HireBoard

APPLICANT TRACKING SYSTEM

Module Development Guide

How to build a new module for HireBoard, step by step.

Version 1.0.0

June 2026

Contents

HireBoard is built around a simple, self-contained module system. This guide explains how modules work and walks you through building a new one from scratch — folder structure, configuration, migrations, permissions, routes, controllers, and views — using a complete worked example.

1. How the Module System Works

A **module** is a self-contained folder under `Modules/` that bundles everything a feature needs: its configuration, database migrations, routes, controllers, models, and views. Modules can be enabled or disabled from **Admin → Modules** without touching code.

At every request, the `ModuleServiceProvider` reads the registry in `config/modules.php`, and for each **enabled** module it:

- loads the module's `routes.php`, automatically wrapping it in the `web`, `auth`, and `module:<key>` middleware;
- registers the module's `Views/` folder under a view namespace equal to the module key (so you reference views as `<key>::view.name`);
- collects the module's sidebar `menu` items for rendering.

When a module is **enabled** for the first time, its migrations are run — creating its tables and seeding its permissions — and it is recorded as enabled. Disabling a module simply stops it loading; **its data is never deleted**.

> **Autoloading.** The `Modules\\` namespace is PSR-4 mapped to the `Modules/` directory in `composer.json`, so module classes (`Modules\YourModule\Controllers\…`) autoload normally.

2. Anatomy of a Module

Every module follows the same layout. Here is the structure you will create:

```
Modules/
└── TaskBoard/
    ├── config.php                # Module identity, permissions, menu
    ├── routes.php                # Module routes
    ├── Controllers/
    │   └── TaskController.php
    ├── Models/
    │   └── Task.php
    ├── Migrations/
    │   ├── 2026_07_01_000000_create_tasks_table.php
    │   └── 2026_07_01_000001_seed_task_board_permissions.php
    └── Views/
        └── tasks/
            └── index.blade.php
```

3. The `config.php` Reference

Every module must contain a `config.php` that returns an array describing it. Each key is explained below.

Key	Type	Description
`name`	string	Human-readable name shown in the Module Manager and sidebar.
`key`	string	Unique snake_case identifier. Must match the key used in `config/modules.php` and the view namespace.
`version`	string	Module version, e.g. `1.0.0`.
`description`	string	Short description shown on the Module Manager card.
`author`	string	Module author.
`icon`	string	An SVG path (Heroicon outline `d` attribute) used for the card and sidebar icon.
`depends_on`	array	Keys of modules that must be enabled first. Empty for none.
`routes_file`	string	Absolute path to the module's routes file — use `__DIR__ . '/routes.php'`.
`migrations_path`	string	Absolute path to the migrations folder — use `__DIR__ . '/Migrations'`.
`permissions`	array	Map of `'permission.name' => 'Label'` the module defines.
`menu`	array	Sidebar items: each has `label`, `route`, `icon`, `permission`, and `section` (`recruitment` or `administration`).

4. Step by Step: Building the "TaskBoard" Module

We'll build a small but complete module: a simple list of recruiting tasks. It demonstrates every part of the system. Create the folder `Modules/TaskBoard/` and add the files below.

4.1 — Create `config.php`

```
<?php
// Modules/TaskBoard/config.php

return [
    'name'        => 'Task Board',
    'key'         => 'task_board',
    'version'     => '1.0.0',
    'description' => 'A simple to-do board for recruiting tasks.',
    'author'      => 'Your Name',

    // Heroicon outline path (clipboard-list)
    'icon' => 'M9 5H7a2 2 0 00-2 2v12a2 2 0 002 2h10a2 2 0 002-2V7a2 2 0 '
            . '00-2-2h-2M9 5a2 2 0 002 2h2a2 2 0 002-2M9 5a2 2 0 012-2h2a2 2 0 012 2',

    'depends_on'      => [],
    'routes_file'     => __DIR__ . '/routes.php',
    'migrations_path' => __DIR__ . '/Migrations',

    'permissions' => [
        'tasks.view'   => 'View Tasks',
        'tasks.create' => 'Create Tasks',
        'tasks.edit'   => 'Edit Tasks',
        'tasks.delete' => 'Delete Tasks',
    ],

    'menu' => [
        [
            'label'      => 'Tasks',
            'route'      => 'tasks.index',
            'icon'       => 'M9 5H7a2 2 0 00-2 2v12a2 2 0 002 2h10a2 2 0 002-2V7a2 2 0
00-2-2h-2',
            'permission' => 'tasks.view',
            'section'    => 'recruitment',
        ],
    ],
];
```

4.2 — Register the module

Add your module's key and config path to the registry in `config/modules.php`:

```
'modules' => [
    // … existing core modules …
    'task_board' => base_path('Modules/TaskBoard/config.php'),
],
```

> **Keys must match.** The registry key (`task_board`), the `key` in `config.php` , and the view namespace must all be identical.

4.3 — Create the migrations

First, the table migration:

```
<?php
// Modules/TaskBoard/Migrations/2026_07_01_000000_create_tasks_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('tasks', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->text('notes')->nullable();
            $table->boolean('is_done')->default(false);
            $table->foreignId('user_id')->nullable()->constrained()->nullOnDelete();
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('tasks');
    }
};
```

Second, the **permission-seed migration**. HireBoard uses a migration-style seeder (run when the module is enabled) to create the module's permissions and assign them to the default roles. Follow this exact pattern:

```
 <?php
// Modules/TaskBoard/Migrations/2026_07_01_000001_seed_task_board_permissions.php

use Illuminate\Database\Migrations\Migration;
use Spatie\Permission\Models\Permission;
use Spatie\Permission\Models\Role;

return new class extends Migration
{
    // 'permission.name' => [roles that receive it by default]
    private array $permissions = [
        'tasks.view'   => ['Admin', 'HR Manager', 'Recruiter', 'Interviewer', 'Viewer'],
        'tasks.create' => ['Admin', 'HR Manager', 'Recruiter'],
        'tasks.edit'   => ['Admin', 'HR Manager', 'Recruiter'],
        'tasks.delete' => ['Admin', 'HR Manager'],
    ];

    public function up(): void
    {
        app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();

        foreach ($this->permissions as $name => $roleNames) {
            $permission = Permission::firstOrCreate(
                ['name' => $name, 'guard_name' => 'web']
            );

            foreach ($roleNames as $roleName) {
                $role = Role::where('name', $roleName)->where('guard_name', 'web')->first();
                if ($role && ! $role->hasPermissionTo($permission)) {
                    $role->givePermissionTo($permission);
                }
            }
        }
    }

    public function down(): void
    {
        app()[\Spatie\Permission\PermissionRegistrar::class]->forgetCachedPermissions();

        foreach (array_keys($this->permissions) as $name) {
            $permission = Permission::where('name', $name)->where('guard_name', 'web')-
>first();
            if ($permission) {
```

```
                $permission->roles()->detach();
                $permission->delete();
            }
        }
    }
};
```

> **Default roles** you can assign to are: `Admin`, `HR Manager`, `Recruiter`, `Interviewer`, and `Viewer`. The `Admin` role is a super-admin and is granted access to everything automatically, but listing it keeps the intent explicit.

4.4 — Create the model

```
<?php
// Modules/TaskBoard/Models/Task.php

namespace Modules\TaskBoard\Models;

use Illuminate\Database\Eloquent\Model;

class Task extends Model
{
    protected $fillable = ['title', 'notes', 'is_done', 'user_id'];

    protected $casts = ['is_done' => 'boolean'];
}
```

4.5 — Create `routes.php`

The provider automatically applies the `web`, `auth`, and `module:task_board` middleware to this file, so you only declare routes and gate each one by its permission with `can:`.

```php
<?php
// Modules/TaskBoard/routes.php

use Illuminate\Support\Facades\Route;
use Modules\TaskBoard\Controllers\TaskController;

Route::prefix('tasks')->name('tasks.')->group(function () {

    Route::get('/', [TaskController::class, 'index'])
        ->middleware('can:tasks.view')->name('index');

    Route::post('/', [TaskController::class, 'store'])
        ->middleware('can:tasks.create')->name('store');

    Route::put('/{task}', [TaskController::class, 'update'])
        ->middleware('can:tasks.edit')->name('update');

    Route::delete('/{task}', [TaskController::class, 'destroy'])
        ->middleware('can:tasks.delete')->name('destroy');
});
```

4.6 — Create the controller

```php
<?php
// Modules/TaskBoard/Controllers/TaskController.php

namespace Modules\TaskBoard\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Modules\TaskBoard\Models\Task;

class TaskController extends Controller
{
    public function index()
    {
        $tasks = Task::latest()->paginate((int) setting('pagination_limit', 15));

        // Views are namespaced by the module key:
        return view('task_board::tasks.index', compact('tasks'));
    }

    public function store(Request $request)
    {
        $data = $request->validate([
            'title' => ['required', 'string', 'max:255'],
            'notes' => ['nullable', 'string'],
        ]);
        $data['user_id'] = $request->user()->id;

        Task::create($data);

        return back()->with('success', 'Task created.');
    }

    public function update(Request $request, Task $task)
    {
        $task->update($request->validate([
            'title'   => ['required', 'string', 'max:255'],
            'is_done' => ['boolean'],
        ]));

        return back()->with('success', 'Task updated.');
    }

    public function destroy(Task $task)
```

```
    {
        $task->delete();

        return back()->with('success', 'Task deleted.');
    }
}
```

4.7 — Create the view

Views live in the module's `Views/` folder and are referenced with the `<key>::` namespace. Extend the application layout to inherit the sidebar, top bar, and styling.

```
 {{-- Modules/TaskBoard/Views/tasks/index.blade.php --}}
@extends('layouts.app')

@section('title', 'Tasks')

@section('content')
    <h1 class="text-xl font-bold text-slate-900">Tasks</h1>

    <div class="card" style="padding:20px;margin-top:16px;">
        @forelse ($tasks as $task)
            <div style="display:flex;justify-content:space-between;padding:10px 0;border-
bottom:1px solid #f1f5f9;">
                <span>{{ $task->title }}</span>
                <span class="badge-slate">{{ $task->is_done ? 'Done' : 'Open' }}</span>
            </div>
        @empty
            <p class="text-slate-500">No tasks yet.</p>
        @endforelse

        <div style="margin-top:16px;">{{ $tasks->links() }}</div>
    </div>
@endsection
```

4.8 — Enable the module

Sign in as an administrator, go to **Admin** → **Modules**, find your **Task Board** card, and click **Enable**. HireBoard runs your migrations (creating the table and seeding permissions), records the module as enabled, and your **Tasks** item appears in the sidebar. Done.

That's a complete module. Configuration, schema, permissions, routing, controller, model, view, menu — all self-contained in one folder and toggle-able from the admin panel.

5. Permissions & Roles

HireBoard uses the Spatie permission system. Define your permissions in two places that must agree: the `permissions` array in `config.php` (for display) and the permission-seed migration (which actually creates them and assigns them to roles). Gate routes with `can:<permission>` and check in views with `@can('<permission>')`. The `Admin` role bypasses all permission checks, so administrators always have full access.

6. Routes & Middleware

For normal modules you never add `web`, `auth`, or the module guard yourself — the provider applies them when it loads your `routes.php`. The `module:<key>` guard ensures routes return 404 when the module is disabled. Gate individual routes by permission with `->middleware('can:…')`.

Public modules. A module that needs unauthenticated, public routes (like the Career Portal) can be listed as a public module in the provider, in which case it manages its own middleware inside `routes.php` instead of receiving `auth` automatically.

7. Views & the Layout

Your module's `Views/` directory is registered under a namespace equal to the module key, so a file at `Views/tasks/index.blade.php` is referenced as `task_board::tasks.index`. Extend `layouts.app` to inherit the application shell (sidebar, top bar, flash messages, and compiled styles), and reuse the existing component classes — such as `card`, `badge-slate`, `btn-primary`, and `btn-danger` — so your module looks native.

Styling note. The application's CSS is pre-compiled and shipped. Reuse the existing component classes and inline styles rather than introducing brand-new utility classes, which would not appear in the compiled stylesheet unless the assets are rebuilt.

8. Sidebar Menu Integration

Add one entry to the `menu` array in `config.php` for each sidebar link your module needs. Set `section` to `recruitment` or `administration` to choose the group, and set `permission` so the item is hidden

from users who lack access. The `route` is the named route from your `routes.php`, and `icon` is a Heroicon outline path.

9. Dependencies

If your module relies on another, list that module's key in `depends_on`. The Module Manager shows the requirement on the card, helping administrators enable modules in the right order.

10. Conventions & Best Practices

- **One folder, one feature.** Keep everything a module needs inside its own directory.
- **Namespace consistently.** Classes live under `Modules\YourModule\…`; views under the module key namespace.
- **Seed permissions via the migration pattern** shown above, so enabling the module sets up access correctly.
- **Gate everything.** Protect every route with `can:` and hide menu items by permission.
- **Never assume the module is enabled.** Disabling must be safe — never destroy data on disable.
- **Reuse the shell and component styles** so new modules feel native.

11. Enable / Disable Lifecycle

Enabling a module runs its migrations (idempotently — already-run migrations are skipped), seeds its permissions, marks it enabled, and refreshes the enabled-modules cache so it appears immediately. Disabling a module stops its routes, views, and menu from loading but leaves its tables and data untouched, so it can be safely re-enabled later.

12. Support & Contact

Building on HireBoard or extending it for a client? We're happy to help:

- **Product support (for purchased items)** — support@mes-dev.com. Include your CodeCanyon **purchase code** and your HireBoard version.
- **Custom module development, integrations, or a new application** — contact@mes-dev.com.