# Repository Analyzer – Technical Documentation

## Overview
This system ingests GitHub repository activity (issues, PRs, discussions), stores it in PostgreSQL, and provides a UI for users to explore data. It supports OAuth-based per-user rate limits for GitHub API calls.

## High-Level Design (HLD)
- Ingestion Layer: Cursor-based GraphQL ingestion with crash-safe checkpoints.
- Database: PostgreSQL storage for raw payloads and cursor state.
- Web UI: Next.js app with OAuth login and repository analytics view.
- OAuth: GitHub OAuth to use each user's API token.

## Data Flow
1. User signs in with GitHub OAuth.
2. User submits a repo.
3. Ingestion runs using their token.
4. Issues/PRs/Discussions stored in DB.
5. UI queries DB and renders results.

## Database Schema

### ingestion_cursors
Tracks pagination progress per repo/entity.

### github_issues / github_pull_requests / github_discussions
Stores raw GraphQL payloads for all entities.

### github_oauth_tokens
Stores user tokens and GitHub login.

### github_sessions
Stores active sessions.

## Key Modules

### packages/github
- Reads token from env.
- Creates auth headers.

### packages/db
- Cursor repository
- Issue/PR/Discussion repositories
- Migrations

### packages/ingest
- Incremental sync engine
- Pagination and cursor logic
- Rate limit handling

### apps/web
- OAuth endpoints
- Ingestion API endpoint
- UI to display collected data

## Sequence Diagram (Auth + Ingestion)
```mermaid
sequenceDiagram
  autonumber
  participant U as "User"
  participant UI as "Web UI"
  participant GH as "GitHub OAuth"
  participant API as "Next API"
  participant DB as "PostgreSQL"
  participant GQL as "GitHub GraphQL"

  U->>UI: Open app
  UI->>API: GET /api/me
  API->>DB: Load session
  DB-->>API: Session or null
  API-->>UI: Auth state

  U->>UI: Click "Sign in with GitHub"
  UI->>GH: Redirect to GitHub OAuth
  GH-->>API: /api/auth/github/callback?code=...
  API->>GH: Exchange code for token
  GH-->>API: Access token
  API->>GH: Fetch user profile
  GH-->>API: GitHub user
  API->>DB: Store token + session
  API-->>UI: Redirect to /

  U->>UI: Submit repo URL
  UI->>API: POST /api/ingest {owner, name}
  API->>DB: Load session + token
  API->>GQL: GraphQL Issues (cursor)
  GQL-->>API: Issues page
  API->>DB: Upsert issues + cursor
  API->>GQL: GraphQL PRs (cursor)
  GQL-->>API: PRs page
  API->>DB: Upsert PRs + cursor
  API->>GQL: GraphQL Discussions (cursor)
  GQL-->>API: Discussions page
  API->>DB: Upsert discussions + cursor
  API-->>UI: {repoId}
```

## ER Diagram
```mermaid
erDiagram
  ingestion_cursors {
    text repo_id
    text entity_type
    text cursor
    timestamptz last_synced_at
    timestamptz created_at
    timestamptz updated_at
  }

  github_issues {
    text repo_id
    text issue_id
    int issue_number
    text title
    text state
    text url
    text author_login
    timestamptz created_at
    timestamptz updated_at
    jsonb raw_payload
  }

  github_pull_requests {
    text repo_id
    text pull_request_id
    int pull_request_number
    text title
    text state
    text url
    text author_login
    timestamptz created_at
    timestamptz updated_at
    jsonb raw_payload
  }

  github_discussions {
    text repo_id
    text discussion_id
    int discussion_number
    text title
    text state
    text url
    text author_login
    timestamptz created_at
    timestamptz updated_at
    jsonb raw_payload
  }

  github_oauth_tokens {
    bigint github_user_id
    text login
    text access_token
    timestamptz created_at
    timestamptz updated_at
  }

  github_sessions {
    text session_id
    bigint github_user_id
    timestamptz created_at
    timestamptz expires_at
  }

  ingestion_cursors ||--o{ github_issues : "repo_id + entity_type"
  ingestion_cursors ||--o{ github_pull_requests : "repo_id + entity_type"
  ingestion_cursors ||--o{ github_discussions : "repo_id + entity_type"

  github_oauth_tokens ||--o{ github_sessions : "github_user_id"
```

## Deployment Checklist (Render + Neon)
1. Create Neon DB and copy DATABASE_URL.

2. Run migrations locally against Neon:
   DATABASE_URL="your_neon_url" node packages/db/dist/migrate.js
3. Push code to GitHub.
4. Create Render Web Service.
5. Root Directory: .
6. Build Command: pnpm install --prod=false && pnpm -r build
7. Start Command: pnpm --filter @app/web start
8. Env Vars: DATABASE_URL, GITHUB_CLIENT_ID, GITHUB_CLIENT_SECRET, APP_BASE_URL
9. Update GitHub OAuth callback URL to Render URL.
10. Redeploy and test.

## Notes
- All GitHub calls are made using the signed-in user’s token.
- Cursor sync ensures minimal API usage on repeat requests.
- Raw payloads are stored for future AI analysis.