Below is a comprehensive Markdown migration guide to rebuild this Django project as a FastAPI application. It captures the logic of every view/helper/management command so it can be re-implemented cleanly, with recommended improvements for security and architecture.

CSI Kalamela App → FastAPI Migration Guide

1) High-Level Architecture (Target)
- **FastAPI app layout**
  - `app/main.py`: create FastAPI instance, mount routers, exception handlers, middleware (CORS, auth).
  - `app/config.py`: settings via environment variables (DB URL, secret key, debug, allowed origins).
  - `app/db.py`: SQLAlchemy engine/sessionmaker, Alembic migrations.
  - `app/models/*.py`: SQLAlchemy models.
  - `app/schemas/*.py`: Pydantic request/response models.
  - `app/api/routers/*.py`: route modules per domain (`auth`, `conference`, `kalamela`, `admin`, `files`).
  - `app/services/*.py`: business logic (formerly Django views/helpers).
  - `app/security.py`: password hashing (Passlib), JWT auth, role/permission checks.
  - `app/storage.py`: media upload handling (payment proofs), validation, S3/local support.
  - `app/tasks/`: optional background tasks (email, async exports).
  - `tests/`: pytest + httpx/async clients.
- **Auth model**: Keep custom user fields (roles: admin, regd_unit, district_official). Use JWT bearer auth; add refresh tokens if needed.
- **Static/templates**: Rebuild UI separately (SPA or server-rendered). This guide focuses on APIs.

---

2) Data Models (map from Django to SQLAlchemy)

auth_app
- `ClergyDistrict(id, name)`
- `UnitName(id, clergy_district_id -> ClergyDistrict, name)`
- `CustomUser(id, email unique, username, first_name, last_name, phone_number, user_type ['1','2','3'], unit_name_id?, clergy_district_id?, conference_member_count?, conference_official_count?, conference_id?)`
  - many-to-many groups/permissions can be omitted or replaced with role enum.
- `UnitRegistrationData(id, registered_user_id -> CustomUser, status)`
- `UnitDetails(id, registered_user_id -> CustomUser, registation_year, number_of_unit_members)`

- `UnitMembers(id, registered_user_id -> CustomUser, name, gender, dob, number, qualification, blood_group)`
- `UnitOfficials(id, registered_user_id -> CustomUser, president/vice/secretary/joint_secretary/treasurer fields)`
- `UnitCouncilor(id, registered_user_id -> CustomUser, unit_member_id -> UnitMembers)`

conference
- `ConferenceRegistrationData(id, district_official_id -> CustomUser, status)`
- `Conference(id, title, details, added_on, status)`
- `ConferenceDelegate(id, conference_id -> Conference, officials_id -> CustomUser, members_id -> UnitMembers?)`
- `ConferencePayment(id, conference_id -> Conference, amount_to_pay, uploaded_by_id -> CustomUser, proof_path, date, status)`

kalamela
- `IndividualEvent(id, name, category, description, created_on)`
- `GroupEvent(id, name, description, max_allowed_limit, min_allowed_limit, per_unit_allowed_limit, created_on)`
- `IndividualEventParticipation(id, individual_event_id, participant_id -> UnitMembers, added_by_id -> CustomUser, chest_number, seniority_category, created_on)`
- `GroupEventParticipation(id, group_event_id, participant_id -> UnitMembers, chest_number, added_by_id -> CustomUser)`
- `KalamelaExcludeMembers(id, members_id -> UnitMembers)`
- `KalamelaPayments(id, paid_by_id -> CustomUser, individual_events_count, group_events_count, total_amount_to_pay, payment_proof_path, payment_status, created_on)`
- `IndividualEventScoreCard(id, event_participation_id -> IndividualEventParticipation, participant_id -> UnitMembers, awarded_mark, grade, total_points, added_on)`
- `GroupEventScoreCard(id, event_name, awarded_mark, chest_number, grade, total_points, added_on)`
- `Appeal(id, added_by_id -> UnitMembers, chest_number, event_name, statement, reply, status, created_on)`
- `AppealPayments(id, appeal_id -> Appeal, total_amount_to_pay, payment_type, payment_status, created_on)`

---

3) Auth & Session Flows (Django → FastAPI)

Django logic summary
- `login_page` renders login; `do_login` manually fetches user by username/phone, checks password, routes by `user_type`. No throttling, no is_active check.
- `forgot_password` resets password by email without token verification.
- `do_logout` logs out.

- Custom registration: `add_registered_user` creates unit users, sets reg number as email/username, sets `user_type=2`, creates `UnitRegistrationData` with status “Registration Started”.
- `get_unit_names` returns unit list by district (JSON).

FastAPI re-implementation
- Use JWT (`/auth/login`) with `authenticate` via email/username/phone; verify `is_active`; throttle attempts.
- `/auth/forgot-password/request` + `/auth/forgot-password/confirm` with signed token sent by email; remove unauthenticated direct reset.
- `/auth/register-unit` to create unit accounts:
  - Validate phone uniqueness, unit uniqueness, password hashing.
  - Generate registration_number; set `user_type=2`; create `UnitRegistrationData(status="Registration Started")`.
- `/auth/unit-names?district_id=` returns units.
- Role-based dependencies: `require_admin`, `require_unit`, `require_district_official`.

---

4) Admin exports & dashboards (auth_app views)

Django logic summary
- `admin_export_*` endpoints export officials/councilors/members to Excel for district/unit/all. No auth guard.
- `admin_home_page`/`admin_home_conference`: aggregate counts, unregistered lists, gender counts, top unit by member count.
- `admin_view_*` functions display unit details, officials, councilors, members, print forms, etc., with age calculations and fee totals.
- `member_export_to_excel`, `officials_export_to_excel`, `councilors_export_to_excel` produce spreadsheets.

FastAPI re-implementation
- Convert to `/admin/...` routes protected by admin JWT.
- Extract aggregations into service functions using SQLAlchemy queries with `COUNT`, `Subquery`.
- Exports: generate XLSX in-memory (openpyxl) and stream response; add audit logging.
- Age calculations: move to helper (`calc_age(dob)`).

---

5) Management Commands → FastAPI scripts/tasks

- `assign_chest_numbers`: assigns chest numbers based on DOB ranges (junior/senior) and updates participations lacking chest numbers.

- `backup_db`: DB dump per engine; not needed inside API—use ops pipeline.
- `reset_unit_login_password`: resets district officials' passwords deterministically. Replace with admin-triggered secure reset.

Re-implement as:
- Admin-only POST `/maintenance/assign-chest-numbers` (background task).
- Avoid password reset in bulk; prefer per-user reset tokens.

---

6) Conference module

Django login and access
- `conference_login`: manual auth; `user_type 1 -> admin home`, `user_type 3 -> official`.
- `conference_logout`: session logout.

Admin views (`conference_admin_home_page`, etc.)
- List conferences, districts, form to create/update/delete conference.
- `create_conference`: create active conference.
- `update_conference`: edit title/details.
- `delete_conference`: delete.
- `all_conference_info(conference_id)`: builds district-wise aggregates (officials/members counts, genders) and can export to Excel.
- `view_all_members(conference_id, district_id)`: list members by district.

Official/delegate management (`admin_views.add_delegate_officials`)
- For a POSTed `conference_id` and `member_id`, creates a district official user:
  - Password = cleaned member name + first two digits of phone.
  - Sets conference counts (hardcoded lists for max counts per district).
  - Creates `ConferenceRegistrationData` and `ConferenceDelegate` rows.
- `view_district_official`, `update_district_official`, `delete_district_official`: CRUD with propagation of counts to all users in district.

Payments (`admin_payment_info`)
- Builds district-wise officials/members/payments list; can export to Excel.

Official views (`conference/official_views.py`)
- `view_conference`: shows remaining slots, delegates, allowed counts.
- `add_deligates(member_id)`: add a member as delegate if not exceeding limits; prevents duplicates.
- `view_delegates_official`: lists delegates (members + officials), computes payment status and amount.
- `remove_conference_member`: delete delegate.

- `official_make_conference_payment`: create `ConferencePayment` with optional proof; status PAID/NOT PAID.
- `confirm_payment`: **bug** uses undefined `method`; intended to mark payment paid with proof.
- `all_conference_info_official`: exports district-only view to Excel.

FastAPI re-implementation
- Router: `/conference`.
- Auth: JWT + role guard (`admin` vs `district_official`).
- Services:
  - `conference_service.create/update/delete/list`
  - `delegate_service.add_official_from_member` (with secure password generation or invite flow)
  - `delegate_service.add_member_delegate`, with per-district limits and duplicate checks.
  - `payment_service.create_payment`, upload proof (validate size/type), status transitions, admin confirm/decline.
  - `export_service.conference_summary(conference_id, scope=all|district)` → XLSX.
- Fix confirm-payment flow; avoid deterministic passwords; enforce uniqueness on users.

---

7) Kalamela module

Auth views (`kalamela_auth_views.py`)
- `kalamela_home_page` (landing).
- `kalamela_login`: authenticate by email/password; route admin → admin home, district_official → payment or preview depending on payment exists.
- `kalamela_logout`.
- `kalamela_find_participants`: lookup participants by chest number in individual/group participations; error if none.
- `kalamela_results`: builds top-3 per individual event and per group event (adds unit/district by chest number).
- Appeals:
  - `kalamela_appeal_form`: verify chest number/event exists and within 30 minutes of score publication; renders appeal form.
  - `kalamela_appeal_form_submission`: create `Appeal` + `AppealPayments` (fixed amount 1000, status “Confirmation Pending”).
  - `kalamela_view_appeals`: lists pending appeals (payment_status “Confirmation Pending” and reply present).
- `kalamela_view_kalaprathibha`: computes top aggregated scores by gender (>=2 events, sum of points) for titles.

Official views (`kalamela_official_views.py`)
- `kalamela_official_home_page`: shows events grouped, counts remaining slots.

- Individual events:
  - `kalamela_official_select_individual_event_participant`: list eligible members (filters by district, unit, gender, seniority keywords in event name; excludes already registered and excluded members).
  - `kalamela_official_add_individual_event_participant`: uses `add_individual_participant_to_event` helper, enforces per-district max 2 per event, max 5 events per participant, reuses/assigns chest numbers based on DOB ranges.
  - `kalamela_official_view_individual_participants_list`: grouped list by event with participant details.
  - `kalamela_official_remove_individual_event_participant`: deletes participation.
- Group events:
  - `kalamela_official_select_group_event_participants`: similar, with remaining slot calc per unit and per event; uses helper.
  - `kalamela_official_add_group_event_participant`: bulk add with rules:
    - At most 2 teams per district per event.
    - Enforce max_allowed_limit per unit participation count.
    - Prevent duplicate members in same event.
    - Chest number format derives from event initials; increments per team.
  - `kalamela_official_view_group_participants_list`, `remove_group_event_participant`.
- Payments:
  - `kalamela_official_view_events_preview`: compute counts, amounts (50 per individual, 100 per group team), show payment status.
  - `kalamela_official_make_payment`: create `KalamelaPayments` row with status “Pending, No Proof Uploaded”.
  - `kalamela_official_payment_proof_upload`: upload proof file, set status “Pending, Proof Uploaded”.
  - `kalamela_official_print_form`: render summary for printing.

Admin views (`kalamela_admin_views.py`)
- Dashboard: list/add/update individual/group events.
- Unit/member management: list all units, list members by unit, edit member, exclude/unexclude members.
- View participants (all districts) individual/group.
- Edit chest numbers for group participations.
- Preview events with filters by district; compute counts/fees; expose scores presence.
- Payments admin: list payments, mark invalid (clear proof, status pending), approve (set Paid).
- Exports:
  - `kalamela_admin_export_all_events_data`: XLSX for filtered district/event(s).
  - `admin_export_all_chest_numbers`: XLSX for individual events chest numbers.
  - `kalamela_admin_export_all_scores`: XLSX top 3 results.
- Scoring:
  - `kalamela_admin_individual_events_candidates`: list participants for an event ordered by chest.

  - `kalamela_admin_add_individual_events_score`: bulk create scorecards from JSON (marks, grade, points).
  - `kalamela_admin_view_events_score`: display all scores (individual + group).
  - `kalamela_admin_view_appeals` / `..._action`: reply to appeal, set status Approved, update payment status, then recompute scores via `kalamela_admin_update_individual_event_scorecard`.
  - `kalamela_admin_view_update_individual_event_scorecard` / `...save_updated_individual_scorecard`: fetch and bulk update scorecards via JSON.
  - Group scoring parallels individual: candidates by chest_number, bulk add/update scorecards.
  - Unit-wise / district-wise results: top 3 per unit/district with sum of points.

Helper modules
- `kalamela/utility.py` (official-side helpers):
  - `list_all_individual_events(request)`: returns events annotated with remaining slots per district (max 2).
  - `list_all_group_events(request)`: returns dict event → count per district/unit participation.
  - `individual_event_members_data(request, event_obj, unit_id=None)`: filter eligible members by district, unit, gender inferred from event name (“boys/girls”), seniority inferred from name (“junior/senior”) with DOB ranges; exclude already registered and excluded members; annotate event_seniority/gender.
  - `add_individual_participant_to_event(event_obj, unit_member_obj, request, seniority_category)`: enforce per-district 2 participants per event per seniority, max 5 events per participant, avoid duplicates; assign chest_number based on DOB ranges and reuse/increment scheme.
  - `remove_individual_event_participant_from_event`.
  - `group_event_members_data(request, event_obj, unit_id=None)`: eligible members for group events excluding already registered and excluded.
  - `add_group_participant_to_event(group_event_obj, unit_id, unit_member_objs, request)`: enforce:
    - At most 2 teams per district per event.
    - Per-unit allowed limit (max_allowed_limit minus current for unit).
    - Prevent duplicates; ensure same unit grouping; generate chest_number prefix from event initials, increment team number.
  - `remove_group_event_participant_from_event`.
  - `view_all_individual_event_participants(request)`: grouped dict by event with participant details for the district.
  - `view_all_group_event_participants(request)`: nested dict event -> team_code -> participants with counts.
- `kalamela/admin_utility.py` (admin-side helpers):
  - `view_all_individual_event_participants(request, district_id)` and group variant: same as above but optional district filter.
  - `admin_export_all_events_data(request, district_id, individual_event, group_event)`: XLSX builder for group/individual participants with chest numbers.
  - `admin_export_all_chest_numbers(request)`: XLSX of chest numbers per district/event.

  - `export_all_results(request)`: top 3 results per event to XLSX.
  - `kalamela_admin_update_individual_event_scorecard(event)` (invoked from appeals flow) — implied recalculation.

FastAPI re-implementation
- Routers: `/kalamela/auth`, `/kalamela/admin`, `/kalamela/official`, `/kalamela/public`.
- Services:
  - `event_service` for individual/group CRUD, exclusion logic.
  - `participation_service` for add/remove/list with rules above.
  - `payment_service` for Kalamela payments + proof upload/validation.
  - `score_service` for bulk score ingest/update, top-N queries, kalaprathibha aggregation, unit/district summaries.
  - `appeal_service` for creation, payment linkage, SLA (30 min) enforcement, approval and score recompute.
  - `export_service` for XLSX generations (participants, chest numbers, scores, results).
- Apply consistent seniority ranges, concurrency-safe chest number assignment (transactions + unique constraints).

---

8) Security & Validation Enhancements (recommended while migrating)
- Move secrets/DB credentials to env vars; rotate exposed keys/passwords.
- Enforce JWT auth, role guards, CSRF not needed for pure API; rate-limit login/appeal/payment actions.
- Input validation with Pydantic schemas; strong password policy; uniqueness constraints on phone/email/username.
- File uploads: size/type validation, store path safely, consider S3 with presigned URLs.
- Payments: signed references, idempotency keys, auditable status transitions.
- Logging & audit for exports and admin actions.
- Use Alembic for migrations; add DB constraints (unique chest numbers per event category, etc.).

---

9) Endpoint Mapping (Django → FastAPI)

Auth / Units
- `POST /auth/login` (was `do_login`, `conference_login`, `kalamela_login`)
- `POST /auth/forgot-password/request` / `POST /auth/forgot-password/confirm`
- `POST /auth/register-unit`
- `GET /auth/unit-names?district_id=`
- `POST /auth/logout` (optional for token revocation)

Admin (registration & exports)

- `GET /admin/dashboard` (registrations, counts)
- `GET /admin/units`, `/admin/units/{id}` (members, officials, councilors)
- `GET /admin/exports/{type}` (officials, councilors, members) → XLSX

Conference
- `GET/POST /conference` (list/create)
- `PATCH/DELETE /conference/{id}`
- `POST /conference/{id}/delegates/official-from-member`
- `POST /conference/{id}/delegates/member`
- `GET /conference/{id}/summary` (admin), `GET /conference/{id}/summary/district` (official scope)
- `POST /conference/{id}/payments` (official)
- `POST /conference/payments/{payment_id}/confirm` (admin)
- `GET /conference/{id}/export` (XLSX)

Kalamela – Public
- `GET /kalamela/results` (top 3 individual & group)
- `POST /kalamela/find-participant` (by chest number)
- `POST /kalamela/appeals` (with chest_number/event, within 30 min check)
- `GET /kalamela/appeals/pending` (admin view)
- `GET /kalamela/kalaprathibha` (male/female aggregate leaders)

Kalamela – Official
- `GET /kalamela/official/events` (individual/group with remaining slots)
- `POST /kalamela/official/individual-participations` (add)
- `DELETE /kalamela/official/individual-participations/{id}`
- `POST /kalamela/official/group-participations` (bulk add)
- `DELETE /kalamela/official/group-participations/{id}`
- `GET /kalamela/official/preview` (counts, fees, payment status)
- `POST /kalamela/official/payments` (create)
- `POST /kalamela/official/payments/{id}/proof` (upload)
- `GET /kalamela/official/print` (summary)

Kalamela – Admin
- `POST /kalamela/events/individual`, `PATCH /kalamela/events/individual/{id}`, same for group
- `POST /kalamela/exclusions`, `DELETE /kalamela/exclusions/{id}`
- `GET /kalamela/participants/individual`, `GET /kalamela/participants/group`
- `PATCH /kalamela/group-participations/{id}/chest` (edit chest number)
- `GET /kalamela/admin/preview` (filters)
- `GET /kalamela/payments` (list), `POST /kalamela/payments/{id}/approve|decline`
- `POST /kalamela/scores/individual` (bulk create), `PATCH /kalamela/scores/individual` (bulk update)
- `POST /kalamela/scores/group`, `PATCH /kalamela/scores/group`
- `POST /kalamela/appeals/{id}/reply` (approve + recompute)

- `GET /kalamela/exports/{type}` (events data, chest numbers, scores, results)
- `GET /kalamela/results/unit-wise`, `GET /kalamela/results/district-wise`

---

10) Business Rules to Preserve

- **Individual events**
  - Max 2 participants per district per event per seniority category (Junior/Senior inferred by DOB range or event name keywords).
  - Max 5 individual events per participant.
  - Exclude members marked in `KalamelaExcludeMembers`.
  - Chest numbers: assign deterministically with no duplicates; previously derived from DOB ranges (100/200/…); make it atomic with DB constraints.

- **Group events**
  - At most 2 teams per district per event.
  - Per-unit allowed limit enforced (`max_allowed_limit` and `per_unit_allowed_limit` decremented).
  - Prevent duplicate members in the same event; prefer same-unit grouping.
  - Chest number format: prefix from event initials, increment per team.

- **Payments**
  - Kalamela fees: 50 per individual participation, 100 per group team (per district); conference payments: 300 * (official_count + member_count).
  - Status transitions: Pending → Pending (Proof Uploaded) → Paid / Declined.

- **Appeals**
  - Must be filed within 30 minutes of score publication.
  - Appeal fee 1000; status flows through payment confirmation; after approval, update scorecards.

- **Results**
  - Top 3 per event (individual & group) ordered by points and timestamp.
  - Kalaprathibha/Kalathilakam: gender-based aggregate, min 2 events, highest total points.

---

11) Data Migration Plan
- Export current DB (PostgreSQL/SQLite) to CSV/SQL.
- Create Alembic migrations for new schema.
- Write one-time ETL scripts to load users/districts/units/members, events, participations, payments, scores, appeals.
- Normalize chest numbers and enforce unique constraints during import; report conflicts.

- Hash passwords anew (force reset) or invalidate and send reset links.

---

12) Testing Plan
- Unit tests for services (auth, participation rules, payments, appeals, scoring).
- Integration tests for critical flows:
  - Login, registration, token refresh.
  - Add/remove individual participants (limit checks, duplicate checks).
  - Add group participants (limits, chest numbering).
  - Payment creation, proof upload, admin approval.
  - Appeal submission timing and approval path.
  - Score ingest/update and results ordering.
  - Exports produce non-empty XLSX.

---

13) Implementation Notes per Django View/Helper (for parity)

- **auth_app/views.py**: `do_login`, `forgot_password`, `add_registered_user`, `get_unit_names`, admin export views, admin dashboards, unit/official/councilor/member views, print form.
- **auth_app/management**: `assign_chest_numbers` (DOB-based chest assignment), `backup_db` (engine-specific dumps), `reset_unit_login_password` (bulk predictable reset).
- **conference/views.py**: admin home, login/logout.
- **conference/admin_views.py**: conference CRUD; delegate official creation; update/delete district officials; view members; all_conference_info export; admin_payment_info export; payments aggregation.
- **conference/official_views.py**: view conference; add delegates; list delegates; remove; make payment; confirm payment (bug); export for district.
- **kalamela_auth_views.py**: public landing; login/logout; find participants; results (top 3); appeal form + submission; view appeals; kalaprathibha calculation.
- **kalamela/utility.py**: all helper logic for filtering members, enforcing limits, assigning chest numbers, building participant dicts.
- **kalamela/admin_utility.py**: admin-side participant views, exports (events data, chest numbers), results export, chest-number export.
- **kalamela_admin_views.py**: event CRUD; unit/member management; exclusions; participants listing; chest edits; preview; payments admin; exports; scoring add/update; appeals handling; unit/district-wise results.
- **kalamela_official_views.py**: official event selection/add/remove; previews; payments; print form.
- **kalamela_admin_views scoring**: bulk JSON ingest/update for scorecards; top-3 result pages.
- **kalamela_admin_views appeals**: approve sets reply/status and recomputes scorecards.

- **kalamela_official_payment_proof_upload**: uploads proof with custom filename; set status.

Use these summaries to re-create route handlers in FastAPI services with proper validation and security.