Goal :

Needs to have an admin dashboard using next js (ts),Redis(or local cache or adapter to extend), Cockroach db(or postgres depending on user configuration) for below features.

1. Authentication (admin login, users login)
2. Token Management (JWKS -> jwt token with roles and custom claims)
3. Admin Dashboard features
    a. App setup
    b. Role management
    c. User management
    d. Custom meta claims
    e. SQL Scripts to setup/clean

Desc:

For admin login to admin dashboard, for others redirect to relevant redirect endpoint along with jwt token

Setup/update Constants :

- Envs : tenant_id, app_id, admin_email, cache_host, cache_user, cache_password, cache_type, jwt_private_key, smtp_keys, db_host, db_user, db_password, db_type, db_name if any additional needed
- Note : Needs to store secrets securely without exposing to outside

Authentication:

- Will have 2 authentication mechanisms
    1. /admin -> One for admin login -> email, password (sends an 6 digit otp using smtp settings to login)
        a. If successful login -> redirect to admin dashboard
    2. /login -> One for letting users to login (email otp login -> smtp configured via admin dashboard and stored in db)
        a. If successful login redirect to configured endpoint(only part of vaild_redirect_urls) in admin dashboard along with jwt in header
            i. Issue token, refresh token, logout, verify_token(checks in cache)

Token Management:

- JWT schema- (tenant_id, app_id, is_owner, roles = [‘’] ,additional_claims: {custom_claims_map})
- Endpoints to issue jwt(admin, user), renew jwt(admin, user) - takes valid jwt and generates new one wrt roles and stores in cache, logout(admin, user)
- Tokens stored in redis cache
- Exposes /.well-known/jwks.json -> so other services can verify using these public keys irrespective of language

Admin Dashboard features:

App Setup:

- edit/update App_name, app_description, vaild_redirect_urls (www.localhost.com, something else -> regex mapping), smtp_rate_limit_per_user, smtp_rate_limit_time_frame, supported_login_domain_list(may be gmail.com , apple.com, etc), well_known_jkws-> app_meta table
- Any additional required/useful details

Role Management(Admin dashboard):

- Able to CRUD roles (roles table)
- Able to CRUD permissions (scopes table)
- Able to tag roles to signed in users (user_roles table - crud needed)
- According to the roles tagged jwt token needs to be created

Users Management:

- Provides an interface to check the total users, users logged in last x days, users change graph from the day created
- Provides an interface to block users(will not issue token for blocked users), unblock users, invite users(sends an email)
- filter users by different roles, etc
- CRUD Custom meta claims from UI as well
- Any details required/useful details

Custom meta claims:

- The idea behind this feature is that the admin can expose an api which lets other users/apps to set claims using an api(protected route)
    - This route is protected by access_key has to be set as part of header
    - This access keys will be generated by admin using this feature - has fields like tenant_id,app_id, source, exp, etc needed
    - Each and every access key also has an expiry(till when it's valid or no expiry) (stored in custom_claims table)
    - The input for this custom_claims which gets stored in db and used while issuing a token)
    - CRUD Needed for access keys generation

Note : Code Practices:

- Have to implement code in extendable fashion, not hardcoded
- Everything has to be modular
- Have to be free from bugs
- Needs to have unit tests for backend
- API Specs/Swagger needed
- Working application needed
- Proper Naming conventions/file structure
- Sql scripts to setup/clean create tables/index if any needed (namespace or schema -> tenant_id+""+*app*id)
- Needs to implement all the security features or measures like rate_limiting api access,etc
- Should not expose env secrets to frontend unless it required