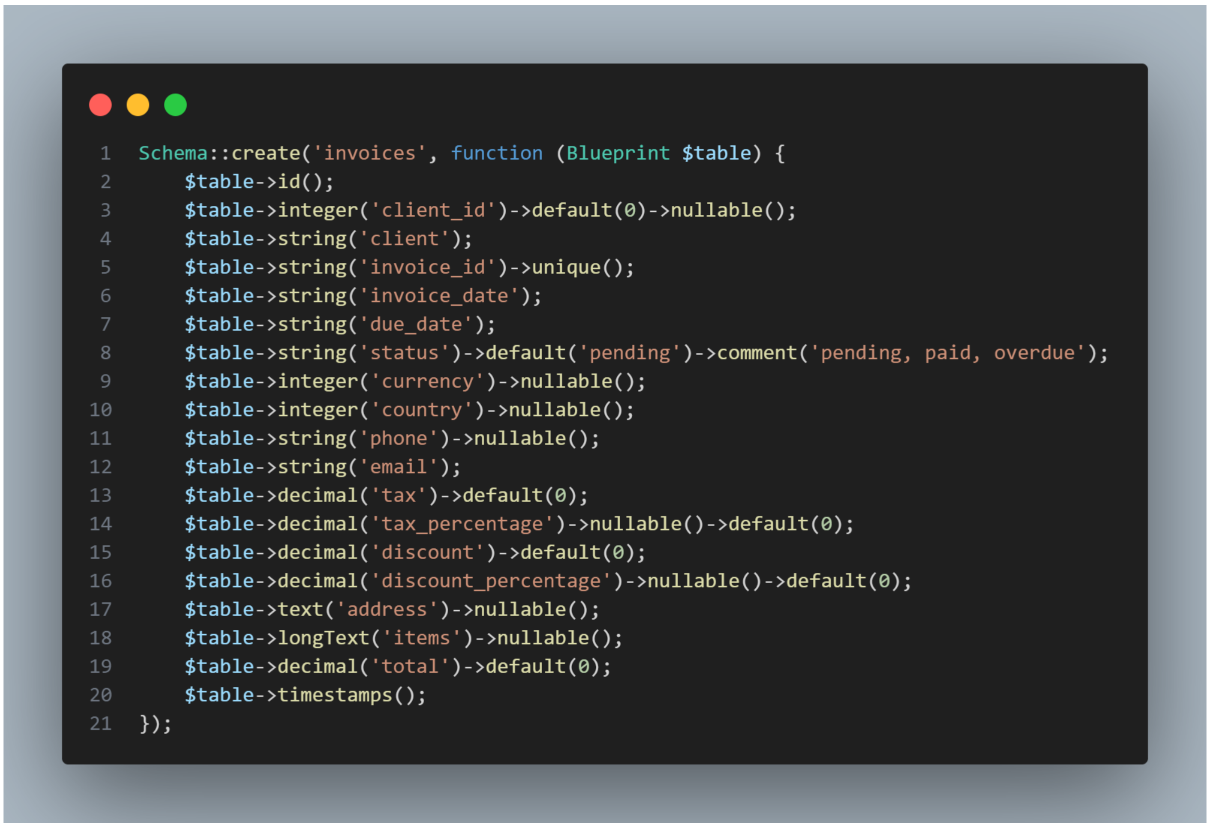
```php
Route::prefix('admin')->middleware(['auth', 'verified', 'xss'])->group(function () {

    // dashboard view
    Route::get('/dashboard', [DashboardController::class, 'index'])->name('admin.dashboard');

    // Manage clients routes
    Route::controller(ClientController::class)->group(function () {
        Route::get('/clients', 'index')->name('admin.clients');
        Route::get('/client-create', 'create')->name('admin.client_create');
        Route::post('/client-store', 'store')->name('admin.client_store');
        Route::get('/client-edit/{id}', 'edit')->name('admin.client_edit');
        Route::post('/client-update/{id}', 'update')->name('admin.client_update');
        Route::get('/client-delete/{id}', 'destroy')->name('admin.client_delete');
    });

    // Manage service routes
    Route::controller(ServiceController::class)->group(function () {
        Route::get('services', 'index')->name('admin.services');
        Route::get('service-create', 'create')->name('admin.service_create');
        Route::post('service-store', 'store')->name('admin.service_store');
        Route::get('service-edit/{id}', 'edit')->name('admin.service_edit');
        Route::post('service-update/{id}', 'update')->name('admin.service_update');
        Route::get('service-delete/{id}', 'destroy')->name('admin.service_delete');
    });

    // Manage invoice routes
    Route::controller(InvoiceController::class)->group(function () {
        Route::get('/invoices', 'index')->name('admin.invoices');
        Route::get('/invoice-create', 'create')->name('admin.invoice_create');
        Route::post('/invoice-store', 'store')->name('admin.invoice_store');
        Route::get('/invoice-edit/{id}', 'edit')->name('admin.invoice_edit');
        Route::post('/invoice-update/{id}', 'update')->name('admin.invoice_update');
        Route::get('/invoice-delete/{id}', 'destroy')->name('admin.invoice_delete');
        Route::get('/invoice-view/{id}', 'view')->name('admin.invoice_view');
        Route::get('/invoice-share/{id}', 'share')->name('admin.invoice.share');
        Route::post('/invoice-share-email/{id}', 'email')->name('admin.invoice.share_email');
        Route::get('/invoice-export/{id}', 'export')->name('admin.invoice.export');
        Route::get('/invoice-export-pdf/{id}', 'exportPdf')->name('admin.invoice.export_pdf');
        Route::post('/search-invoice', 'searchInvoice')->name('admin.search.invoice');
        Route::get('/mark-paid/{id}', 'markPaid')->name('admin.mark.paid');
    });

});
```

Database & Migrations:

- All database tables are defined via migrations in `/database/migrations/`.
- Use `php artisan migrate` to create tables.

- Seed demo data: `php artisan db:seed DatabaseSeeder`

Example Migration:



```
Schema::create('invoices', function (Blueprint $table) {
    $table->id();
    $table->integer('client_id')->default(0)->nullable();
    $table->string('client');
    $table->string('invoice_id')->unique();
    $table->string('invoice_date');
    $table->string('due_date');
    $table->string('status')->default('pending')->comment('pending, paid, overdue');
    $table->integer('currency')->nullable();
    $table->integer('country')->nullable();
    $table->string('phone')->nullable();
    $table->string('email');
    $table->decimal('tax')->default(0);
    $table->decimal('tax_percentage')->nullable()->default(0);
    $table->decimal('discount')->default(0);
    $table->decimal('discount_percentage')->nullable()->default(0);
    $table->text('address')->nullable();
    $table->longText('items')->nullable();
    $table->decimal('total')->default(0);
    $table->timestamps();
});
```

Configuration:

Main configuration files:

- `.env` – Environment settings
- `/config/app.php` – Application info
- `/config/mail.php` – SMTP settings
- `/config/services.php` – Third-party integrations

Customization Guidelines:

Add a New Module

1. Create a new **Model**, **Controller**, and **Blade view**.
2. Add routes in `web.php`.
3. Optionally, create a migration for database structure.

Security & Best Practices:

- Always validate input using `Request` classes.
- Use CSRF protection (`@csrf`) in forms.
- Never expose sensitive data in `.env`.
- Regularly update dependencies (`composer update`).
- Limit file upload types and size.

Troubleshooting

Issue	Solution
500 Error	Check file permissions and `.env` configuration
Database not found	Verify DB name and credentials in `.env`
Storage not linked	Run `php artisan storage:link`
Cache issue	Run `php artisan optimize:clear`

Model, Controller & Migration Commands

Command	Description
`php artisan make:model Invoice`	Creates a new model (e.g., `Invoice.php`) in `app/Models/`.
`php artisan make:model Invoice -m`	Creates a model **and** its corresponding migration file.
`php artisan make:controller InvoiceController`	Creates a new controller for managing logic.

Command	Description
`php artisan make:controller InvoiceController --resource`	Creates a **resource controller** with standard CRUD methods (`index`, `create`, `store`, `edit`, `update`, `destroy`).
`php artisan make:migration create_invoices_table`	Creates a new migration file for the invoices table.
`php artisan migrate:rollback`	Reverts the last batch of migrations.
`php artisan make:seeder InvoiceSeeder`	Creates a seeder to populate initial data.
`php artisan db:seed --class=InvoiceSeeder`	Runs a specific seeder class.

Optimization & Maintenance Commands

Command	Description
`php artisan cache:clear`	Clears the application cache.
`php artisan config:clear`	Clears the configuration cache.
`php artisan route:clear`	Clears the route cache.
`php artisan view:clear`	Clears compiled Blade view files.
`php artisan optimize`	Optimizes the application for better performance.
`php artisan optimize:clear`	Clears all caches and optimizations.

Debugging & Environment Commands

Command	Description
`php artisan tinker`	Opens Laravel's interactive REPL (useful for testing models and queries).
`php artisan route:list`	Displays all registered routes with their methods, controllers, and middleware.

Command	Description
`php artisan env`	Displays the current environment (local, production, etc.).
`php artisan down`	Puts the application in maintenance mode.
`php artisan up`	Brings the application back online.

Developer Tips:

- Keep your code **modular** and **PSR-12 compliant**.
- Use `php artisan make:` commands to generate components.
- Always work on a **local branch** before deployment.
- Document all new functions and customizations.

Contact & Credits:

Developed by: RokByte Technologies
Website: https://rokbyte.com
Support Email: rokbyte@gmail.com

Credits:

- Laravel Framework
- Bootstrap 5
- jQuery
- Chart.js