

**This is my FastAPI backend. I want a detailed analysis and report on this backend and all the available features and their functionality. Read, analyse and understand the files, structure and code within the files.**

This backend is a role-based HRM API built with FastAPI, SQLAlchemy, and JWT auth, featuring user registration/login, attendance logging, leave requests with approval flow, and a monthly “leave coin” balance system that grants, consumes, and expires coins over a 12‑month rolling window with a cap. It uses APScheduler for background jobs, Pydantic for schema validation/serialization, and environment-driven configuration for DB and secrets.[[1]](#fn1)[[2]](#fn2)[[3]](#fn3)[[4]](#fn4)[[5]](#fn5)[[6]](#fn6)

**Project overview**

* Core capabilities: authentication (JWT), user registration, employee records, attendance CRUD with RBAC, leave requests and admin approvals, leave coin balance with monthly grants, FIFO consumption, expiry, and admin/dev maintenance endpoints.[[3]](#fn3)[[5]](#fn5)[[7]](#fn7)[[6]](#fn6)[[8]](#fn8)
* Tech stack: FastAPI, SQLAlchemy ORM with MSSQL via pyodbc, Pydantic v2 serializers, jose for JWT, bcrypt for password hashing, APScheduler for scheduled jobs, zoneinfo for IST conversions, python-dotenv for config.[[2]](#fn2)[[9]](#fn9)[[10]](#fn10)[[1]](#fn1)[[3]](#fn3)

**Configuration and security**

* Env file provides MSSQL connection string and a long SECRET\_KEY, loaded via dotenv in auth and db modules; missing SECRET\_KEY raises at import.[[11]](#fn11)[[1]](#fn1)[[2]](#fn2)
* CORS is fully open (allow\_origins=["\*"], allow\_credentials=True), which simplifies development but is risky for production if cookies/credentials are involved.[[3]](#fn3)
* Passwords are stored as bcrypt hashes; verification uses bcrypt.checkpw; JWT uses HS256 with 30-minute access tokens and includes sub and role claims.[[1]](#fn1)[[3]](#fn3)

**Data model**

* User: id, username (unique, <=15), hashed\_password, role in {"super\_admin","admin","employee"} with default employee.[[4]](#fn4)
* Employee: id, name, user\_id unique and non-null; establishes one-to-one user-employee mapping.[[4]](#fn4)
* Attendance: employee\_id, login/logout timestamps, on\_leave flag, computed work\_hours float; basic indexing.[[4]](#fn4)
* LeaveRequest: employee\_id, start/end, type, status {"pending","approved","denied"}, reason; relationship to Employee.[[4]](#fn4)
* LeaveCoin: grants with grant\_date, expiry\_date (grant + 12 months), quantity, remaining, source, created\_at (UTC), index on (employee\_id, expiry\_date).[[4]](#fn4)
* LeaveCoinTxn: immutable ledger of grants, consume, expire, adjust, restore with occurred\_at and optional references to leave requests.[[4]](#fn4)

**API surface**

* Auth
  + POST /token: form-based login (OAuth2PasswordRequestForm), validates bcrypt hash, returns JWT with sub=username and role.[[3]](#fn3)
  + POST /register: creates a user as employee role and auto-creates Employee profile if absent.[[3]](#fn3)
* Employees
  + POST /employees: admin-only create; currently binds new Employee to current admin’s user\_id, which is likely a placeholder that should accept a target user\_id.[[12]](#fn12)
  + GET /employees: admin-only list with pagination.[[12]](#fn12)
  + GET /employees/{id}: self-access or admin; forbids others.[[12]](#fn12)
  + PUT /employees/{id}: admin-only update name.[[12]](#fn12)
  + DELETE /employees/{id}: admin-only delete.[[12]](#fn12)
* Attendance
  + POST /attendance/log: employee-only; logs own record; enforces ownership and logout\_time >= login\_time; sets work\_hours if both timestamps present.[[7]](#fn7)
  + GET /attendance: admins see all; employees see own records; pagination via skip/limit.[[7]](#fn7)
  + GET /attendance/{id}: access-controlled per ownership/admin; 404 if missing.[[7]](#fn7)
  + PUT /attendance/{id}: admin-only updates; recomputes work\_hours on change.[[7]](#fn7)
  + DELETE /attendance/{id}: admin-only delete.[[7]](#fn7)
* Leaves
  + POST /leaves: employee or admin; employees can only create for own employee\_id; returns created pending request.[[6]](#fn6)
  + POST /leaves/{id}/approve: admin-only; validates date order and computes inclusive day count; attempts to consume coins equal to days; on insufficient coins, rolls back and returns 400; else marks approved.[[6]](#fn6)
  + POST /leaves/{id}/deny: admin-only; idempotency guard if already denied.[[6]](#fn6)
  + GET /leaves: admins see all; employees see own; pagination via skip/limit.[[6]](#fn6)
  + GET /leaves/{id}: access-controlled per ownership/admin.[[6]](#fn6)
  + PUT /leaves/{id}: admin-only updates after submission; employees cannot update post submit.[[6]](#fn6)
  + DELETE /leaves/{id}: admin-only, 204 on success.[[6]](#fn6)
* Leave balance
  + GET /leave-balance/me: returns summary (available\_coins capped at 10, raw\_available uncapped within rolling window, expiring\_soon within 60 days, last 10 txns) for current user’s employee profile.[[5]](#fn5)[[8]](#fn8)
  + GET /leave-balance/employees/{employee\_id}: admin-only balance for a given employee.[[8]](#fn8)
* Dev/admin maintenance
  + POST /**dev**/grant-now: super\_admin-only, grants 1 coin to all employees immediately via service.[[3]](#fn3)
  + POST /**dev**/expire-now: super\_admin-only trigger expiration run.[[3]](#fn3)

**Auth and dependencies**

* OAuth2PasswordBearer configured with tokenUrl="token"; get\_current\_user decodes JWT, extracts sub and role, fetches DB user, and attaches role from token as authoritative, then returns the user model.[[13]](#fn13)
* RoleChecker provides Depends guards: allow\_admin, allow\_super\_admin, allow\_employee used on routes.[[13]](#fn13)
* Security note: trusting role from token rather than DB can allow privilege escalation if a compromised token contains elevated role; typically one would re-check role from database.[[13]](#fn13)

**Scheduling and background jobs**

* BackgroundScheduler started in FastAPI lifespan; three jobs configured: remove\_old\_attendance daily (older than 30 days), grant\_monthly\_coins on 1st of month at 00:00, and expire\_old\_coins daily; each uses independent DB sessions with proper commit/rollback.[[3]](#fn3)
* grant\_monthly\_coins enforces cap via service function and logs total granted; expire job tallies expirations.[[5]](#fn5)[[3]](#fn3)

**Leave coin service logic**

* Time handling uses helpers to ensure UTC-aware datetimes for comparisons and ledger events; DB comparisons use naive timestamps stripped to align with typical SQL datetime storage.[[5]](#fn5)
* Rolling window start computed as now - 12 months via dateutil.relativedelta with timedelta fallback; expiry is grant + 12 months.[[5]](#fn5)
* get\_available\_coins filters active, non-expired, remaining>0 within rolling 12 months and returns both raw sum and capped available (min with CAP\_COINS=10); expiring\_soon consolidates amounts expiring within next 60 days; recent\_txns returns last 10 with UTC-aware occurred\_at.[[5]](#fn5)
* grant\_coins enforces cap before granting; grants only up to remaining capacity; records LeaveCoin and a grant txn; returns granted amount.[[5]](#fn5)
* expire\_coins sets remaining to 0 for all coins with expiry\_date <= now; records expire txns; returns total expired.[[5]](#fn5)
* consume\_coins performs FIFO by expiry then id; deducts remaining until requested amount met; records consume txns; if insufficient, returns 0 without commit so caller can rollback partial deductions.[[5]](#fn5)

**Serialization and time zones**

* Pydantic v2 field\_serializer converts datetime fields to IST for API outputs in AttendanceOut and LeaveRequestOut; work\_hours rounded to 2 decimals, and a computed work\_duration renders “Xh Ym”.[[9]](#fn9)
* to\_ist ensures naive datetimes are treated as UTC before converting to IST; zoneinfo used instead of pytz.[[10]](#fn10)[[9]](#fn9)

**Database and session management**

* MSSQL URL from env using pyodbc and ODBC Driver 18; engine created with default settings; SessionLocal configured for dependency injection; Base.metadata.create\_all(bind=engine) is called at import time in main, creating tables automatically.[[11]](#fn11)[[2]](#fn2)[[3]](#fn3)
* Attendance purge job deletes records older than 30 days based on login\_time in UTC; consider aligning with IST if business rules require.[[3]](#fn3)

**Strengths**

* Clear separation of concerns: models, schemas, services, routers, dependencies, and auth modules.[[9]](#fn9)[[8]](#fn8)[[13]](#fn13)[[7]](#fn7)[[4]](#fn4)[[6]](#fn6)[[3]](#fn3)[[5]](#fn5)
* Robust leave balance domain with cap, rolling window, expiry, FIFO consumption, and immutable transaction ledger for auditability.[[4]](#fn4)[[5]](#fn5)
* Sound RBAC pattern and ownership checks on sensitive routes (attendance, leaves).[[7]](#fn7)[[12]](#fn12)[[6]](#fn6)
* Consistent timezone handling and API-facing IST conversion for better UX in India context.[[10]](#fn10)[[9]](#fn9)
* Scheduler encapsulated in lifespan with safe session handling and logging.[[3]](#fn3)

**Gaps and risks**

* Token role authority: role is taken from token rather than DB; if a token is forged or stolen, claims could be abused; safer to load role from DB and ignore role in token, or sign short-lived tokens and validate rotation.[[13]](#fn13)
* Open CORS with allow\_credentials=True is unsafe for production; restrict origins and set allow\_credentials judiciously.[[3]](#fn3)
* Employee creation binds to current admin user\_id, which likely isn’t intended; the route should accept a user\_id parameter with validation to create employees for target users.[[12]](#fn12)
* Base.metadata.create\_all in main tightly couples runtime with schema creation; consider Alembic migrations and avoid auto-create in production.[[3]](#fn3)
* Attendance purge deletes records older than 30 days; verify compliance and reporting needs before data removal; consider archiving instead.[[3]](#fn3)
* Error handling: some endpoints return generic 400/401/403; for UX, consider standardized error responses; for security, avoid leaking user existence on login (current code leaks “User is not registered”).[[3]](#fn3)
* JWT expiry/refresh: only access tokens are implemented; consider refresh tokens or re-login flows; also consider token revocation on password change.[[1]](#fn1)[[3]](#fn3)
* Database timezone consistency: code mixes aware and naive by stripping tzinfo for DB filters; ensure DB stores UTC consistently and columns are timezone-agnostic or normalized to UTC.[[5]](#fn5)
* Concurrency: grant/consume operations rely on transactional integrity without explicit row-level locks; under high concurrency, consider SELECT ... FOR UPDATE patterns to avoid over-consumption.[[5]](#fn5)

**Suggested improvements**

* Security hardening
  + Validate role from DB in get\_current\_user and ignore role claim in token; or sign tokens with minimal claims and re-fetch role per request.[[13]](#fn13)
  + Lock down CORS to known origins and set allow\_credentials=False unless cookies are in use; prefer Authorization header tokens.[[3]](#fn3)
  + Unify login error message to avoid user enumeration; e.g., “Invalid credentials.”[[3]](#fn3)
* API design
  + employees POST: accept and validate a target user\_id; enforce one-to-one invariant at API level.[[12]](#fn12)
  + Add GET /users/me (already present in dependencies router) to fetch current profile; consider a dedicated users router for clarity.[[13]](#fn13)
  + Add pagination metadata (total count) and consistent default limits.[[7]](#fn7)[[12]](#fn12)[[6]](#fn6)
* Observability
  + Add structured logging for critical flows (login, leave approvals, coin grants/consumes) and correlation IDs.[[3]](#fn3)
  + Include audit fields (created\_at/updated\_at) on core tables like Attendance and LeaveRequest.[[4]](#fn4)
* Data and migrations
  + Remove Base.metadata.create\_all from main; introduce Alembic with autogenerate for schema evolution.[[3]](#fn3)
  + Consider unique indexes and constraints where appropriate (e.g., username length enforced at app level but not validated in schema).[[4]](#fn4)
* Business logic
  + Attendance: prevent overlapping sessions per employee and auto-calc work\_hours server-side using now when logout is omitted; optionally support clock-in/out endpoints.[[7]](#fn7)
  + Leaves: handle half-days/holidays, weekend skipping, and per-type coin costs if business requires; add state transitions auditing.[[6]](#fn6)
  + Coins: provide an admin adjustment API (adjust/restore) building on LeaveCoinTxn types already modeled.[[4]](#fn4)[[5]](#fn5)
* Performance and safety
  + Add indices for frequent filters (Attendance.employee\_id, LeaveRequest.employee\_id, created\_at fields).[[4]](#fn4)
  + Validate and sanitize inputs with Pydantic constraints (username length, password min length, leave\_type enum).[[9]](#fn9)
  + Wrap critical updates in explicit transactions with repeatable read/serializable isolation or row locks in MSSQL for financial-like coin operations.[[5]](#fn5)

**Developer UX**

* Comprehensive OpenAPI via FastAPI defaults; consider tags and descriptions for each route and models for better docs; module routers already set tags.[[8]](#fn8)[[12]](#fn12)[[7]](#fn7)[[6]](#fn6)
* Add a /test-db-connection endpoint (commented out) gated to admins or health checks with no secrets; use SQLAlchemy text and handle exceptions cleanly.[[3]](#fn3)
* Provide seed/admin bootstrap script to create a super\_admin securely and initial employees for local testing.[[4]](#fn4)[[3]](#fn3)

**Notable implementation details**

* Inclusive day count for leave approval uses (end\_date - start\_date).days + 1, which matches typical HR rules; rejects negative or zero-day spans and inverted ranges.[[6]](#fn6)
* Attendance work\_hours is recomputed on update to ensure consistency after edits.[[7]](#fn7)
* The leave balance endpoint provides both a hard cap view and raw availability with “expiring soon” within 60 days, plus recent transactions, enabling a solid UX for planning leaves.[[8]](#fn8)[[5]](#fn5)
* Dev endpoints require super\_admin token and let operators trigger grant/expire logic on demand; useful for testing monthly jobs.[[3]](#fn3)

⁂

1. [auth.py](http://auth.py)

1. [db.py](http://db.py)

1. [main.py](http://main.py)

1. [models.py](http://models.py)

1. leave\_coins.py

1. [leave.py](http://leave.py)

1. [attendance.py](http://attendance.py)

1. leave\_coin.py

1. [schemas.py](http://schemas.py)

1. [utils.py](http://utils.py)

1. file.env

1. [employees.py](http://employees.py)

1. [dependencies.py](http://dependencies.py)