MASTER INDEX — ALO Education Website + CRM (GoStudyIn-style)

Use this as your copy-paste checklist in GitHub. Create each file with the exact path and paste the content from the sections below (already in this canvas).

Copy Order (recommended)

1. `package.json`
2. `prisma/schema.prisma`
3. `prisma/seed.js`
4. `app/lib/prisma.js`
5. `app/lib/authOptions.js`
6. API routes
7. `app/api/auth/[...nextauth]/route.js`
8. `app/api/students/route.js`
9. `app/api/students/[id]/route.js`
10. `app/api/students/assign/route.js`
11. `app/api/courses/search/route.js`
12. `app/api/profile/route.js`
13. `app/api/match/route.js`
14. `app/api/applications/route.js`
15. `app/api/upload/compliance/route.js`
16. `app/api/comms/email/route.js`
17. `app/api/comms/sms/route.js`
18. `app/api/comms/whatsapp/route.js`
19. `app/api/facebook/lead/route.js`
20. Public pages
21. `app/layout.js`
22. `app/page.js`
23. `app/courses/page.js`
24. `app/blog/page.js`
25. `app/contact/page.js`
26. `app/apply/page.js`
27. `app/finder/page.js`
28. `app/profile/page.js`
29. `app/match/page.js`
30. CRM pages
31. `app/login/page.js`
32. `app/dashboard/page.js`
33. `app/students/page.js`
34. `app/students/new/page.js`
35. `app/students/[id]/page.js`

36. `app/templates/page.js`

After Commit — One-time commands in Vercel

- Set env vars (DATABASE_URL, NEXTAUTH_SECRET, NEXTAUTH_URL, BLOB_READ_WRITE_TOKEN, SMTP_, *TWILIO_*, FB_*)
- Redeploy
- Deployments → **Execute Command**:

```
npx prisma db push && node prisma/seed.js
```

Below is a **complete, GitHub-ready Next.js 14 (App Router) website + CRM** that mirrors the GoStudyIn flow, branded for **ALO Education**. It's all **plain JavaScript** (no TypeScript), so it builds cleanly on Vercel.

> How to use: In your repo, create each file/folder exactly as shown. After commit, go to Vercel → Redeploy. Then run the seed command shown at the bottom.

---

1) `package.json`

```
{
  "name": "aloeducation-crm",
  "version": "1.0.0",
  "private": true,
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "postinstall": "prisma generate || true"
  },
  "dependencies": {
    "next": "14.0.4",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "next-auth": "^4.24.7",
    "@prisma/client": "^5.9.0",
    "bcryptjs": "^2.4.3"
  },
  "devDependencies": {
    "prisma": "^5.9.0"
  }
}
```

Keep your Vercel env vars: `DATABASE_URL`, `NEXTAUTH_SECRET`, `NEXTAUTH_URL=https://aloeducation.org`.

---

2) Prisma (database)

`prisma/schema.prisma`

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
  id        String    @id @default(cuid())
  email     String    @unique
  password  String
  role      String    // "ADMIN" | "COUNSELOR" | "COMPLIANCE" | "FINANCE"
  students  Student[] @relation("CounselorStudents")
  createdAt DateTime  @default(now())
}

model Student {
  id          String   @id @default(cuid())
  name        String
  email       String
  phone       String
  stage       String   // "LEAD" | "OFFER" | "CAS" | "VISA" | "ENROLLED"
  counselor   User?    @relation("CounselorStudents", fields: [counselorId],
references: [id])
  counselorId String?
  compliance  Boolean  @default(false)
  commission  Float?
  notes       String?  @db.Text
  createdAt   DateTime @default(now())
}

model Template {
  id        String   @id @default(cuid())
  type      String   // "EMAIL" | "SMS" | "WHATSAPP"
  name      String
  body      String   @db.Text
```

```
  createdAt DateTime @default(now())
}
```

prisma/seed.js

```
import { PrismaClient } from "@prisma/client";
import bcrypt from "bcryptjs";

const prisma = new PrismaClient();

async function main() {
  // Default users
  const users = [
    { email: "admin@aloeducation.org", password: "admin123", role: "ADMIN" },
    { email: "counselor@aloeducation.org", password: "counselor123", role:
"COUNSELOR" },
    { email: "compliance@aloeducation.org", password: "compliance123", role:
"COMPLIANCE" }
  ];

  for (const u of users) {
    const hashed = bcrypt.hashSync(u.password, 10);
    await prisma.user.upsert({
      where: { email: u.email },
      update: {},
      create: { email: u.email, password: hashed, role: u.role },
    });
  }

  // Some starter templates
  await prisma.template.createMany({
    data: [
      { type: "EMAIL", name: "Welcome Lead", body:
"Hi {{name}}, thanks for your interest in ALO Education." },
      { type: "SMS", name: "Lead Follow-up", body: "Hi {{name}}, we received
your inquiry. A counselor will contact you soon." },
      { type: "WHATSAPP", name: "Document Reminder", body: "Hello {{name}},
please upload your CAS documents to proceed." }
    ],
    skipDuplicates: true
  });

  console.log("✅ Seed complete: users + templates");
}
```

```
main().then(() => prisma.$disconnect()).catch((e) => { console.error(e); prisma.
$disconnect(); process.exit(1); });
```

---

3) Shared utilities

`app/lib/prisma.js`

```
import { PrismaClient } from "@prisma/client";

// Ensure single Prisma instance in dev
let prisma;
if (process.env.NODE_ENV === "production") {
  prisma = new PrismaClient();
} else {
  if (!global.prisma) global.prisma = new PrismaClient();
  prisma = global.prisma;
}

export default prisma;
```

`app/lib/authOptions.js`

```
import CredentialsProvider from "next-auth/providers/credentials";
import prisma from "../lib/prisma";
import bcrypt from "bcryptjs";

const authOptions = {
  providers: [
    CredentialsProvider({
      name: "Credentials",
      credentials: {
        email: { label: "Email", type: "text" },
        password: { label: "Password", type: "password" },
      },
      async authorize(credentials) {
        if (!credentials?.email || !credentials?.password) return null;
        const user = await prisma.user.findUnique({ where: { email:
credentials.email } });
        if (!user) return null;
        const ok = bcrypt.compareSync(credentials.password, user.password);
        if (!ok) return null;
        return { id: user.id, email: user.email, role: user.role };
      },
```

```
    }),
  ],
  callbacks: {
    async jwt({ token, user }) {
      if (user) {
        token.id = user.id;
        token.role = user.role;
      }
      return token;
    },
    async session({ session, token }) {
      if (token) {
        session.user.id = token.id;
        session.user.role = token.role;
      }
      return session;
    },
  },
  pages: { signIn: "/login" },
  secret: process.env.NEXTAUTH_SECRET,
};

export default authOptions;
```

---

4) Auth routes (NextAuth v4 on App Router)

`app/api/auth/[...nextauth]/route.js`

```
import NextAuth from "next-auth";
import authOptions from "@/app/lib/authOptions";

const handler = NextAuth(authOptions);
export { handler as GET, handler as POST };
```

---

5) Students API (CRUD + simple assign)

`app/api/students/route.js` **(list + create)**

```
import prisma from "@/app/lib/prisma";

export async function GET() {
```

```
  const students = await prisma.student.findMany({ include: { counselor:
true }, orderBy: { createdAt: "desc" } });
  return Response.json(students);
}

export async function POST(req) {
  const body = await req.json();
  const { name, email, phone } = body;
  const s = await prisma.student.create({ data: { name, email, phone, stage:
"LEAD" } });
  return Response.json(s, { status: 201 });
}
```

`app/api/students/[id]/route.js` **(read/update/delete)**

```
import prisma from "@/app/lib/prisma";

export async function GET(_, { params }) {
  const s = await prisma.student.findUnique({ where: { id: params.id },
include: { counselor: true } });
  if (!s) return new Response("Not found", { status: 404 });
  return Response.json(s);
}

export async function PUT(req, { params }) {
  const body = await req.json();
  const s = await prisma.student.update({ where: { id: params.id }, data:
body });
  return Response.json(s);
}

export async function DELETE(_, { params }) {
  await prisma.student.delete({ where: { id: params.id } });
  return new Response(null, { status: 204 });
}
```

`app/api/students/assign/route.js` **(assign counselor)**

```
import prisma from "@/app/lib/prisma";

export async function POST(req) {
  const { studentId, counselorId } = await req.json();
  const s = await prisma.student.update({
    where: { id: studentId },
    data: { counselorId },
```

```
    include: { counselor: true },
  });
  return Response.json(s);
}
```

---

6) Public Website (GoStudyIn-style pages)

`app/layout.js` **(brand shell)**

```
export const metadata = {
  title: "ALO Education — Study Abroad",
  description: "Your Dream — Our Commitment",
};

export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body style={{ margin: 0, fontFamily: "Arial, sans-serif", background:
"#fff" }}>
        <header style={{ background: "#0071BC", color: "#fff", padding: "12px
20px", display: "flex", alignItems: "center", justifyContent: "space-between" }}
>
          <div style={{ fontWeight: 800, letterSpacing: 0.5 }}>ALO <span
style={{ color: "#F7931E" }}>Education</span></div>
          <nav style={{ display: "flex", gap: 16 }}>
            <a href="/" style={{ color: "#fff", textDecoration: "none" }}>Home</
a>
            <a href="/courses" style={{ color: "#fff", textDecoration: "none" }}
>Courses</a>
            <a href="/blog" style={{ color: "#fff", textDecoration: "none" }}
>Blog</a>
            <a href="/contact" style={{ color: "#fff", textDecoration: "none" }}
>Contact</a>
            <a href="/apply" style={{ color: "#fff", textDecoration: "none" }}
>Apply</a>
            <a href="/login" style={{ color: "#FDE68A", textDecoration:
"none" }}>Login</a>
          </nav>
        </header>
        <main style={{ maxWidth: 1100, margin: "28px auto", padding: "0 16px" }}
>{children}</main>
        <footer style={{ marginTop: 40, padding: "16px 20px", background:
"#f6f8fa", borderTop: "1px solid #e5e7eb", textAlign: "center", color:
"#374151" }}>
```

```
          © {new Date().getFullYear()} ALO Education — Study Abroad
        </footer>
      </body>
    </html>
  );
}
```

`app/page.js` **(homepage)**

```
export default function Home() {
  return (
    <section>
      <div style={{ background: "#eaf4fe", padding: 24, borderRadius: 12 }}>
        <h1 style={{ color: "#0f172a", marginTop: 0 }}>Study in UK | Canada |
Australia</h1>
        <p>Trusted guidance from application to visa — counselor-led and
compliance-ready.</p>
        <a href="/apply" style={{ background: "#0071BC", color: "#fff",
padding: "10px 16px", borderRadius: 8, display: "inline-block", marginTop: 12 }}
>Apply Now</a>
      </div>
      <div style={{ display: "grid", gridTemplateColumns: "repeat(auto-fit,
minmax(220px, 1fr))", gap: 16, marginTop: 24 }}>
        {["Application", "Offer", "CAS", "Visa", "Enrolled"].map((s) => (
          <div key={s} style={{ border: "1px solid #e5e7eb", borderRadius: 10,
padding: 16 }}>
            <strong>{s}</strong>
            <p style={{ color: "#334155", marginBottom: 0 }}>We support you
end-to-end.</p>
          </div>
        ))}
      </div>
    </section>
  );
}
```

`app/courses/page.js`

```
export default function Courses() {
  const samples = [
    { country: "UK", name: "MSc Data Science", uni: "University of X" },
    { country: "Canada", name: "MBA", uni: "University of Y" },
    { country: "Australia", name: "BSc Computer Science", uni: "University of
Z" },
  ];
```

```
  return (
    <section>
      <h1>Popular Courses</h1>
      <ul>
        {samples.map((c, i) => (
          <li key={i}>{c.country} — {c.name} ({c.uni})</li>
        ))}
      </ul>
    </section>
  );
}
```

`app/blog/page.js`

```
export default function Blog() {
  return (
    <section>
      <h1>Guides & Articles</h1>
      <ul>
        <li>How to prepare for CAS interview</li>
        <li>Statement of Purpose (SOP) tips</li>
        <li>Visa checklist for UK</li>
      </ul>
    </section>
  );
}
```

`app/contact/page.js`

```
export default function Contact() {
  return (
    <section>
      <h1>Contact Us</h1>
      <p>Email: info@aloeducation.org</p>
      <p>Phone: +880-000-000-000</p>
    </section>
  );
}
```

`app/apply/page.js` **(public lead form → creates Student LEAD)**

```
async function createLead(formData) {
  "use server";
  const name = formData.get("name");
```

```
  const email = formData.get("email");
  const phone = formData.get("phone");
  await fetch(`${process.env.NEXTAUTH_URL || ""}/api/students`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name, email, phone }),
    cache: "no-store",
  });
}

export default function Apply() {
  return (
    <section>
      <h1>Apply Now</h1>
      <form action={createLead} style={{ maxWidth: 480 }}>
        <input name="name" placeholder="Full name" required style={{ width:
"100%", padding: 8, marginBottom: 8 }} />
        <input name="email" type="email" placeholder="Email" required style={{
width: "100%", padding: 8, marginBottom: 8 }} />
        <input name="phone" placeholder="Phone" required style={{ width:
"100%", padding: 8, marginBottom: 8 }} />
        <button style={{ background: "#0071BC", color: "#fff", padding: "10px
16px", border: 0, borderRadius: 8 }}>Submit</button>
      </form>
    </section>
  );
}
```

---

7) Auth UI + Dashboard

`app/login/page.js`

```
"use client";
import { useState } from "react";
import { signIn } from "next-auth/react";
import { useRouter } from "next/navigation";

export default function LoginPage() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const router = useRouter();

  async function handleLogin(e) {
    e.preventDefault();
```

```
    const res = await signIn("credentials", { redirect: false, email,
password });
    if (!res?.error) router.push("/dashboard");
    else alert("Invalid login");
  }

  return (
    <div style={{ maxWidth: 400, margin: "3rem auto", textAlign: "center" }}>
      <h2>Login to ALO Education</h2>
      <form onSubmit={handleLogin} style={{ marginTop: "1rem" }}>
        <input type="email" placeholder="Email" value={email} onChange={(e) =>
setEmail(e.target.value)} required style={{ width: "100%", padding: 8,
marginBottom: 12 }} />
        <input type="password" placeholder="Password" value={password}
onChange={(e) => setPassword(e.target.value)} required style={{ width: "100%",
padding: 8, marginBottom: 12 }} />
        <button type="submit" style={{ width: "100%", padding: 10, background:
"#0071BC", color: "#fff", border: "none" }}>Login</button>
      </form>
    </div>
  );
}
```

app/dashboard/page.js

```
import { getServerSession } from "next-auth";
import authOptions from "@/app/lib/authOptions";
import prisma from "@/app/lib/prisma";
import Link from "next/link";

export default async function DashboardPage() {
  const session = await getServerSession(authOptions);
  if (!session) {
    return (
      <div>
        <h2>Not signed in</h2>
        <a href="/login">Go to Login</a>
      </div>
    );
  }

  const counts = {
    total: await prisma.student.count(),
    lead: await prisma.student.count({ where: { stage: "LEAD" } }),
    offer: await prisma.student.count({ where: { stage: "OFFER" } }),
    cas: await prisma.student.count({ where: { stage: "CAS" } }),
```

```
    visa: await prisma.student.count({ where: { stage: "VISA" } }),
    enrolled: await prisma.student.count({ where: { stage: "ENROLLED" } }),
  };

  return (
    <section>
      <h1>Dashboard</h1>
      <p>Welcome, {session.user.email} — Role: {session.user.role}</p>
      <div style={{ display: "grid", gridTemplateColumns: "repeat(auto-fit,
minmax(180px, 1fr))", gap: 12 }}>
        {Object.entries(counts).map(([k, v]) => (
          <div key={k} style={{ border: "1px solid #e5e7eb", borderRadius: 8,
padding: 12 }}>
            <div style={{ textTransform: "uppercase", fontSize: 12, color:
"#64748b" }}>{k}</div>
            <div style={{ fontSize: 22, fontWeight: 700 }}>{v}</div>
          </div>
        ))}
      </div>
      <div style={{ marginTop: 16 }}>
        <Link href="/students">Manage Students →</Link>
      </div>
    </section>
  );
}
```

---

8) Students UI (list + detail)

`app/students/page.js`

```
import prisma from "@/app/lib/prisma";
import Link from "next/link";

export default async function Students() {
  const students = await prisma.student.findMany({ include: { counselor:
true }, orderBy: { createdAt: "desc" } });
  return (
    <section>
      <div style={{ display: "flex", justifyContent: "space-between",
alignItems: "center" }}>
        <h1>Students</h1>
        <a href="/students/new" style={{ background: "#0071BC", color: "#fff",
padding: "8px 12px", borderRadius: 8 }}>+ New</a>
      </div>
```

```
      <table width="100%" cellPadding="8" style={{ borderCollapse: "collapse" }}
>
        <thead>
          <tr style={{ background: "#f1f5f9" }}>
            <th align="left">Name</th><th align="left">Email</th><th
align="left">Phone</th><th align="left">Stage</th><th align="left">Counselor</
th><th></th>
          </tr>
        </thead>
        <tbody>
          {students.map(s => (
            <tr key={s.id} style={{ borderTop: "1px solid #e5e7eb" }}>
              <td>{s.name}</td>
              <td>{s.email}</td>
              <td>{s.phone}</td>
              <td>{s.stage}</td>
              <td>{s.counselor?.email || "—"}</td>
              <td><Link href={`/students/${s.id}`}>Open →</Link></td>
            </tr>
          ))}
        </tbody>
      </table>
    </section>
  );
}
```

app/students/new/page.js

```
async function create(formData) {
  "use server";
  const name = formData.get("name");
  const email = formData.get("email");
  const phone = formData.get("phone");
  await fetch(`${process.env.NEXTAUTH_URL || ""}/api/students`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ name, email, phone }),
    cache: "no-store",
  });
}

export default function NewStudent() {
  return (
    <section>
      <h1>New Student</h1>
      <form action={create} style={{ maxWidth: 480 }}>
```

```
        <input name="name" placeholder="Full name" required style={{ width:
"100%", padding: 8, marginBottom: 8 }} />
        <input name="email" type="email" placeholder="Email" required style={{
width: "100%", padding: 8, marginBottom: 8 }} />
        <input name="phone" placeholder="Phone" required style={{ width:
"100%", padding: 8, marginBottom: 8 }} />
        <button style={{ background: "#0071BC", color: "#fff", padding: "10px
16px", border: 0, borderRadius: 8 }}>Create</button>
      </form>
    </section>
  );
}
```

app/students/[id]/page.js

```
"use client";
import { useEffect, useState } from "react";
import { useRouter } from "next/navigation";

export default function StudentDetail({ params }) {
  const router = useRouter();
  const [s, setS] = useState(null);
  const [saving, setSaving] = useState(false);

  useEffect(() => {
    (async () => {
      const r = await fetch(`/api/students/${params.id}`, { cache: "no-
store" });
      if (r.ok) setS(await r.json());
    })();
  }, [params.id]);

  if (!s) return <p>Loading…</p>;

  async function update(field, value) {
    setSaving(true);
    await fetch(`/api/students/${s.id}`, {
      method: "PUT",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ [field]: value }),
    });
    setSaving(false);
  }

  return (
    <section>
```

```
      <button onClick={() => router.back()} style={{ marginBottom: 12 }}>←
Back</button>
      <h1>{s.name}</h1>
      <p><strong>Email:</strong> {s.email}</p>
      <p><strong>Phone:</strong> {s.phone}</p>

      <div style={{ marginTop: 16 }}>
        <label>Stage: </label>
        <select defaultValue={s.stage} onChange={(e) => update("stage",
e.target.value)}>
          {['LEAD','OFFER','CAS','VISA','ENROLLED'].map(x => <option key={x}
value={x}>{x}</option>)}
        </select>
      </div>

      <div style={{ marginTop: 16 }}>
        <label>Compliance: </label>
        <input type="checkbox" defaultChecked={s.compliance} onChange={(e) =>
update("compliance", e.target.checked)} />
      </div>

      <div style={{ marginTop: 16 }}>
        <label>Commission (USD): </label>
        <input type="number" step="0.01" defaultValue={s.commission || ''}
onBlur={(e) => update("commission", parseFloat(e.target.value || 0))} />
      </div>

      {saving && <p>Saving…</p>}
    </section>
  );
}
```

---

9) Templates UI (Email/SMS/WhatsApp)

`app/templates/page.js`

```
import prisma from "@/app/lib/prisma";

export default async function Templates() {
  const items = await prisma.template.findMany({ orderBy: { createdAt:
"desc" } });
  return (
    <section>
      <h1>Templates</h1>
```

```
      <ul>
        {items.map(t => (
          <li key={t.id}><strong>{t.type}</strong> — {t.name}</li>
        ))}
      </ul>
      <p style={{ marginTop: 12, color: "#64748b" }}>Sending right now is
disabled in this starter. We'll wire SMTP/WhatsApp gateways next.</p>
    </section>
  );
}
```

---

10) (Optional) Facebook Lead webhook stub (will wire later)

`app/api/facebook/lead/route.js`

```
import prisma from "@/app/lib/prisma";

export async function POST(req) {
  try {
    const { full_name, email, phone } = await req.json();
    const s = await prisma.student.create({ data: { name: full_name ||
"Unknown", email, phone, stage: "LEAD" } });
    return Response.json({ ok: true, id: s.id });
  } catch (e) {
    return new Response(JSON.stringify({ ok: false, error: e.message }), {
status: 500 });
  }
}
```

For Zapier/Meta later, we'll secure this endpoint with a secret token and verify origin.

---

11) Final steps (once all files are committed)

1. **Vercel → Settings → Environment Variables (Production)**
2. `DATABASE_URL` = your Neon connection string (without `channel_binding`)
3. `NEXTAUTH_SECRET` = the long key I generated for you
4. `NEXTAUTH_URL` = `https://aloeducation.org`
5. **Redeploy** the project in Vercel.
6. **Initialize DB** (Vercel → Deployments → … Execute Command):
7. `npx prisma db push && node prisma/seed.js`
8. **Test**
9. Visit `/login` → sign in as Admin (`admin@aloeducation.org` / `admin123`)

10. Open `/dashboard`, `/students`, `/templates`, and public pages `/`, `/courses`, `/blog`, `/contact`, `/apply`.

What's next (after you confirm it's live)

- Hook up **email** (SMTP), **SMS/WhatsApp** gateways, and secure the **Facebook Leads** endpoint.
- Add role-based guards in UI (hide actions from non-admins).
- Add file uploads for compliance documents (S3 or Vercel Blob).
- Add reporting/exports (CSV/PDF) for commissions.

When you're ready, say **"ready for comms + FB"** and I'll drop in the minimal, copy-paste integrations (with your chosen providers).

PHASE 2 — University & Course Finder + Profile Matching (GoStudyIn-style)

Below is an **easy, copy-paste pack** that adds:

- **University & Course Finder** (filters by country, level, subject, IELTS, tuition)
- **Student Profile** (GPA/IELTS/background)
- **Auto-Match**: recommends courses that fit profile requirements
- **Apply Flow**: creates an Application tied to a Student + Course (tracked in CRM)

It reuses your existing Neon DB + Prisma + Vercel setup.

A) Prisma schema — add new models

Create/replace your `prisma/schema.prisma` with the **merged** schema below (it keeps Users/Students/Templates and adds Universities/Courses/Applications/Profile). If you already customized, just copy the new models and relations.

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model User {
```

```
  id        String     @id @default(cuid())
  email     String     @unique
  password  String
  role      String     // "ADMIN" | "COUNSELOR" | "COMPLIANCE" | "FINANCE"
  students  Student[]  @relation("CounselorStudents")
  createdAt DateTime   @default(now())
}

model Student {
  id          String        @id @default(cuid())
  name        String
  email       String
  phone       String
  stage       String        // "LEAD" | "OFFER" | "CAS" | "VISA" | "ENROLLED"
  counselor   User?         @relation("CounselorStudents", fields:
[counselorId], references: [id])
  counselorId String?
  compliance  Boolean       @default(false)
  commission  Float?
  notes       String?       @db.Text
  profile     StudentProfile?
  applications Application[]
  createdAt   DateTime      @default(now())
}

model StudentProfile {
  id         String   @id @default(cuid())
  student    Student  @relation(fields: [studentId], references: [id])
  studentId  String   @unique
  gpa        Float?
  ielts      Float?
  background String?  // e.g., "BBA", "CSE", "HSC Science"
  preferredCountries String? // CSV list (e.g., "UK,Canada")
  preferredLevel     String? // "Bachelors" | "Masters"
  preferredSubject   String?
}

model University {
  id        String   @id @default(cuid())
  name      String
  country   String
  city      String?
  website   String?
  courses   Course[]
  createdAt DateTime @default(now())
}

model Course {
```

```
  id           String     @id @default(cuid())
  university   University @relation(fields: [universityId], references: [id])
  universityId String
  name         String
  level        String     // "Bachelors" | "Masters"
  subject      String
  tuition      Int?
  ieltsMin     Float?
  gpaMin       Float?
  intakes      String?    // CSV (e.g., "Jan,May,Sep")
  createdAt    DateTime   @default(now())
  applications Application[]
}

model Application {
  id         String   @id @default(cuid())
  student    Student  @relation(fields: [studentId], references: [id])
  studentId  String
  course     Course   @relation(fields: [courseId], references: [id])
  courseId   String
  status     String   @default("SUBMITTED") // SUBMITTED | OFFERED | CAS | VISA
| ENROLLED | REJECTED
  createdAt  DateTime @default(now())
}

model Template {
  id        String   @id @default(cuid())
  type      String   // "EMAIL" | "SMS" | "WHATSAPP"
  name      String
  body      String   @db.Text
  createdAt DateTime @default(now())
}
```

Deploy schema:

```
npx prisma db push
```

B) Seed data — universities & courses

Append this to your existing `prisma/seed.js` (or replace with full version below) to preload a few samples so Finder works immediately.

```
// === Universities & Courses seed ===
const uniData = [
  {
    name: "London International University",
    country: "UK",
    city: "London",
    website: "https://example-uni.uk",
    courses: [
      { name: "MSc Data Science", level: "Masters", subject: "Data Science",
tuition: 18000, ieltsMin: 6.5, gpaMin: 3.0, intakes: "Jan,Sep" },
      { name: "MBA", level: "Masters", subject: "Business", tuition: 20000,
ieltsMin: 6.5, gpaMin: 3.0, intakes: "Jan,May,Sep" }
    ]
  },
  {
    name: "Toronto Metropolitan University",
    country: "Canada",
    city: "Toronto",
    website: "https://example-uni.ca",
    courses: [
      { name: "BSc Computer Science", level: "Bachelors", subject: "Computer
Science", tuition: 23000, ieltsMin: 6.0, gpaMin: 2.7, intakes: "Jan,Sep" }
    ]
  }
];

for (const u of uniData) {
  const uni = await prisma.university.upsert({
    where: { name: u.name },
    update: {},
    create: { name: u.name, country: u.country, city: u.city, website:
u.website },
  });
  for (const c of u.courses) {
    await prisma.course.upsert({
      where: { name_universityId: { name: c.name, universityId: uni.id } },
      update: {},
      create: { ...c, universityId: uni.id },
    });
  }
}
```

Note: Add a unique composite in your DB via Prisma by changing Course model to include `@@unique([name, universityId], name: "name_universityId")` if you prefer upsert by (name, uni). If you skip that, change `where` to use another unique field.

After updating `seed.js`, run:

```
npx prisma db push && node prisma/seed.js
```

C) Finder API — search courses

Create `app/api/courses/search/route.js`:

```
import prisma from "@/app/lib/prisma";

export async function POST(req) {
  const { country, level, subject, ielts, maxTuition } = await req.json();
  const where = {};
  if (country) where.university = { country };
  if (level) where.level = level;
  if (subject) where.subject = { contains: subject, mode: "insensitive" };
  if (maxTuition) where.tuition = { lte: Number(maxTuition) };
  if (ielts) where.ieltsMin = { lte: Number(ielts) };

  const courses = await prisma.course.findMany({
    where,
    include: { university: true },
    orderBy: { createdAt: "desc" },
    take: 50,
  });
  return Response.json(courses);
}
```

D) Finder page — `/finder`

Create `app/finder/page.js`:

```
"use client";
import { useState } from "react";

export default function Finder() {
  const [filters, setFilters] = useState({ country: "", level: "", subject: "",
ielts: "", maxTuition: "" });
  const [results, setResults] = useState([]);
  const [loading, setLoading] = useState(false);
```

```
  async function search() {
    setLoading(true);
    const r = await fetch("/api/courses/search", { method: "POST", headers: {
"Content-Type": "application/json" }, body: JSON.stringify(filters) });
    const data = await r.json();
    setResults(data);
    setLoading(false);
  }

  return (
    <section>
      <h1>University & Course Finder</h1>
      <div style={{ display: "grid", gridTemplateColumns: "repeat(auto-
fit,minmax(180px,1fr))", gap: 8, marginBottom: 12 }}>
        <input placeholder="Country (UK/Canada/...)" value={filters.country}
onChange={e=>setFilters({ ...filters, country: e.target.value })} />
        <select value={filters.level} onChange={e=>setFilters({ ...filters,
level: e.target.value })}>
          <option value="">Level</option>
          <option>Masters</option>
          <option>Bachelors</option>
        </select>
        <input placeholder="Subject (e.g., Data Science)"
value={filters.subject} onChange={e=>setFilters({ ...filters, subject:
e.target.value })} />
        <input placeholder="Min IELTS (≤)" type="number" step="0.5"
value={filters.ielts} onChange={e=>setFilters({ ...filters, ielts:
e.target.value })} />
        <input placeholder="Max Tuition (USD)" type="number"
value={filters.maxTuition} onChange={e=>setFilters({ ...filters, maxTuition:
e.target.value })} />
        <button onClick={search} style={{ background: "#0071BC", color: "#fff",
border: 0, borderRadius: 6, padding: "8px 12px" }}
>{loading?"Searching…":"Search"}</button>
      </div>
      <ul>
        {results.map(c => (
          <li key={c.id} style={{ borderTop: "1px solid #e5e7eb", padding: "8px
0" }}>
            <strong>{c.name}</strong> — {c.level}, {c.subject} @
{c.university?.name} ({c.university?.country})
            {typeof c.tuition === "number" && <> — Tuition: ${c.tuition}</>}
          </li>
        ))}
      </ul>
    </section>
```

```
  );
}
```

Optional: Add a nav link in `app/layout.js` → `<a href="/finder">Finder</a>`.

E) Student Profile page — `/profile`

Create `app/profile/page.js`:

```
async function saveProfile(formData) {
  "use server";
  const payload = Object.fromEntries(formData);
  await fetch(`${process.env.NEXTAUTH_URL || ""}/api/profile`, { method:
"POST", headers: { "Content-Type": "application/json" }, body:
JSON.stringify(payload) });
}

export default function Profile() {
  return (
    <section>
      <h1>Student Profile</h1>
      <form action={saveProfile} style={{ maxWidth: 520 }}>
        <input name="studentEmail" type="email"
placeholder="Student Email (must exist)" required style={{ width: "100%",
padding: 8, marginBottom: 8 }} />
        <input name="gpa" type="number" step="0.01" placeholder="GPA (e.g.,
3.2)" style={{ width: "100%", padding: 8, marginBottom: 8 }} />
        <input name="ielts" type="number" step="0.5" placeholder="IELTS (e.g.,
6.5)" style={{ width: "100%", padding: 8, marginBottom: 8 }} />
        <input name="background" placeholder="Background (e.g., BBA/HSC/CSE)"
style={{ width: "100%", padding: 8, marginBottom: 8 }} />
        <input name="preferredCountries"
placeholder="Preferred Countries (e.g., UK,Canada)" style={{ width: "100%",
padding: 8, marginBottom: 8 }} />
        <select name="preferredLevel" style={{ width: "100%", padding: 8,
marginBottom: 8 }}>
          <option value="">Preferred Level</option>
          <option>Bachelors</option>
          <option>Masters</option>
        </select>
        <input name="preferredSubject" placeholder="Preferred Subject (e.g.,
Data Science)" style={{ width: "100%", padding: 8, marginBottom: 8 }} />
        <button style={{ background: "#0071BC", color: "#fff", border: 0,
borderRadius: 8, padding: "10px 16px" }}>Save Profile</button>
      </form>
```

```
    </section>
  );
}
```

Create `app/api/profile/route.js`:

```
import prisma from "@/app/lib/prisma";

export async function POST(req) {
  const { studentEmail, gpa, ielts, background, preferredCountries,
preferredLevel, preferredSubject } = await req.json();
  const student = await prisma.student.findUnique({ where: { email:
studentEmail } });
  if (!student) return new Response("Student not found", { status: 404 });

  const data = {
    studentId: student.id,
    gpa: gpa ? parseFloat(gpa) : null,
    ielts: ielts ? parseFloat(ielts) : null,
    background: background || null,
    preferredCountries: preferredCountries || null,
    preferredLevel: preferredLevel || null,
    preferredSubject: preferredSubject || null,
  };

  await prisma.studentProfile.upsert({
    where: { studentId: student.id },
    update: data,
    create: data,
  });

  return Response.json({ ok: true });
}
```

F) Auto-Match page — `/match`

Create `app/match/page.js`:

```
"use client";
import { useState } from "react";

export default function Match() {
  const [email, setEmail] = useState("");
```

```
  const [rows, setRows] = useState([]);

  async function run() {
    const r = await fetch(`/api/match?email=${encodeURIComponent(email)}`, {
cache: "no-store" });
    if (r.ok) setRows(await r.json());
    else alert("Profile or matches not found");
  }

  return (
    <section>
      <h1>Match Courses to Student Profile</h1>
      <div style={{ display: "flex", gap: 8, marginBottom: 12 }}>
        <input placeholder="Student Email" value={email}
onChange={e=>setEmail(e.target.value)} />
        <button onClick={run} style={{ background: "#0071BC", color: "#fff",
border: 0, borderRadius: 6, padding: "8px 12px" }}>Match</button>
      </div>
      <ul>
        {rows.map(c => (
          <li key={c.id} style={{ borderTop: "1px solid #e5e7eb", padding: "8px
0" }}>
            <strong>{c.name}</strong> — {c.level}, {c.subject} @
{c.university?.name} ({c.university?.country})
            {typeof c.tuition === "number" && <> — Tuition: ${c.tuition}</>}
          </li>
        ))}
      </ul>
    </section>
  );
}
```

Create `app/api/match/route.js`:

```
import prisma from "@/app/lib/prisma";

export async function GET(req) {
  const { searchParams } = new URL(req.url);
  const email = searchParams.get("email");
  if (!email) return new Response("Email required", { status: 400 });

  const student = await prisma.student.findUnique({ where: { email }, include:
{ profile: true } });
  if (!student?.profile) return new Response("Profile not found", { status:
404 });
```

```
  const p = student.profile;
  const where = {};
  if (p.preferredCountries) where.university = { country: { in:
p.preferredCountries.split(",").map(x=>x.trim()) } };
  if (p.preferredLevel) where.level = p.preferredLevel;
  if (p.preferredSubject) where.subject = { contains: p.preferredSubject, mode:
"insensitive" };
  if (p.ielts) where.ieltsMin = { lte: p.ielts };
  if (p.gpa) where.gpaMin = { lte: p.gpa };

  const courses = await prisma.course.findMany({ where, include: { university:
true }, take: 50 });
  return Response.json(courses);
}
```

G) Apply flow — create Application from CRM

Create `app/api/applications/route.js`:

```
import prisma from "@/app/lib/prisma";

export async function POST(req) {
  const { studentId, courseId } = await req.json();
  const app = await prisma.application.create({ data: { studentId, courseId,
status: "SUBMITTED" } });
  return Response.json(app, { status: 201 });
}
```

Create a simple UI button on Student Detail to apply — edit `app/students/[id]/page.js` and append inside the component:

```
// ... after existing controls
<div style={{ marginTop: 24, paddingTop: 12, borderTop: "1px solid #e5e7eb" }}>
  <h3>Apply to a Course</h3>
  <ApplyForm studentId={s.id} />
</div>
```

Add a small client subcomponent at the bottom of the same file:

```
function ApplyForm({ studentId }) {
  const [courseId, setCourseId] = useState("");
  const [msg, setMsg] = useState("");
```

```
  async function submit(e){
    e.preventDefault();
    const r = await fetch("/api/applications", { method: "POST", headers: {
"Content-Type": "application/json" }, body: JSON.stringify({ studentId,
courseId }) });
    setMsg(r.ok ? "Application submitted" : "Failed");
  }
  return (
    <form onSubmit={submit} style={{ display: "flex", gap: 8 }}>
      <input placeholder="Course ID" value={courseId}
onChange={e=>setCourseId(e.target.value)} />
      <button style={{ background: "#0071BC", color: "#fff", border: 0,
borderRadius: 6, padding: "8px 12px" }}>Apply</button>
      {msg && <span>{msg}</span>}
    </form>
  );
}
```

In production, replace the free-text `Course ID` with a picker bound to `/finder` results.

H) Navigation updates

Open `app/layout.js` and add these links to the nav:

```
<a href="/finder" style="color:#fff;text-decoration:none">Finder</a>
<a href="/profile" style="color:#fff;text-decoration:none">Profile</a>
<a href="/match" style="color:#fff;text-decoration:none">Match</a>
```

I) One-time commands (after committing files)

1. **Deploy** (Vercel auto build on push)
2. **Init DB** from Vercel → Deployments → Execute Command:

```
npx prisma db push && node prisma/seed.js
```

1. **Test flow**
2. Public: `/finder` searches seeded courses
3. CRM: create a Student → `/profile` save profile with same email → `/match` shows recommended courses
4. Student page → paste a `courseId` from Finder and click **Apply** (creates Application)

What this gives you now

- A GoStudyIn-style **Finder + Profile + Matching** pipeline
- **Apply** to a course and track via `Application` model
- All branded with your ALO colors/logo

When you're ready, I can add:

- File uploads for **Compliance** (SOP, Passport, Bank, CAS)
- **WhatsApp/SMS/Email** send using templates
- **Facebook Lead Ads** secure webhook (token + verify) and auto-assign rules
- Downloadable **Reports** for commissions & intakes

PHASE 3 — Comms (Email/SMS/WhatsApp), Compliance File Uploads, Secure Facebook Leads

This pack adds:

- **Compliance uploads** stored in Vercel Blob (simple + free tier) and linked to Students
- **Email** via SMTP (Nodemailer)
- **SMS & WhatsApp** via Twilio
- **Facebook Lead Ads** webhook with **verification** + **secure token** + mapping to Students

It's copy-paste ready. After committing files, set the env vars and run the single DB command at the end.

A) Prisma — add ComplianceDocument model

Append this to your existing `prisma/schema.prisma` models and re-push. (If you still have the Phase-2 schema from above, just add this model.)

```
model ComplianceDocument {
  id         String   @id @default(cuid())
  student    Student  @relation(fields: [studentId], references: [id])
  studentId  String
  kind       String   // PASSPORT | SOP | TRANSCRIPT | BANK | CAS | OTHER
  url        String   // Blob/S3 URL
  uploadedAt DateTime @default(now())
}
```

Then run: `npx prisma db push`

B) Vercel Blob setup (for uploads)

Add dependency in `package.json` if not present:

```
"@vercel/blob": "^0.21.0"
```

(Commit `package.json`; Vercel will auto-install.)

Add an env var in Vercel (Project → Settings → Environment Variables):

- `BLOB_READ_WRITE_TOKEN` = (Create in Vercel → Storage → **Blob** → Tokens → Generate RW token)

C) API — Compliance upload (multipart)

Create `app/api/upload/compliance/route.js`:

```
import { put } from "@vercel/blob";
import prisma from "@/app/lib/prisma";

export const runtime = "edge"; // blob works great on edge

export async function POST(req) {
  try {
    const form = await req.formData();
    const file = form.get("file");
    const studentId = form.get("studentId");
    const kind = form.get("kind") || "OTHER";

    if (!file || !studentId) return new Response("Missing file or studentId", {
status: 400 });

    const filename = `${studentId}-${Date.now()}-${file.name}`;
    const { url } = await put(filename, file, { access: "public" });

    await prisma.complianceDocument.create({
      data: { studentId, kind, url }
    });

    return Response.json({ ok: true, url });
  } catch (e) {
    return new Response(JSON.stringify({ ok: false, error: e.message }), {
status: 500 });
```

```
  }
}
```

Add a simple UI to Student Detail page (append inside `app/students/[id]/page.js`, inside the component, e.g., after commission field):

```
<div style={{ marginTop: 24, paddingTop: 12, borderTop: "1px solid #e5e7eb" }}>
  <h3>Compliance Documents</h3>
  <ComplianceUpload studentId={s.id} />
</div>
```

Then at the bottom of the same file add this client subcomponent:

```
function ComplianceUpload({ studentId }) {
  const [file, setFile] = useState(null);
  const [kind, setKind] = useState("PASSPORT");
  const [msg, setMsg] = useState("");

  async function submit(e){
    e.preventDefault();
    const fd = new FormData();
    fd.append("file", file);
    fd.append("studentId", studentId);
    fd.append("kind", kind);
    const r = await fetch("/api/upload/compliance", { method: "POST", body:
fd });
    setMsg(r.ok ? "Uploaded" : "Failed");
  }

  return (
    <form onSubmit={submit} style={{ display: "flex", gap: 8, alignItems:
"center", flexWrap: "wrap" }}>
      <select value={kind} onChange={e=>setKind(e.target.value)}>
        {['PASSPORT','SOP','TRANSCRIPT','BANK','CAS','OTHER'].map(k=> <option
key={k}>{k}</option>)}
      </select>
      <input type="file" onChange={e=>setFile(e.target.files?.[0]||null)}
required />
      <button style={{ background: "#0071BC", color: "#fff", border:
```