COMPLETION CRITERIA

At the end of this implementation, we must have:
- Fully working APIs
- A frontend that is entirely data-driven
- Admin updates reflected immediately on the site
- A stable foundation for future feature development

STAGE 1 – FOUNDATION & CONTRACTS

BACKEND
- Django project initialized
- Required apps created
- Django Ninja integrated
- Swagger documentation accessible
- Environment variables configured and working

FRONTEND
- Next.js application initialized
- Folder structure finalized
- API base configuration set up
- Placeholder pages created (no UI design)

SHARED
- API routes agreed upon
- Response field names agreed upon

STAGE 2 – CORE MODELS & ADMIN

BACKEND
Create core models such as:
- Member
- Project / Contribution
- Home page configuration

- Register all models in Django Admin
- Admins must be able to manage all content without developer assistance

FRONTEND
- Prepare API utility functions
- Create mock data using the agreed schema
- Implement loading and error-handling states

SHARED
- Once schemas are finalized, avoid arbitrary renaming of fields

STAGE 3 – PUBLIC READ APIS

BACKEND
Implement the following endpoints:
- Members list
- Member detail
- Projects list
- Home page data

- Ensure responses are clean, consistent, and well-structured JSON

FRONTEND
- Replace mock data with real API calls
- Verify that all pages render correctly
- Handle empty and edge states safely

STAGE 4 – MEDIA HANDLING (SIMPLE VERSION)

BACKEND
- Use local Django media storage
- Enable image uploads via Admin
- APIs should return accessible image URLs

FRONTEND
- Render images from API responses
- Provide fallback UI when images are missing

STAGE 5 – CONTACT FLOW

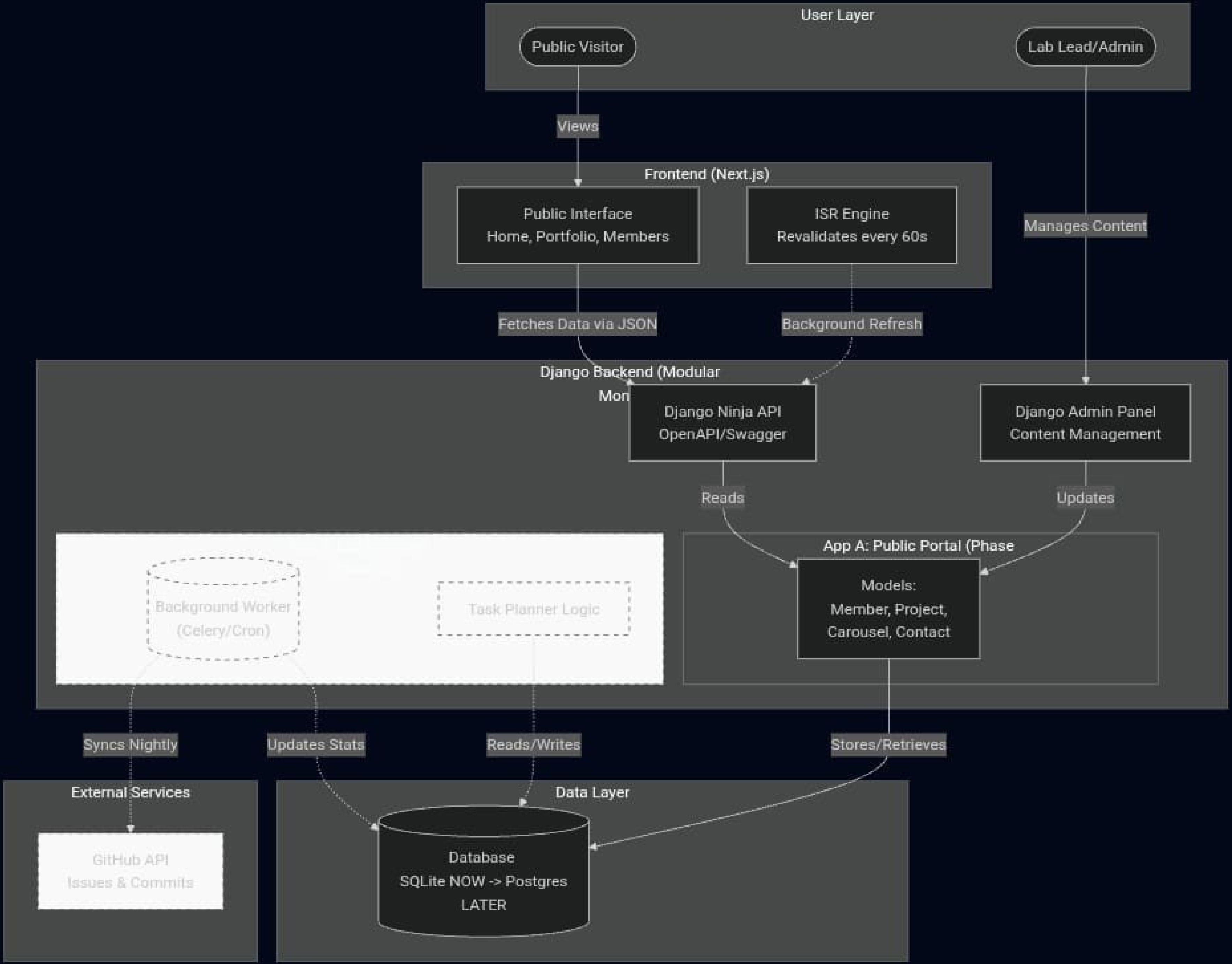
BACKEND
- Create POST endpoint
- Validate input data
- Store submissions or trigger email notifications

FRONTEND
- Connect the contact form to the API
- Implement success and error states
- Prevent duplicate submissions

STAGE 6 – STABILITY PASS

BACKEND
- Improve validation logic
- Review permissions
- Clean up and standardize code

FRONTEND
- Remove all leftover mock data
- Eliminate hardcoded values
- Handle API failures gracefully


User Layer
Public Visitor
Lab Lead/Admin
Views
Manages Content
Frontend (Next.js)
Public Interface
Home, Portfolio, Members
ISR Engine
Revalidates every 60s
Fetches Data via JSON
Background Refresh
Django Backend (Modular
Mon
Django Ninja API
OpenAPI/Swagger
Django Admin Panel
Content Management
Reads
Updates
App A: Public Portal (Phase
Models:
Member, Project,
Carousel, Contact
Background Worker
(Celery/Cron)
Task Planner Logic
Syncs Nightly
Updates Stats
Reads/Writes
Stores/Retrieves
External Services
GitHub API
Issues & Commits
Data Layer
Database
SQLite NOW -> Postgres
LATER