Big picture (what you're actually building)

ARCADE is **not** "just a notes website".
It's a **guided academic + career preparation platform**, where:

- content is **verified** (faculty involved)
- learning is **directed** (roadmaps, not random PDFs)
- career prep is **diagnostic** (skill gap, role-based guidance)

Think of it as:
📚 *Google Drive* + 🧭 *roadmap visualizer* + 🧠 *basic career advisor*
—but limited to a university ecosystem.

Good scope. Defensible. Review-safe.

---

1. Users of the system (very important)

You actually have **4 user roles**, not 2.

1. **Student (primary user)**
   - View notes
   - Upload own notes (subject to approval)
   - Follow roadmaps
   - Select skills & roles
   - See suggestions
   - View sample resumes
   - Access external resources
2. **Faculty (teacher)**
   - Upload subject materials
   - Approve / reject student-shared notes
   - Suggest or validate roadmaps
   - Monitor student engagement (basic level)
3. **HOD (conditional authority)**
   - Only involved **if required**
   - Final approval for sensitive or widely shared student notes
   - Oversight role, not daily usage
4. **Admin (system role)**

- Manage users
- Assign roles
- Control visibility
- Upload verified resumes
- Maintain external resources list

This role separation alone makes your project look **mature**.

---

2. Frontend modules (what the UI will have)

Let's break it cleanly.

A. Authentication & Profile Module

- Login / Register
- Role-based access (student / faculty / admin / HOD)
- Student profile: skills selected, roadmap progress
- Faculty profile: subjects handled

Tech:
Next.js (React) → perfect choice

B. Notes Sharing Module

This is your backbone.

Features:

- Subject-wise notes listing
- Faculty-uploaded notes (direct publish)
- Student-uploaded notes (go to approval queue)
- Approval flow:
    - Faculty → optional HOD → publish
- Access control (only relevant students)

UI ideas:

- Simple cards
- Filters by subject, semester
- "Verified by faculty" tag

C. Roadmap Module (pictorial one)

This is where you stand out.

Features:

- Visual roadmaps (flowchart / step-based)
- Goal-based:
    - Exam prep
    - Placement prep
    - Skill learning
- Progress tracking (manual checkmarks)

Frontend:

- Static SVG / diagram-based roadmap initially
- No need for complex animations (keep it safe)

D. Skill & Role Selection Program

This is your **logic-heavy but simple module**.

Flow:

1. Student selects technologies (checkbox list)
    - e.g., Python, React, SQL
2. Student selects target role
    - e.g., Frontend Dev, Data Analyst
3. System compares:
    - selected skills vs required skills for role
4. Output:
    - Missing technologies
    - Suggested improvements
    - Extra tools to learn

No ML needed here.
This is **rule-based logic** (and that's GOOD).

E. Guidance & Recommendation Module

- Shows remaining skills
- Shows suggested learning order
- Links to roadmap or external resources
- Simple text-based guidance

This avoids risky "AI hallucination" territory.

F. Resume Repository Module

- Around **25 verified sample resumes**
- Uploaded only by admin
- Categorized by role/domain
- Read-only for students

Important:
These are **examples**, not personal uploads → privacy-safe.

G. Resources & Practice Links Module

- Competitive programming sites
- Practice platforms
- Extra materials
- Filter by domain

Admin-managed list.

---

3. Backend schema (conceptual, human-readable)

I'll keep this *review-friendly*, not DB-nerdy.

Core Tables / Collections

- **User**
    - user_id
    - name
    - email
    - role (student/faculty/HOD/admin)

- **Subject**
  - subject_id
  - subject_name
  - faculty_id
- **Notes**
  - note_id
  - subject_id
  - uploaded_by
  - file_path
  - approval_status
  - approved_by
- **Roadmap**
  - roadmap_id
  - goal_type
  - steps (stored as structured data)
- **Student_Skills**
  - student_id
  - selected_skills
  - target_role
- **Role_Requirements**
  - role_name
  - required_skills
- **Resume_Samples**
  - resume_id
  - domain
  - file_path
- **Resources**
  - resource_id
  - category
  - link
  - description

This schema is **simple, scalable, and explainable**.

---

4. Suggested tech stack (no bullshit)

Frontend

- **Next.js (React)**
  SEO-ready, routing built-in, Vercel-friendly

Backend

Pick **one**, don't mix:

- **Node.js + Express** (best fit with Next.js)
- OR Django (only if you want Python everywhere)

I'd say: **Node.js wins here**.

Database

- **PostgreSQL** → structured data, relationships
- MongoDB also okay, but PostgreSQL looks better in reviews

ML (optional, future)

- Python
- NLP only for resume parsing later
- Not mandatory in current scope (and that's fine)

Version control

- GitHub (mandatory, obviously)

---

5. Traffic & scale (faculty loves this part)

Expected users:

- Students: 200–1000 (college-level)
- Faculty: 10–50
- HOD/Admin: very few

Traffic type:

- Mostly **read-heavy**
- Low write frequency
- No real-time operations

Conclusion:
👉 **Very manageable load**

---

6. Is Vercel deployment fine?

Short answer: **YES. Perfectly fine.**

Long answer:

- Frontend on **Vercel** → excellent
- Backend:
  - Vercel Serverless OR
  - Separate Node backend on Render / Railway
- Database:
  - Cloud PostgreSQL (Railway / Supabase)

For SGP:

- This setup is **more than enough**
- No infra drama
- Easy demo during review

Final honest verdict

What you've designed is:

- realistic
- not overambitious
- academically solid
- future-expandable
- resume-worthy

Most SGPs fail because they try to be startups.
Yours succeeds because it tries to be **useful inside a university**.

This project is absolutely under control.