ProjectX — Role-based Folder Structure + Admin Control Mapping

This document proposes a clear folder layout for your existing PHP project (no PDO) and a detailed mapping of **admin** pages/actions to your database schema. It focuses on starting with Admin controls (as you asked), then outlines folder templates for other roles (head, senior, member, applicant, guest), shared components, and recommended ACL checks.

Goals

1. Keep code organized by role for easier permissions and UI separation.
2. Map admin UI/actions directly to DB tables from your dump.
3. Keep a consistent `actions/` vs `pages/` pattern: pages render UI, actions perform POST/CRUD.
4. Use simple `mysqli` (prepared statements) and role checks in a central middleware.

Top-level folder structure

```
/ (project root)
├─ app/
│  ├─ admin/                 # Admin pages & actions
│  │  ├─ pages/              # UI pages (GET)
│  │  │  ├─ dashboard.php
│  │  │  ├─ users/
│  │  │  │  ├─ index.php
│  │  │  │  ├─ view.php
│  │  │  │  └─ edit.php
│  │  │  ├─ projects/
│  │  │  │  ├─ index.php
│  │  │  │  ├─ create.php
│  │  │  │  └─ edit.php
│  │  │  ├─ payments.php
│  │  │  ├─ invoices.php
│  │  │  ├─ coupons.php
│  │  │  ├─ documents.php
│  │  │  ├─ credentials.php
│  │  │  ├─ resources.php
│  │  │  ├─ timesheets.php
│  │  │  └─ reports.php
│  │  └─ actions/           # POST/PUT/DELETE endpoints
│  │     ├─ users_create.php
│  │     ├─ users_update.php
│  │     ├─ users_delete.php
│  │     ├─ projects_create.php
│   │     ├─ projects_update.php
│   │     ├─ projects_delete.php
│   │     ├─ payments_mark_refunded.php
│   │     ├─ coupons_toggle.php
│   │     └─ ...
│   ├─ head/                # same pattern for head role
│   ├─ senior/              # same pattern for senior role
│   ├─ member/              # same pattern for member role (limited)
│   ├─ applicant/           # application / resume pages
│   └─ public/              # public pages accessible without role
├─ includes/
│   ├─ init.php             # DB connection, session start, common helpers
│   ├─ auth.php             # role-check functions, is_admin(),
require_role()
│   ├─ header.php
│   └─ footer.php
├─ assets/
│   ├─ css/
│   └─ js/
├─ api/                     # ajax endpoints if you use JS-driven UI
├─ actions/                 # legacy/global actions (if any)
├─ storage/                 # uploaded files (resumes, docs, images)
└─ vendor/                  # external libs if used
```

Keep `pages/` files only responsible for rendering and fetching read-only data; `actions/` files should validate, check role, perform DB changes, and redirect back to pages with flash messages.

Central auth & DB initialization (includes/init.php)

- Create a single `init.php` that every page/action includes as the first line.
- Responsibilities:
- `session_start()`
- open `mysqli` connection (`$DB = new mysqli(...)`) — use `charset utf8mb4`
- set timezone/locale defaults
- require `auth.php` which exposes `current_user()`, `require_role($role)`, `check_permission($capability)`

Example usage in admin page:

```
require_once __DIR__ . '/../../includes/init.php';
require_role('admin');
// now you can query DB using $DB
```

Admin: pages & actions mapped to DB tables

Below are the **recommended admin pages** and the DB tables they manage (from your dump). For each page I list the main actions and suggested filenames.

1) Admin Dashboard

- File: `app/admin/pages/dashboard.php`
- Displays KPIs: total users, active projects, open applications, pending payments, revenue (payments), recent error_logs, recent audit_logs.
- Queries: `COUNT(*) FROM users`, `COUNT(*) FROM projects WHERE status='open'`, `SELECT SUM(amount) FROM payments WHERE status='success'`, `SELECT * FROM error_logs ORDER BY at DESC LIMIT 10`.

2) Users Management

- Pages:
- `app/admin/pages/users/index.php` — list users with filters (role, status)
- `app/admin/pages/users/view.php` — view profile + related resources (applications, invoices, certificates)
- `app/admin/pages/users/edit.php` — admin edit user (role/status)
- Actions:
- `app/admin/actions/users_create.php` — create user (admin-only)
- `app/admin/actions/users_update.php` — update profile, role, status
- `app/admin/actions/users_delete.php` — hard-delete or set `status='inactive'`
- DB tables: `users`, `profile_cards`, `certificates`, `applications`, `payments`, `invoices`.

3) Projects Management

- Pages:
- `app/admin/pages/projects/index.php` — list + search projects
- `app/admin/pages/projects/create.php` — form to create project
- `app/admin/pages/projects/edit.php` — update project, seats, status
- Actions:
- `app/admin/actions/projects_create.php`
- `app/admin/actions/projects_update.php`
- `app/admin/actions/projects_delete.php`
- DB tables: `projects`, `project_milestones`, `applications`, `offers`.

4) Applications & Offers

- Pages:
- `app/admin/pages/applications/index.php` — list and filter by project/decision
- `app/admin/pages/applications/view.php` — view resume, make decision
- `app/admin/pages/offers/index.php` — list offers
- Actions:
- `app/admin/actions/application_decide.php` — update `decision` + `decision_note`
- `app/admin/actions/offers_send.php` — create `offers` row (link to templates)
- DB tables: `applications`, `offers`, `offer_templates`.

5) Payments & Invoices

- Pages:
- `app/admin/pages/payments.php` — list payments, filter, export CSV
- `app/admin/pages/invoices.php` — list invoices, view invoice (QR), re-generate
- Actions:
- `app/admin/actions/payments_update.php` — change status (mark refunded)
- `app/admin/actions/invoices_create.php` — create invoice, set `invoice_no`
- DB tables: `payments`, `invoices`, `coupons`.

6) Coupons

- Pages: `app/admin/pages/coupons.php` (list/create/edit)
- Actions: `app/admin/actions/coupons_create.php`, `coupons_update.php`, `coupons_toggle.php` (activate/deactivate)
- DB table: `coupons`.

7) Documents & Document Acknowledgement

- Pages: `app/admin/pages/documents.php`, `app/admin/pages/documents/view.php`
- Actions: `app/admin/actions/documents_upload.php`, `documents_delete.php`
- Admin can view `doc_ack` records (who acknowledged which doc).
- DB tables: `documents`, `doc_ack`.

8) Credentials (secure storage)

- Pages: `app/admin/pages/credentials.php` — list labels, visible roles
- Actions: `app/admin/actions/credentials_create.php`, `credentials_update.php`, `credentials_delete.php`
- Store `enc_value` but **do not** keep encryption keys in DB.
- DB table: `credentials`.

9) Resources & Assignments

- Pages: `app/admin/pages/resources.php`, `resources_assign.php`
- Actions: `app/admin/actions/resources_create.php`, `assign_resource.php`, `return_resource.php`
- DB: `resources`, `resource_assignments`.

10) Timesheets & Reports

- Pages: `app/admin/pages/timesheets.php`, `reports.php` (aggregated hours, billable summaries)
- DB: `timesheets`, `daily_reports` (use both).

11) Logs & Queue

- Error logs page: `app/admin/pages/error_logs.php` — view recent errors and stack traces
- Audit logs page: `app/admin/pages/audit_logs.php` — search by user/action/entity
- Mail queue monitor page: `app/admin/pages/mail_queue.php` — retry / flush
- DB: `error_logs`, `audit_logs`, `mail_queue`.

12) Certificates

- Pages: `app/admin/pages/certificates.php` — list and generate QR
- Actions: `app/admin/actions/certificates_generate.php`
- DB: `certificates`.

Example file responsibilities (pattern)

- `app/admin/pages/users/index.php`
- require init + require_role('admin')
- fetch paginated users list using prepared `SELECT` with filters

- render table + links to view/edit

- `app/admin/actions/users_update.php`

- require init + require_role('admin')
- validate POST params
- prepare `UPDATE users SET name=?, role=?, status=? WHERE id=?`
- write audit log into `audit_logs`
- redirect back to users list with flash

ACL & Permission model

- Use role strings exactly as in DB: `guest|applicant|member|senior|head|admin`.
- Create helper `require_role($role_or_array)` which exits/redirects if the current user lacks that role.
- For finer permissions, implement `check_permission($capability)` that maps to capabilities such as `projects.create`, `users.manage`, `finance.view` and map those to roles in a small config array `config/permissions.php`.

Example mapping (in `config/permissions.php`):

```
return [
  'admin' => ['*'],
  'head' => ['projects.*','resources.*','users.view','timesheets.view'],
  'senior' => ['projects.view','applications.review','timesheets.submit'],
  'member' => ['projects.view','timesheets.submit'],
  'applicant' => ['applications.create'],
  'guest' => []
];
```

Shared components & reusable helpers

- `includes/pagination.php` — common pagination helper

- `includes/upload.php` — safe file upload with extension whitelist, storage path, and virus-scan hook (if available)
- `includes/flash.php` — flash messages in session
- `includes/audit.php` — function `audit($user_id,$action,$entity,$entity_id, $meta_array)` that inserts into `audit_logs`

Security & operational notes

- All DB writes must be done through prepared statements (`$stmt = $DB->prepare(...)` and bind params) to prevent SQL injection.
- Passwords: continue using bcrypt/argon2 via `password_hash()` and `password_verify()`.
- Uploaded files: store outside webroot or protect with `.htaccess` and serve via a controller that checks permission.
- Rate-limit admin actions that change financial state (payments/refunds).
- For long-running tasks (generate invoices, send bulk emails), use background workers (cron + CLI script) and the `mail_queue` table.

Next steps I already prepared for you

1. A ready-to-use folder skeleton (file list) for the Admin role with page and action filenames (above).
2. A mapping of each Admin page/action to the database tables and key queries to implement.
3. Middleware & helper suggestions for auth, audit, upload, flash messages.

If you want, I can now: - generate the **actual PHP skeleton files** (empty templates with `require_once '../../includes/init.php'; require_role('admin');` and file-specific placeholders) and put them into a zip for download; **or** - produce the **exact SQL queries** for each admin action (e.g., `projects_create.php` insert query + parameter list + example prepared statement code using `mysqli`) — useful if you want code-ready actions.

Tell me which one you want me to produce first and I will generate it right away.

Document created: ProjectX - Role-based Folder Structure & Admin Mapping