Act like a senior Laravel architect and full‑stack engineer who specializes in turning Laravel API projects into full, production‑grade admin dashboards using the AdminLTE template. You write meticulous, long, deeply detailed answers that include complete code, diffs, commands, and testing. You do not hand‑wave; you provide everything needed to paste into a real project and run.

OBJECTIVE
Build an Admin Control Panel (within the same Laravel project) for a real‑estate classifieds website that already has a Laravel API. The panel must use AdminLTE, auto‑discover what exists in the project, and then generate new admin pages and features based on the available models/DB schema and business needs for real‑estate listings.

LANGUAGE & OUTPUT STYLE
- Write in English.
- Be exhaustive and explicit. Prefer full files over snippets. Avoid “example only”.
- For every file you modify, show a unified diff. For every new file, show the complete file.
- Provide copy‑pasteable terminal commands.
- Use numbered steps and clear headings.
- Where choices exist (e.g., AdminLTE integration methods), briefly compare, then pick one based on the project’s detected setup, and proceed decisively.

ASSUMPTIONS & DISCOVERY (perform first)
1) Project Scan (read what I provide next: repo tree, composer.json, package.json, routes, migrations, models, config/*, resources/*, .env.example). If any of these aren’t provided, proceed with robust defaults and clearly mark TODOs.
2) Detect:
   - Laravel version from composer.json.
   - Auth stack (Laravel Breeze/Fortify/Jetstream/custom).
   - Asset tool (Vite vs. Mix) from package.json and resources structure.
   - Existing models/migrations relevant to real estate (e.g., Listing/Property, Category/Type, City, Neighborhood, Amenity, Photo/Media, User/Agent, Lead/Inquiry, Package/Plan, Favorite, SavedSearch).
   - API routes & controllers that the admin must coordinate with.
3) Produce a brief “Current State Report” summarizing findings and risks (1–2 paragraphs).

ARCHITECTURE & CONVENTIONS
- Keep the admin app inside the same Laravel project (no separate app).
- Namespaces: App\Http\Controllers\Admin, App\Http\Requests\Admin, App\Policies, App\View\Components\Admin, App\Services, App\Repositories (if needed).
- Views under resources/views/admin with a shared layout.
- Authorization via policies + gates. Role/permission via spatie/laravel-permission.
- Media handling via spatie/laravel-medialibrary (for listing galleries).
- Validation with Form Request classes.
- Follow PSR‑12; include PHPDoc and types where the Laravel version allows.

- Use Eloquent resources and pagination where applicable.
- Prefer server-side DataTables or efficient index queries with filters.

ADMINLTE INTEGRATION (choose one and fully implement)
A) Package approach (recommended): jeroennoten/laravel-adminlte
   - Install via composer.
   - Publish config, views, and assets.
   - Configure title, logo, menu, sidebar, dark mode.
   - Wire to Laravel auth guards/middleware.
B) Direct AdminLTE integration:
   - Install via NPM (AdminLTE + Bootstrap + dependencies).
   - Configure Vite/Mix entry points, compile assets, and create base Blade layout.

Provide the exact commands and all changed files for the chosen path, plus a short note describing why it was selected given the detected project.

SECURITY, AUTH & RBAC
1) Require login for all admin routes via a dedicated guard or middleware.
2) Install and configure spatie/laravel-permission with roles:
   - SuperAdmin, Admin, Moderator, Agent, User.
3) Seed an initial SuperAdmin and baseline permissions.
4) Add policies for Listings, Users, Media, Settings, and Approvals.
5) Add middleware to protect routes, plus menu authorization checks in Blade.

ROUTES & LAYOUT
- Create routes under routes/admin.php. Register in RouteServiceProvider with a proper prefix (/admin) and name spacing (admin.*).
- Build a master layout with AdminLTE:
  - Header (user menu, notifications), Sidebar (role-aware), Footer, Breadcrumbs.
  - Blade components for: PageHeader, Breadcrumbs, StatCard, DataTable, ImageUploader, FormGroup, Toggle, TagInput, MapPicker (if applicable).
- Provide example sidebar configuration and dynamic active states.

FEATURES & PAGES TO IMPLEMENT (CRUD + extras)
For each entity below, implement: migration (if missing), model tweaks, policy, Form Request validation, Resource Controller (index/create/store/show/edit/update/destroy), routes, Blade views (index with filters, create/edit forms with validation, show with details), server-side search/sort/pagination, bulk actions (where noted), flash messages, and tests.
1) Dashboard
   - KPI cards: Active Listings, Pending Approvals, Featured Listings, New Leads (7d).
   - Charts (Chart.js): Listings per day (30d), Top cities, Lead conversion.
   - Recent activity (last 10 events).
2) Listings (Properties)

   - Fields: title, slug, description, type (rent/sale), category/type, price, currency, period (monthly/yearly), bedrooms, bathrooms, area, unit, year_built, furnished, amenities[], address, city_id, neighborhood_id, latitude, longitude, status (draft/pending/active/rejected/expired), is_featured, owner/agent_id, expires_at.
   - Filters: status, type, category, city, price range, bedrooms, created_at range, featured.
   - Bulk actions: approve, reject (with reason), feature/unfeature, publish/unpublish, delete/restore.
   - Media gallery: multiple images with drag-drop order, cover image, alt text, responsive conversions (thumb, medium, large).
   - SEO: slug generation, meta title/description fields (optional).
3) Categories / Property Types
   - Name, slug, icon (optional), parent (optional).
4) Locations (Cities, Neighborhoods)
   - City with country, state; Neighborhood linked to City.
5) Amenities
   - Name, icon (optional), category (optional).
6) Users & Agents
   - Manage users, assign roles, toggle active, view agent's listings and leads.
7) Leads / Inquiries
   - Source (listing, contact form), contact info, message, status (new/in-progress/closed), assignee, internal notes.
   - Quick actions and email reply log (store meta only).
8) Media Library
   - Global media manager view filtered by model/type.
9) Moderation Queue
   - Listings awaiting approval; side-by-side detail panel; approve/reject with notes.
10) Settings
    - Site settings (name, logo, contact, address).
    - Listing settings (default status, expiration days, featured limits).
    - SEO (site title, meta defaults).
    - Storage settings (image sizes), map keys (if used).
    - Use DB-backed settings (e.g., spatie/laravel-settings or simple key-value table). Provide full implementation.

OPTIONAL ENHANCEMENTS (implement if feasible with detected stack)
- Favorites & Saved Searches admin views.
- Packages/Plans management (if monetization exists).
- Import/Export CSV for Listings, Amenities, Locations (with validation).
- Localization (Laravel localization). Use trans() in views; provide en/ messages.
- Scout integration (Algolia/Meilisearch) for admin quick search bar.

IMPLEMENTATION DETAILS (produce all of this)
1) Installation & Setup
   - Exact composer/npm commands.

  - Vite/Mix config updates.
  - storage:link and intervention for images if needed.
2) Migrations
  - Create missing tables/columns with careful, idempotent migrations.
  - Never break existing API schema; use additive changes.
3) Models
  - Fillable/guarded, casts, relationships, scopes for filters (e.g., scopeCity, scopeStatus).
  - Slug generation (custom or package), observers if needed.
4) Requests
  - Separate Store/Update Request classes with explicit rules and messages.
5) Controllers
  - Resource controllers in App\Http\Controllers\Admin\*. Include index filtering logic, authorization, transactions for writes, robust error handling.
6) Policies
  - Per-entity with create/update/delete/approve/feature gates as appropriate.
7) Views (Blade)
  - Base layout extends AdminLTE.
  - Views: index/create/edit/show/partials/_form for each entity.
  - Components for repeatable UI: <x-admin.form.group>, <x-admin.image-uploader>, etc. Provide full component code.
  - Datatables: either Yajra DataTables server-side or handcrafted server pagination + filters. Pick one and implement fully.
8) Routes
  - routes/admin.php file with Route::middleware([...])->prefix('admin')->name('admin.')->group(...).
9) Sidebar/Menu
  - Role-aware menu entries. Active state highlights.
10) Seeders & Factories
  - Roles/permissions seeder; SuperAdmin user; sample cities/amenities; sample listings and leads.
11) Tests
  - Use Pest or PHPUnit (match project). Provide tests for:
    - Policy/authorization for Listings.
    - Listings CRUD (happy + validation failures).
    - Approvals & bulk actions.
    - Settings update.
    - Menu visibility by role.
12) Documentation
  - “RUNBOOK.md” with setup steps, commands, common pitfalls, and rollback notes.
  - “MAPPING.md” mapping existing API models/fields to admin fields.
  - “PERMISSIONS.md” listing all permissions and which roles get them.

DELIVERABLE FORMAT (strict)
- Start with “Current State Report”.

- Then “Plan & Decisions” (why and what you will build).
- Then “Commands” (install, publish, build).
- Then “File Changes”:
  - For each modified file: show a unified diff patch beginning with the file path and @@ hunks.
  - For each new file: show the full file with a clear file path heading.
- Then “Database Changes” (migrations).
- Then “Seeders & Factories”.
- Then “Tests”.
- Then “How to Run & Verify” (manual QA steps and URLs).
- Then “Post-Install Checklist”.
- Then “Next Improvements” (short list).

REAL-ESTATE DOMAIN CHECKLIST (use for forms, filters, validations)
- Required listing fields: title, type (rent/sale), category/type, price, currency, location (city, neighborhood), bedrooms/bathrooms, area + unit, images[] min 1 for publish, status.
- Optional: furnished, parking, balcony, heating/cooling, year_built, floor, total_floors, description, video_url, map coords.
- Validation: sensible min/max, currency format, integer/decimal constraints, safe HTML (strip scripts), image mime/size limits.
- Filters: by status, city, category, type, price range, bedrooms, date range, featured.
- Lifecycle: draft → pending → approved/active → expired; record approver_id and timestamps; rejection reason field.
- Media: enforce cover image; generate conversions; alt text; delete orphaned media.

IF INFORMATION IS MISSING
- If a required model/table doesn’t exist, add it with migrations and explain the rationale.
- If auth stack is absent, scaffold minimal Laravel auth guarded to “admin”.
- If assets pipeline is unclear, default to Vite and show how to migrate if on Mix.
- Never block on missing info; proceed with best, clearly marked defaults and explain how to adapt.

CONSTRAINTS
- Don’t break existing API endpoints; keep routes/api.php intact.
- Back up/commit before making changes (show git commands).
- Keep sensitive data out of code; use .env for keys.

AT THE END
- Provide a Verification Script (a checklist of pages/URLs to click and what should be visible).
- Provide a quick “Rollback Plan” (how to revert migrations and config).
- Provide time/space complexity notes for heavy queries and suggest indexes.

INPUT YOU WILL RECEIVE FROM ME NEXT
I will either:
- Paste the repo tree and key files (composer.json, package.json, routes, models, migrations), or

- Tell you to proceed with defaults (no further files). In that case, generate the full scaffold and mark integration points.

Now begin:
Step 1) Perform the “Project Scan” based on the inputs I will paste (or proceed with defaults if none), and produce the “Current State Report”.
Step 2) Present “Plan & Decisions”.
Step 3) Execute all implementation steps and output the deliverables in the exact format above, with full code, diffs, migrations, seeds, tests, and commands.
Take a deep breath and work on this problem step-by-step.