🚀 Nardaa: Project Overview & Architecture

1. Product Vision

Nardaa is a "Developer-First" Transactional Email Service (SaaS). It solves the friction developers face when trying to send emails from their local development environments (`localhost`).

The Goal: Eliminate the need for developers to configure SMTP servers or use personal Gmail credentials during development. We provide a secure API and a "Sandbox" dashboard to view emails without actually spamming real users.

2. The Core Problem vs. Our Solution

The Problem (Current State)	The Nardaa Solution
Security Risk: Developers often hardcode Gmail App Passwords or SMTP credentials in `.env` files.	**Secure API:** Developers use a revocable `API_KEY` associated with a project.
Messy Code: HTML email templates are often hardcoded strings inside backend logic.	**Dynamic Templates:** Templates are created visually in our Dashboard and referenced by ID (e.g., `WELCOME_MAIL`).
Testing Pain: Sending test emails requires a real recipient, leading to accidental spamming of real users.	**Sandbox Mode:** A "Trap" system that captures emails sent from localhost and displays them in our UI, ensuring no real email leaves the system during dev.
Setup Fatigue: Setting up Nodemailer/Postfix takes time.	**Instant Setup:** `npm install nardaa` -> Paste API Key -> Send.

3. How It Works (The "Push" Workflow)

We utilize a **Push-based Architecture**. The developer's local application pushes a JSON payload to our cloud API.

The User Journey

1. **Setup:** Developer creates a Project on Nardaa (e.g., "E-commerce Dev").
2. **Template:** Developer designs a generic email in our UI:
   - *Subject:* `Order #{{orderId}} Confirmed`
   - *Body:* `Hi {{name}}, thanks for buying {{product}}!`
3. **Trigger (Localhost):** The developer writes code in their local app:

```
// In their Node/Python/Go app
nardaa.send({
  key: "nk_live_xyz",
  template: "ORDER_CONFIRM",
  data: { orderId: 101, name: "Lara", product: "Laptop" }
});
```

4. **Processing:**
    - Nardaa accepts the request instantly.
    - Nardaa puts the job in a **Queue** (Redis).
    - **Worker Node** picks up the job, injects the data into the HTML template.
5. **Delivery:**
    - If **Sandbox Mode** is ON: Save to DB, show in Dashboard "Inbox".
    - If **Live Mode** is ON: Relay to AWS SES/SMTP provider to deliver to the actual email address.

4. System Architecture

We are building a decoupled system to ensure high performance and non-blocking API responses.

A. Frontend (The Dashboard)

- **Tech:** Next.js (React), Tailwind CSS.
- **Purpose:**
    - User Authentication (Google Login).
    - Project Management (Create/Delete Projects).
    - **Template Editor:** A simple editor to write HTML/Text with `{{variable}}` syntax.
    - **Live Logs:** A real-time list of emails triggered by their localhost.

B. Backend (The API Gateway)

- **Tech:** Node.js (Express).
- **Purpose:**
    - Validates API Keys.
    - Validates payload data against the requested Template.
    - **Crucial:** Does NOT send email immediately. It pushes the task to Redis. This ensures the user's API call returns `200 OK` in <100ms.

C. The Worker Engine (Background Service)

- **Tech:** Node.js + BullMQ + Redis.
- **Purpose:**
    - Watches the Queue for new email jobs.
    - Renders the HTML (replaces `{{name}}` with actual data).
    - Decides logic: **Sandbox** (save to DB) vs **Live** (send via external SMTP).
    - Updates the "Log" status to `DELIVERED` or `FAILED` .

D. Database & Auth

- **Tech:** Firebase (Firestore & Auth).
- **Purpose:**
    - **Auth:** Handles Google Sign-in instantly.

  - **Firestore:** Stores User Profiles, Projects, Templates, and Email Logs.
  - *Why Firestore?* It is real-time. When a developer sends an email from localhost, the "Log" on their dashboard will appear instantly without refreshing the page.

5. Conceptual Data Model

We need to structure our NoSQL database to handle multi-tenancy (Users having multiple projects).

- **Users:** `uid`, `email`, `name`
- **Projects:** `id`, `owner_uid`, `api_key`, `mode` (sandbox/live)
- **Templates:** `id`, `project_id`, `html_content`, `required_variables` (array)
- **Logs:** `id`, `project_id`, `template_id`, `request_payload` (json), `status` (sent/bounced), `timestamp`

6. MVP Roadmap (Phase 1)

Sprint 1: The Core

- [ ] Setup Firebase Auth & Firestore.
- [ ] Create "Project" & Generate API Key logic.
- [ ] Build "Log Ingestion" API (Receive JSON, save to DB).

Sprint 2: The Queue & Engine

- [ ] Setup Redis.
- [ ] Implement BullMQ to process incoming API requests.
- [ ] Build the "Sandbox" logic (Just save to DB, don't send).

Sprint 3: The Dashboard

- [ ] View Logs Table (Real-time).
- [ ] Template Creator (Simple text area).
- [ ] Documentation Page (How to use the API).

7. Technical Stack Summary

Component	Technology	Reasoning
Frontend	Next.js 14	SEO friendly, easy routing, server components.
Backend API	Node.js (Express)	Fast I/O, vast ecosystem for libraries.
Database	Firebase Firestore	Schema-less, real-time updates for logs.
Auth	Firebase Auth	Secure, handles Google/Social logins easily.
Queue	Redis + BullMQ	Essential for separating "Accepting Request" from "Sending Email".
Styling	Tailwind CSS	Rapid UI development.