class-action (gofundalawsuit.com) — Production Blueprint

> Windows + VS Code + Next.js (App Router) + Netlify + Prisma (MySQL/PlanetScale) + Stripe Connect (payments) + Resend (emails) + TOTP (speakeasy). All secrets via environment variables. All sensitive logic runs server-side.

---

0) Quick Setup (Windows)

```
# 1) Prereqs
node -v           # >= 20
npm -v            # or use pnpm

# 2) Create app
npx create-next-app@latest class-action --ts --eslint --tailwind --app --src-dir --use-npm
cd class-action

# 3) Install deps
npm i @prisma/client prisma bcrypt jsonwebtoken zod next-safe-action
npm i -D @types/jsonwebtoken @types/bcrypt ts-node
# Email + TOTP + payments + uploads + utils
npm i resend speakeasy qrcode stripe formidable
# DB driver (PlanetScale or other MySQL)
npm i mysql2

# 4) Initialize Prisma
npx prisma init

# 5) Create Netlify files
# (created below: netlify.toml + edge functions enabled)

# 6) After env vars are set
npm run dev
```

> Recommended DB: **PlanetScale (MySQL)** for serverless-friendly MySQL. Alternative: **Neon/Supabase** (Postgres). This blueprint uses MySQL.

---

1) File Tree (App Router)

```
class-action/
├─ netlify.toml
├─ package.json
├─ prisma/
│  └─ schema.prisma
├─ src/
│  ├─ app/
│  │  ├─ (public)/
│  │  │  ├─ page.tsx                      # Home
│  │  │  ├─ how-it-works/page.tsx
│  │  │  ├─ pricing/page.tsx
│  │  │  ├─ trust-safety/page.tsx
│  │  │  ├─ faq/page.tsx
│  │  │  ├─ blog/page.tsx
│  │  │  └─ blog/[slug]/page.tsx
│  │  ├─ cases/
│  │  │  ├─ page.tsx                      # Explore
│  │  │  └─ [slug]/page.tsx               # Case Details
│  │  ├─ start/
│  │  │  ├─ page.tsx                      # Wizard entry
│  │  │  └─ steps/
│  │  │     ├─ about/page.tsx
│  │  │     ├─ jurisdiction/page.tsx
│  │  │     ├─ parties/page.tsx
│  │  │     ├─ counsel/page.tsx
│  │  │     ├─ budget/page.tsx
│  │  │     ├─ evidence/page.tsx
│  │  │     ├─ escrow/page.tsx
│  │  │     ├─ kyc/page.tsx
│  │  │     └─ preview/page.tsx
│  │  ├─ legal/
│  │  │  ├─ terms/page.tsx
│  │  │  ├─ privacy/page.tsx
│  │  │  ├─ risk/page.tsx
│  │  │  └─ aup/page.tsx
│  │  ├─ compliance/page.tsx
│  │  ├─ contact/page.tsx
│  │  ├─ auth/
│  │  │  ├─ login/page.tsx
│  │  │  ├─ signup/page.tsx
│  │  │  ├─ callback/route.ts             # oauth/magic links if needed later
│  │  │  └─ logout/route.ts
│  │  ├─ dashboard/
```

```
│   │   │   ├─ backer/page.tsx
│   │   │   ├─ organizer/page.tsx
│   │   │   ├─ messages/page.tsx
│   │   │   ├─ pledges/page.tsx
│   │   │   ├─ withdrawals/page.tsx
│   │   │   ├─ verification/page.tsx
│   │   │   ├─ settings/page.tsx
│   │   │   └─ saved/page.tsx
│   │   ├─ admin/
│   │   │   ├─ page.tsx
│   │   │   ├─ cases/page.tsx
│   │   │   ├─ users/page.tsx
│   │   │   ├─ payments/page.tsx
│   │   │   ├─ disputes/page.tsx
│   │   │   ├─ moderation/page.tsx
│   │   │   ├─ compliance/page.tsx
│   │   │   ├─ settings/page.tsx
│   │   │   └─ audit-logs/page.tsx
│   │   ├─ api/
│   │   │   ├─ auth/
│   │   │   │   ├─ request-magic-link/route.ts
│   │   │   │   ├─ verify-magic-link/route.ts
│   │   │   │   ├─ totp/setup/route.ts
│   │   │   │   ├─ totp/verify/route.ts
│   │   │   │   └─ session/route.ts
│   │   │   ├─ uploads/route.ts              # formidable for uploads (server-
only)
│   │   │   ├─ cases/route.ts                # POST create, GET list
│   │   │   ├─ pledges/route.ts              # POST create pledge intent
│   │   │   ├─ webhooks/
│   │   │   │   └─ stripe/route.ts           # payment + escrow events
│   │   │   └─ admin/
│   │   │      └─ moderation/route.ts
│   │   ├─ layout.tsx
│   │   ├─ globals.css
│   │   └─ middleware.ts                     # auth gating + security headers
│   ├─ components/
│   │   ├─ AppShell.tsx
│   │   ├─ NavBar.tsx
│   │   ├─ Footer.tsx
│   │   ├─ CaseCard.tsx
│   │   ├─ PledgeWidget.tsx
│   │   ├─ ProgressBar.tsx
│   │   ├─ FilterPanel.tsx
│   │   ├─ IdentityBadge.tsx
│   │   ├─ KYCStatus.tsx
```

```
│   │   ├─ UploadField.tsx
│   │   ├─ RedactionNotice.tsx
│   │   ├─ Timeline.tsx
│   │   ├─ UpdatesFeed.tsx
│   │   ├─ Comments.tsx
│   │   ├─ Stepper.tsx
│   │   ├─ Forms/
│   │   │   ├─ Text.tsx
│   │   │   ├─ Select.tsx
│   │   │   ├─ Money.tsx
│   │   │   ├─ Checkbox.tsx
│   │   │   └─ File.tsx
│   │   └─ Admin/
│   │      ├─ ModerationTable.tsx
│   │      └─ AuditLogTable.tsx
│   ├─ lib/
│   │   ├─ db.ts                          # Prisma client
│   │   ├─ auth.ts                        # session utils (cookies, JWT sign)
│   │   ├─ email.ts                       # Resend helper
│   │   ├─ totp.ts                        # speakeasy helpers
│   │   ├─ payments.ts                    # Stripe helpers (checkout,
transfers)
│   │   ├─ webhook.ts                     # signature verify + idempotency
│   │   ├─ security.ts                    # rate limits, headers
│   │   └─ validation.ts                  # zod schemas
│   ├─ server/
│   │   ├─ actions/                       # server actions with next-safe-
action
│   │   └─ policies/                      # role-based authorization checks
│   ├─ styles/
│   │   └─ theme.css
│   ├─ types/
│   │   ├─ auth.d.ts
│   │   ├─ cases.d.ts
│   │   ├─ payments.d.ts
│   │   └─ common.d.ts
│   └─ utils/
│      └─ format.ts
└─ .env.example
```

---

2) `.env.example` (copy to `.env.local`)

```
# App
NEXT_PUBLIC_APP_NAME="class-action"
NEXT_PUBLIC_APP_URL="http://localhost:3000"
NODE_ENV="development"

# Database (PlanetScale)
DATABASE_URL="mysql://USER:PASSWORD@HOST/DATABASE?sslaccept=strict"

# Auth + JWT
JWT_SECRET="replace-with-strong-random-64+chars"
SESSION_COOKIE_NAME="classaction.sid"
SESSION_COOKIE_DOMAIN="localhost"        # set to .gofundalawsuit.com in prod
SESSION_COOKIE_SECURE="false"             # true in prod
SESSION_COOKIE_SAMESITE="strict"

# Email (Resend)
RESEND_API_KEY="re_..."
RESEND_FROM_EMAIL="no-reply@gofundalawsuit.com"

# TOTP
TOTP_ISSUER="class-action"

# Stripe (payments + Connect)
STRIPE_SECRET_KEY="sk_live_..."
STRIPE_WEBHOOK_SECRET="whsec_..."
STRIPE_PRICE_PLEDGE="price_..."            # optional if using Checkout
STRIPE_PLATFORM_FEE_BPS="500"              # 5.00% as basis points example

# Uploads (S3-compatible if desired)
S3_ENDPOINT=""
S3_REGION=""
S3_ACCESS_KEY_ID=""
S3_SECRET_ACCESS_KEY=""
S3_BUCKET=""

# Security + Rate Limit (simple in-memory or external like Upstash)
RATE_LIMIT_WINDOW_MS="60000"
RATE_LIMIT_MAX="60"
```

---

3) Prisma schema (MySQL / PlanetScale)

`prisma/schema.prisma`

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

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

model User {
  id            String   @id @default(cuid())
  email         String   @unique
  emailVerified DateTime?
  hashedPassword String?  // optional if you add password auth later
  role          Role      @default(BACKER)
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt

  profile       Profile?
  totp          TOTPSecret?
  sessions      Session[]
  pledges       Pledge[]
  cases         Case[]    @relation("OrganizerCases")
}

enum Role {
  BACKER
  ORGANIZER
  COUNSEL
  ADMIN
  COMPLIANCE
}

model Profile {
  id        String @id @default(cuid())
  userId    String @unique
  user      User   @relation(fields: [userId], references: [id])
  name      String?
  country   String?
  timezone  String?
}

model Session {
  id           String   @id @default(cuid())
  userId       String
  user         User     @relation(fields: [userId], references: [id])
  token        String   @unique
```

```
  createdAt    DateTime @default(now())
  expiresAt    DateTime
  userAgent    String?
  ip           String?
}

model VerificationToken {
  id           String   @id @default(cuid())
  email        String
  token        String   @unique
  createdAt    DateTime @default(now())
  expiresAt    DateTime
  usedAt       DateTime?
}

model TOTPSecret {
  id           String   @id @default(cuid())
  userId       String   @unique
  user         User     @relation(fields: [userId], references: [id])
  secretHash   String   // store a bcrypt hash of the base32 secret
  enabled      Boolean  @default(false)
  createdAt    DateTime @default(now())
}

model Case {
  id            String   @id @default(cuid())
  slug          String   @unique
  title         String
  narrative     String
  jurisdiction  String
  goalCents     Int
  raisedCents   Int       @default(0)
  status        CaseStatus @default(DRAFT)
  visibility    Visibility  @default(PUBLIC)
  organizerId   String
  organizer     User       @relation("OrganizerCases", fields: [organizerId],
references: [id])
  createdAt     DateTime   @default(now())
  updatedAt     DateTime   @updatedAt

  documents     CaseDocument[]
  budgets       BudgetItem[]
  pledges       Pledge[]
  updates       Update[]
  comments      Comment[]
}

enum CaseStatus {
```

```
  DRAFT
  REVIEW
  LIVE
  PAUSED
  CLOSED
  REJECTED
}

enum Visibility {
  PUBLIC
  UNLISTED
  PRIVATE
}

model CaseDocument {
  id         String  @id @default(cuid())
  caseId     String
  case       Case    @relation(fields: [caseId], references: [id])
  key        String  // storage key
  kind       String  // e.g., "evidence", "order", "image"
  redacted   Boolean @default(false)
  createdAt  DateTime @default(now())
}

model BudgetItem {
  id         String  @id @default(cuid())
  caseId     String
  case       Case    @relation(fields: [caseId], references: [id])
  label      String
  amountCents Int
}

model Pledge {
  id            String   @id @default(cuid())
  userId        String
  user          User     @relation(fields: [userId], references: [id])
  caseId        String
  case          Case     @relation(fields: [caseId], references: [id])
  amountCents   Int
  currency      String   @default("usd")
  status        PledgeStatus @default(PENDING)
  provider      String   // "stripe"
  providerRef   String?  // checkout/session/intent id
  createdAt     DateTime @default(now())
}

enum PledgeStatus {
  PENDING
```

```
  SUCCEEDED
  REFUNDED
  DISPUTED
  FAILED
}

model EscrowLedger {
  id           String   @id @default(cuid())
  caseId       String
  case         Case     @relation(fields: [caseId], references: [id])
  deltaCents   Int      // positive for funds in, negative for release/refund
  reason       String
  providerRef  String?
  createdAt    DateTime @default(now())
}

model Update {
  id        String   @id @default(cuid())
  caseId    String
  case      Case     @relation(fields: [caseId], references: [id])
  title     String
  body      String
  createdAt DateTime @default(now())
}

model Comment {
  id        String   @id @default(cuid())
  caseId    String
  case      Case     @relation(fields: [caseId], references: [id])
  userId    String
  user      User     @relation(fields: [userId], references: [id])
  body      String
  flagged   Boolean  @default(false)
  createdAt DateTime @default(now())
}

model WebhookEvent {
  id            String   @id @default(cuid())
  provider      String
  eventType     String
  payloadHash   String   // HMAC for immutability checks
  idempotencyKey String  @unique
  receivedAt    DateTime @default(now())
  processedAt   DateTime?
  success       Boolean  @default(false)
  error         String?
}
```

```
model AuditLog {
  id        String   @id @default(cuid())
  actorId   String?
  action    String
  entity    String
  entityId  String?
  meta      Json?
  createdAt DateTime @default(now())
}
```

Run migrations:

```
npx prisma migrate dev --name init
npx prisma db push   # for PlanetScale shadow envs if needed
```

---

4) Middleware: Security headers + simple auth gating

`src/app/middleware.ts`

```
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

export function middleware(req: NextRequest) {
  const res = NextResponse.next();

  // Security headers
  res.headers.set('X-Frame-Options', 'DENY');
  res.headers.set('X-Content-Type-Options', 'nosniff');
  res.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');
  res.headers.set('Permissions-Policy', 'camera=(), microphone=(),
geolocation=()');

  // Example: protect /admin
  if (req.nextUrl.pathname.startsWith('/admin')) {
    const sid = req.cookies.get(process.env.SESSION_COOKIE_NAME ||
'classaction.sid');
    if (!sid) return NextResponse.redirect(new URL('/auth/login', req.url));
  }

  return res;
}

export const config = {
```

```
  matcher: ['/((?!_next|api/webhooks/stripe|favicon.ico).*)'],
};
```

---

5) Auth flow (Magic Link + TOTP)

5.1 Generate and email magic link

`src/app/api/auth/request-magic-link/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { z } from 'zod';
import { prisma } from '@/lib/db';
import crypto from 'crypto';
import { sendLoginEmail } from '@/lib/email';

const schema = z.object({ email: z.string().email() });

export async function POST(req: NextRequest) {
  const body = await req.json();
  const { email } = schema.parse(body);

  const token = crypto.randomUUID();
  const expires = new Date(Date.now() + 1000 * 60 * 15); // 15 min

  await prisma.verificationToken.create({ data: { email, token, expiresAt:
expires } });

  const baseUrl = process.env.NEXT_PUBLIC_APP_URL!;
  const url = `${baseUrl}/api/auth/verify-magic-link?token=${token}`;
  await sendLoginEmail(email, url);

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

`src/lib/email.ts`

```
import { Resend } from 'resend';

const resend = new Resend(process.env.RESEND_API_KEY!);
const FROM = process.env.RESEND_FROM_EMAIL!;

export async function sendLoginEmail(to: string, url: string) {
  await resend.emails.send({
```

```
    from: FROM,
    to,
    subject: 'Your secure sign-in link',
    html: `<p>Click to sign in: <a href="${url}">${url}</a></p><p>This link
expires in 15 minutes.</p>`,
  });
}
```

5.2 Verify magic link, start session

`src/app/api/auth/verify-magic-link/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/db';
import { cookies } from 'next/headers';
import jwt from 'jsonwebtoken';

export async function GET(req: NextRequest) {
  const token = req.nextUrl.searchParams.get('token');
  if (!token) return NextResponse.json({ error: 'Missing token' }, { status:
400 });

  const vt = await prisma.verificationToken.findUnique({ where: { token } });
  if (!vt || vt.usedAt || vt.expiresAt < new Date()) {
    return NextResponse.json({ error: 'Invalid or expired token' }, { status:
400 });
  }

  await prisma.verificationToken.update({ where: { id: vt.id }, data: { usedAt:
new Date() } });

  // Upsert user
  const user = await prisma.user.upsert({
    where: { email: vt.email },
    update: {},
    create: { email: vt.email },
  });

  // Issue session JWT
  const payload = { sub: user.id, role: user.role };
  const signed = jwt.sign(payload, process.env.JWT_SECRET!, { expiresIn:
'7d' });

  cookies().set({
    name: process.env.SESSION_COOKIE_NAME!,
    value: signed,
```

```
    httpOnly: true,
    sameSite: process.env.SESSION_COOKIE_SAMESITE as any,
    secure: process.env.SESSION_COOKIE_SECURE === 'true',
    domain: process.env.SESSION_COOKIE_DOMAIN,
    path: '/',
    maxAge: 60 * 60 * 24 * 7,
  });

  // If user has TOTP enabled, redirect to TOTP verification UI
  const totp = await prisma.tOTPSecret.findUnique({ where: { userId:
user.id } });
  const redirect = totp?.enabled ? '/account/2fa' : '/dashboard/backer';

  return NextResponse.redirect(new URL(redirect,
process.env.NEXT_PUBLIC_APP_URL!));
}
```

5.3 TOTP helpers

`src/lib/totp.ts`

```
import speakeasy from 'speakeasy';
import bcrypt from 'bcrypt';

export function generateTotpSecret(issuer: string, email: string) {
  const secret = speakeasy.generateSecret({ length: 20, issuer, name: `$
{issuer}:${email}` });
  return secret; // { ascii, hex, base32, otpauth_url }
}

export function verifyTotp(token: string, base32: string) {
  return speakeasy.totp.verify({ secret: base32, encoding: 'base32', token,
window: 1 });
}

export async function hashSecret(base32: string) {
  return bcrypt.hash(base32, 12);
}

export async function compareSecret(base32: string, hash: string) {
  return bcrypt.compare(base32, hash);
}
```

5.4 TOTP setup and verify routes

`src/app/api/auth/totp/setup/route.ts`

```ts
import { NextRequest, NextResponse } from 'next/server';
import { cookies } from 'next/headers';
import jwt from 'jsonwebtoken';
import qrcode from 'qrcode';
import { prisma } from '@/lib/db';
import { generateTotpSecret, hashSecret } from '@/lib/totp';

export async function POST(req: NextRequest) {
  const cookie = cookies().get(process.env.SESSION_COOKIE_NAME!);
  if (!cookie) return NextResponse.json({ error: 'Unauthorized' }, { status:
401 });

  const payload = jwt.verify(cookie.value, process.env.JWT_SECRET!) as any;
  const user = await prisma.user.findUnique({ where: { id: payload.sub } });
  if (!user) return NextResponse.json({ error: 'Unauthorized' }, { status:
401 });

  const secret = generateTotpSecret(process.env.TOTP_ISSUER!, user.email);
  const svg = await qrcode.toString(secret.otpauth_url!, { type: 'svg' });
  const secretHash = await hashSecret(secret.base32);

  await prisma.tOTPSecret.upsert({
    where: { userId: user.id },
    update: { secretHash: secretHash, enabled: false },
    create: { userId: user.id, secretHash: secretHash, enabled: false },
  });

  return NextResponse.json({ svg, base32: secret.base32 });
}
```

`src/app/api/auth/totp/verify/route.ts`

```ts
import { NextRequest, NextResponse } from 'next/server';
import { prisma } from '@/lib/db';
import { verifyTotp, compareSecret } from '@/lib/totp';
import { cookies } from 'next/headers';
import jwt from 'jsonwebtoken';

export async function POST(req: NextRequest) {
  const { token } = await req.json();
  const cookie = cookies().get(process.env.SESSION_COOKIE_NAME!);
  if (!cookie) return NextResponse.json({ error: 'Unauthorized' }, { status:
401 });

  const payload = jwt.verify(cookie.value, process.env.JWT_SECRET!) as any;
  const stored = await prisma.tOTPSecret.findUnique({ where: { userId:
```

```
payload.sub } });
  if (!stored) return NextResponse.json({ error: 'No TOTP secret' }, { status:
400 });

  // We cannot decrypt hash; ask user to submit the base32 they received during
setup
  // Alternatively store encrypted secret instead of hash if you want silent
verify
  const { base32 } = await req.json();
  const ok = await compareSecret(base32, stored.secretHash);
  if (!ok) return NextResponse.json({ error: 'Secret mismatch' }, { status:
400 });

  const valid = verifyTotp(token, base32);
  if (!valid) return NextResponse.json({ error: 'Invalid code' }, { status:
400 });

  await prisma.tOTPSecret.update({ where: { id: stored.id }, data: { enabled:
true } });
  return NextResponse.json({ ok: true });
}
```

> **Note**: For a smoother UX, encrypt and store the plaintext base32 with a server-side key instead of hashing, so you do not need the user to post it again during verify. Hashing is stricter privacy but adds friction.

---

6) Payments + Escrow (Stripe Connect) and Webhooks

6.1 Helper: initialize Stripe and create pledge intent

`src/lib/payments.ts`

```
import Stripe from 'stripe';
import { prisma } from '@/lib/db';

export const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!, { apiVersion:
'2024-06-20' });

export async function createCheckoutSession({
  userId,
  caseId,
  amountCents,
  successUrl,
  cancelUrl,
}: { userId: string; caseId: string; amountCents: number; successUrl: string;
```

```
cancelUrl: string; }) {
  const caseRec = await prisma.case.findUnique({ where: { id: caseId } });
  if (!caseRec || caseRec.status !== 'LIVE') throw new Error('Case not live');

  const feeBps = Number(process.env.STRIPE_PLATFORM_FEE_BPS || '500');
  const applicationFeeAmount = Math.floor((amountCents * feeBps) / 10000);

  // Optional: route funds to organizer connected account later; start with
platform account balance (pseudo-escrow)
  const session = await stripe.checkout.sessions.create({
    mode: 'payment',
    success_url: successUrl,
    cancel_url: cancelUrl,
    line_items: [{ price_data: { currency: 'usd', product_data: { name:
caseRec.title }, unit_amount: amountCents }, quantity: 1 }],
    payment_intent_data: {
      metadata: { userId, caseId },
      application_fee_amount: applicationFeeAmount,
    },
  });

  await prisma.pledge.create({ data: { userId, caseId, amountCents, provider:
'stripe', providerRef: session.id, status: 'PENDING' } });
  return session.url!;
}
```

6.2 API route to create pledge intent

`src/app/api/pledges/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { createCheckoutSession } from '@/lib/payments';
import jwt from 'jsonwebtoken';
import { cookies } from 'next/headers';

export async function POST(req: NextRequest) {
  const cookie = cookies().get(process.env.SESSION_COOKIE_NAME!);
  if (!cookie) return NextResponse.json({ error: 'Unauthorized' }, { status:
401 });
  const payload = jwt.verify(cookie.value, process.env.JWT_SECRET!) as any;

  const { caseId, amountCents } = await req.json();
  const url = await createCheckoutSession({
    userId: payload.sub,
    caseId,
    amountCents,
```

```
    successUrl: `${process.env.NEXT_PUBLIC_APP_URL}/dashboard/pledges`,
    cancelUrl: `${process.env.NEXT_PUBLIC_APP_URL}/cases/${caseId}`,
  });
  return NextResponse.json({ url });
}
```

6.3 Webhook (idempotent)

`src/lib/webhook.ts`

```
import crypto from 'crypto';

export function hashPayload(buf: Buffer) {
  return crypto.createHash('sha256').update(buf).digest('hex');
}
```

`src/app/api/webhooks/stripe/route.ts`

```
import { NextRequest, NextResponse } from 'next/server';
import { stripe } from '@/lib/payments';
import { prisma } from '@/lib/db';
import { hashPayload } from '@/lib/webhook';

export const config = { api: { bodyParser: false } } as any; // we need raw body

export async function POST(req: NextRequest) {
  const secret = process.env.STRIPE_WEBHOOK_SECRET!;
  const sig = req.headers.get('stripe-signature');
  const buf = Buffer.from(await req.arrayBuffer());

  let event;
  try {
    event = stripe.webhooks.constructEvent(buf, sig!, secret);
  } catch (err: any) {
    return NextResponse.json({ error: `Invalid signature: ${err.message}` }, {
status: 400 });
  }

  const idempotencyKey = (event as any).id as string;
  const payloadHash = hashPayload(buf);

  const exists = await prisma.webhookEvent.findUnique({ where: {
idempotencyKey } });
  if (exists) return NextResponse.json({ ok: true, duplicate: true });
```

```
  await prisma.webhookEvent.create({ data: {
    provider: 'stripe',
    eventType: event.type,
    payloadHash,
    idempotencyKey,
    success: false,
  }});

  try {
    switch (event.type) {
      case 'checkout.session.completed': {
        const sess = event.data.object as any;
        const intentId = sess.payment_intent as string;
        const pi = await stripe.paymentIntents.retrieve(intentId);

        const caseId = pi.metadata?.caseId;
        const userId = pi.metadata?.userId;
        const amount = pi.amount ?? 0;

        await prisma.pledge.updateMany({
          where: { providerRef: sess.id },
          data: { status: 'SUCCEEDED' },
        });

        await prisma.case.update({
          where: { id: caseId },
          data: { raisedCents: { increment: amount } },
        });

        await prisma.escrowLedger.create({
          data: { caseId, deltaCents: amount, reason: 'pledge_captured',
providerRef: intentId },
        });
        break;
      }
      case 'charge.dispute.created': {
        // mark pledge as DISPUTED
        break;
      }
      default:
        break;
    }

    await prisma.webhookEvent.update({
      where: { idempotencyKey },
      data: { success: true, processedAt: new Date() },
    });
  } catch (err: any) {
```

```
    await prisma.webhookEvent.update({ where: { idempotencyKey }, data: {
success: false, error: err.message } });
    return NextResponse.json({ error: 'Processing error' }, { status: 500 });
  }

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

Escrow releases: implement a secure admin action that moves funds by creating a Stripe **transfer** to an organizer's connected account when milestones are reached or case settles, and record a negative `EscrowLedger` delta.

---

7) Netlify configuration (Next.js on Netlify)

`netlify.toml`

```
[build]
  command = "npm run build"
  publish = ".next"

[[plugins]]
  package = "@netlify/plugin-nextjs"

[functions]
  external_node_modules = ["@prisma/client", "prisma", "stripe", "resend",
"speakeasy", "qrcode", "mysql2"]
  node_bundler = "esbuild"

[headers]
  for = "/*"
    [headers.values]
    X-Frame-Options = "DENY"
    X-Content-Type-Options = "nosniff"
    Referrer-Policy = "strict-origin-when-cross-origin"
```

In Netlify dashboard: set the environment variables from `.env.example`. Also set `SESSION_COOKIE_DOMAIN=.gofundalawsuit.com` and `SESSION_COOKIE_SECURE=true` in production.

---

8) Minimal placeholders for key pages

`src/app/(public)/page.tsx`

```
export default function Home() {
  return (
    <main className="max-w-5xl mx-auto p-8">
      <h1 className="text-3xl font-bold">class-action</h1>
      <p className="mt-2">Crowdfunding for lawsuits. Build a fair fight in
court.</p>
    </main>
  );
}
```

`src/components/PledgeWidget.tsx`

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

export default function PledgeWidget({ caseId }: { caseId: string }) {
  const [amount, setAmount] = useState(2500);
  const submit = async () => {
    const res = await fetch('/api/pledges', { method: 'POST', body:
JSON.stringify({ caseId, amountCents: amount }), headers: { 'Content-Type':
'application/json' } });
    const data = await res.json();
    if (data.url) window.location.href = data.url;
  };
  return (
    <div className="border rounded p-4">
      <label className="block text-sm">Amount (cents)</label>
      <input className="border px-2 py-1" value={amount} onChange={e =>
setAmount(Number(e.target.value))} />
      <button className="ml-2 px-3 py-1 border" onClick={submit}>Pledge</button>
    </div>
  );
}
```

---

9) Security must-dos before going live

- Use **HTTPS** everywhere and set `secure` cookie flag in production.
- Set a strict **CSP** (Content Security Policy) via headers or `next-safe`.
- Enable **rate limiting** for `/api/auth/*`, `/api/pledges`, `/api/webhooks/*`.
- Validate **all** inputs with **Zod** and re-check permissions on the server.
- Scan and validate file uploads; do not ever serve raw user uploads without a signed URL layer.
- Add full audit logging for admin actions and money movements.
- Conduct legal review for **jurisdiction gating** and required disclosures. ```