## T3 Style Init Checklist 0) Create the app

# 0.1 scaffold

pnpm dlx create-next-app@latest fundraise-web \

--ts --eslint --tailwind --app --src-dir --import-alias "@/\*" --use-pnpm

cd fundraise-web

# 0.2 deps (app)

pnpm add @clerk/nextjs stripe cloudinary @mux/mux-node @mux/mux-player-react \

typesense pusher @pusher/pusher-js postmark twilio zod \

@t3-oss/env-nextjs @growthbook/growthbook @growthbook/growthbook-react

# 0.3 orm & tooling

pnpm add @prisma/client

pnpm add -D prisma

**1) Environment variables**

Create **.env.local** (fill with your keys later):

# App

NEXT\_PUBLIC\_APP\_URL=http://localhost:3000

NODE\_ENV=development

# Database (Postgres)

DATABASE\_URL="postgresql://USER:PASSWORD@HOST:PORT/DB?sslmode=require"

# Clerk (Auth)

NEXT\_PUBLIC\_CLERK\_PUBLISHABLE\_KEY=pk\_test\_...

CLERK\_SECRET\_KEY=sk\_test\_...

# Stripe (Payments + Connect + Webhooks)

STRIPE\_SECRET\_KEY=sk\_live\_or\_test\_...

NEXT\_PUBLIC\_STRIPE\_PUBLISHABLE\_KEY=pk\_live\_or\_test\_...

STRIPE\_WEBHOOK\_SECRET=whsec\_... # for /api/stripe/webhook

STRIPE\_CONNECT\_WEBHOOK\_SECRET=whsec\_... # optional if you use a separate endpoint

# Cloudinary (Images)

CLOUDINARY\_CLOUD\_NAME=your\_cloud

CLOUDINARY\_API\_KEY=...

CLOUDINARY\_API\_SECRET=...

# Mux (Video)

MUX\_TOKEN\_ID=...

MUX\_TOKEN\_SECRET=...

# Typesense (Search)

TYPESENSE\_HOST=xxx.a1.typesense.net

TYPESENSE\_PROTOCOL=https

TYPESENSE\_PORT=443

TYPESENSE\_API\_KEY=tskey\_...

# Pusher (Realtime) — or use Ably if you prefer

PUSHER\_APP\_ID=...

PUSHER\_KEY=...

PUSHER\_SECRET=...

PUSHER\_CLUSTER=us2

NEXT\_PUBLIC\_PUSHER\_KEY=${PUSHER\_KEY}

NEXT\_PUBLIC\_PUSHER\_CLUSTER=${PUSHER\_CLUSTER}

# Postmark (Email)

POSTMARK\_SERVER\_TOKEN=...

# Twilio (SMS/Voice)

TWILIO\_ACCOUNT\_SID=AC...

TWILIO\_AUTH\_TOKEN=...

TWILIO\_MESSAGING\_SERVICE\_SID=MG...

# GrowthBook (Feature flags)

NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY=gb\_...

GROWTHBOOK\_API\_HOST=https://cdn.growthbook.io

# Optional: Cloudflare Turnstile

NEXT\_PUBLIC\_TURNSTILE\_SITE\_KEY=...

TURNSTILE\_SECRET\_KEY=...

**2) Prisma schema**

pnpm dlx prisma init --datasource-provider postgresql

Replace **prisma/schema.prisma** with:

generator client {

provider = "prisma-client-js"

}

datasource db {

provider = "postgresql"

url = env("DATABASE\_URL")

}

model User {

id String @id @default(cuid())

clerkId String @unique

email String @unique

displayName String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

// Example relations

campaigns Campaign[]

donations Donation[]

}

model Campaign {

id String @id @default(cuid())

slug String @unique

title String

city String

state String

goalCents Int

coverUrl String

status String // draft|live|ended|not\_funded|suspended

isAon Boolean @default(false)

startDate DateTime

endDate DateTime

organizerId String

organizer User @relation(fields: [organizerId], references: [id])

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

donations Donation[]

}

model Donation {

id String @id @default(cuid())

campaignId String

donorId String

amountCents Int

tipCents Int @default(0)

isAnonymous Boolean @default(false)

hideName Boolean @default(true)

intentId String? // Stripe payment\_intent id

createdAt DateTime @default(now())

campaign Campaign @relation(fields: [campaignId], references: [id])

donor User @relation(fields: [donorId], references: [id])

}

Generate & push:

pnpm prisma generate

pnpm dlx prisma db push

Create **src/server/db.ts**:

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

const globalForPrisma = global as unknown as { prisma: PrismaClient };

export const prisma =

globalForPrisma.prisma ?? new PrismaClient({ log: ["error", "warn"] });

if (process.env.NODE\_ENV !== "production") globalForPrisma.prisma = prisma;

**3) Env validation (T3‑style)**

Create **src/env.ts**:

import { createEnv } from "@t3-oss/env-nextjs";

import { z } from "zod";

export const env = createEnv({

server: {

DATABASE\_URL: z.string().url(),

CLERK\_SECRET\_KEY: z.string(),

STRIPE\_SECRET\_KEY: z.string(),

STRIPE\_WEBHOOK\_SECRET: z.string(),

CLOUDINARY\_API\_KEY: z.string(),

CLOUDINARY\_API\_SECRET: z.string(),

MUX\_TOKEN\_ID: z.string(),

MUX\_TOKEN\_SECRET: z.string(),

TYPESENSE\_API\_KEY: z.string(),

TYPESENSE\_HOST: z.string(),

TYPESENSE\_PROTOCOL: z.string().default("https"),

TYPESENSE\_PORT: z.coerce.number().default(443),

PUSHER\_APP\_ID: z.string(),

PUSHER\_SECRET: z.string(),

POSTMARK\_SERVER\_TOKEN: z.string(),

TWILIO\_ACCOUNT\_SID: z.string(),

TWILIO\_AUTH\_TOKEN: z.string(),

TWILIO\_MESSAGING\_SERVICE\_SID: z.string(),

GROWTHBOOK\_API\_HOST: z.string(),

},

client: {

NEXT\_PUBLIC\_APP\_URL: z.string().url(),

NEXT\_PUBLIC\_CLERK\_PUBLISHABLE\_KEY: z.string(),

NEXT\_PUBLIC\_STRIPE\_PUBLISHABLE\_KEY: z.string(),

NEXT\_PUBLIC\_PUSHER\_KEY: z.string(),

NEXT\_PUBLIC\_PUSHER\_CLUSTER: z.string(),

NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY: z.string(),

NEXT\_PUBLIC\_TURNSTILE\_SITE\_KEY: z.string().optional(),

},

experimental\_\_runtimeEnv: {

NEXT\_PUBLIC\_APP\_URL: process.env.NEXT\_PUBLIC\_APP\_URL,

NEXT\_PUBLIC\_CLERK\_PUBLISHABLE\_KEY: process.env.NEXT\_PUBLIC\_CLERK\_PUBLISHABLE\_KEY,

NEXT\_PUBLIC\_STRIPE\_PUBLISHABLE\_KEY: process.env.NEXT\_PUBLIC\_STRIPE\_PUBLISHABLE\_KEY,

NEXT\_PUBLIC\_PUSHER\_KEY: process.env.NEXT\_PUBLIC\_PUSHER\_KEY,

NEXT\_PUBLIC\_PUSHER\_CLUSTER: process.env.NEXT\_PUBLIC\_PUSHER\_CLUSTER,

NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY: process.env.NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY,

NEXT\_PUBLIC\_TURNSTILE\_SITE\_KEY: process.env.NEXT\_PUBLIC\_TURNSTILE\_SITE\_KEY,

},

});

**4) Core libs (Stripe, Clerk, Pusher, Typesense, Cloudinary, Mux, GrowthBook)**

Create **src/lib/stripe.ts**:

import Stripe from "stripe";

import { env } from "@/env";

export const stripe = new Stripe(env.STRIPE\_SECRET\_KEY, {

apiVersion: "2024-06-20",

});

**src/lib/pusher.ts**:

import PusherServer from "pusher";

import PusherClient from "@pusher/pusher-js";

import { env } from "@/env";

export const pusherServer = new PusherServer({

appId: env.PUSHER\_APP\_ID,

key: process.env.NEXT\_PUBLIC\_PUSHER\_KEY!,

secret: env.PUSHER\_SECRET,

cluster: process.env.NEXT\_PUBLIC\_PUSHER\_CLUSTER!,

useTLS: true,

});

export const pusherClient = new PusherClient(

process.env.NEXT\_PUBLIC\_PUSHER\_KEY!,

{ cluster: process.env.NEXT\_PUBLIC\_PUSHER\_CLUSTER! }

);

**src/lib/typesense.ts**:

import Typesense from "typesense";

import { env } from "@/env";

export const typesense = new Typesense.Client({

nodes: [

{

host: env.TYPESENSE\_HOST,

port: env.TYPESENSE\_PORT,

protocol: env.TYPESENSE\_PROTOCOL as "http" | "https",

},

],

apiKey: env.TYPESENSE\_API\_KEY,

connectionTimeoutSeconds: 5,

});

**src/lib/cloudinary.ts**:

import { v2 as cloudinary } from "cloudinary";

import { env } from "@/env";

cloudinary.config({

cloud\_name: process.env.CLOUDINARY\_CLOUD\_NAME!,

api\_key: env.CLOUDINARY\_API\_KEY,

api\_secret: env.CLOUDINARY\_API\_SECRET,

secure: true,

});

export { cloudinary };

**src/lib/mux.ts**:

import { Mux } from "@mux/mux-node";

import { env } from "@/env";

export const mux = new Mux({ tokenId: env.MUX\_TOKEN\_ID, tokenSecret: env.MUX\_TOKEN\_SECRET });

**src/lib/growthbook.tsx**:

"use client";

import { GrowthBook, GrowthBookProvider } from "@growthbook/growthbook-react";

const gb = new GrowthBook({

apiHost: process.env.NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY ? process.env.GROWTHBOOK\_API\_HOST : undefined,

clientKey: process.env.NEXT\_PUBLIC\_GROWTHBOOK\_CLIENT\_KEY,

enableDevMode: process.env.NODE\_ENV !== "production",

});

export function GBProvider({ children }: { children: React.ReactNode }) {

return <GrowthBookProvider growthbook={gb}>{children}</GrowthBookProvider>;

}

**5) App layout & auth guard**

**src/app/layout.tsx**:

import "./globals.css";

import { ClerkProvider } from "@clerk/nextjs";

import { GBProvider } from "@/lib/growthbook";

export const metadata = { title: "Fundraise", description: "Campaigns & donations" };

export default function RootLayout({ children }: { children: React.ReactNode }) {

return (

<ClerkProvider>

<html lang="en">

<body>

<GBProvider>{children}</GBProvider>

</body>

</html>

</ClerkProvider>

);

}

**src/middleware.ts** (protect portal/admin routes):

import { clerkMiddleware, createRouteMatcher } from "@clerk/nextjs/server";

const isProtected = createRouteMatcher([

"/portal(.\*)",

"/admin(.\*)",

]);

export default clerkMiddleware((auth, req) => {

if (isProtected(req)) auth().protect();

});

export const config = {

matcher: ["/((?!\_next|static|.\*\\..\*|api/public).\*)", "/"],

};

**6) Example pages**

**Public campaign page** — **src/app/c/[slug]/page.tsx**:

import { prisma } from "@/server/db";

import Image from "next/image";

import Link from "next/link";

export default async function CampaignPage({ params }: { params: { slug: string } }) {

const campaign = await prisma.campaign.findUnique({ where: { slug: params.slug } });

if (!campaign) return <div className="p-8">Not found</div>;

const pct = Math.min(100, Math.round(

((await prisma.donation.aggregate({ where: { campaignId: campaign.id }, \_sum: { amountCents: true } })).\_sum.amountCents || 0) \* 100 / campaign.goalCents

));

return (

<main className="mx-auto max-w-5xl p-6 space-y-6">

<h1 className="text-3xl font-semibold">{campaign.title}</h1>

<div className="relative aspect-video w-full overflow-hidden rounded-md bg-gray-100">

<Image src={campaign.coverUrl} alt={campaign.title} fill className="object-cover" />

</div>

<div className="rounded-md border p-4">

<div className="mb-2 flex items-center justify-between">

<span>${(campaign.goalCents/100).toLocaleString()} goal</span>

<span>{pct}% funded</span>

</div>

<div className="h-3 w-full rounded bg-gray-200">

<div className="h-full rounded bg-gray-800" style={{ width: `${pct}%` }} />

</div>

<Link href={`/c/${params.slug}/donate`} className="mt-4 inline-block rounded bg-black px-4 py-2 text-white">

Donate

</Link>

</div>

</main>

);

}

**Portal home (protected)** — **src/app/portal/page.tsx**:

import { currentUser } from "@clerk/nextjs/server";

export default async function Portal() {

const user = await currentUser();

return (

<div className="p-8">

<h1 className="text-2xl">Welcome{user?.firstName ? `, ${user.firstName}` : ""}</h1>

<p className="text-gray-600">This is your dashboard.</p>

</div>

);

}

**7) Stripe: create PaymentIntent + webhook**

**src/app/api/donate/create-intent/route.ts**:

import { NextResponse } from "next/server";

import { stripe } from "@/lib/stripe";

import { z } from "zod";

const Body = z.object({ amount: z.number().int().min(500) /\* cents \*/, campaignId: z.string() });

export async function POST(req: Request) {

const json = await req.json();

const { amount, campaignId } = Body.parse(json);

const paymentIntent = await stripe.paymentIntents.create({

amount,

currency: "usd",

automatic\_payment\_methods: { enabled: true },

metadata: { campaignId },

});

return NextResponse.json({ clientSecret: paymentIntent.client\_secret });

}

**src/app/api/stripe/webhook/route.ts**:

import { headers } from "next/headers";

import { stripe } from "@/lib/stripe";

import { env } from "@/env";

import { prisma } from "@/server/db";

export const runtime = "nodejs"; // use Node runtime for raw body

export const dynamic = "force-dynamic";

export async function POST(req: Request) {

const sig = headers().get("stripe-signature") as string;

const rawBody = await req.text();

let event;

try {

event = stripe.webhooks.constructEvent(rawBody, sig, env.STRIPE\_WEBHOOK\_SECRET);

} catch (err: any) {

return new Response(`Webhook Error: ${err.message}`, { status: 400 });

}

switch (event.type) {

case "payment\_intent.succeeded": {

const pi = event.data.object as any;

const campaignId = pi.metadata?.campaignId as string | undefined;

if (campaignId) {

const amountCents = pi.amount\_received as number;

// TODO: resolve donor userId from session / attach at client when confirming

await prisma.donation.create({

data: {

campaignId,

donorId: "anonymous", // replace with actual user id

amountCents,

tipCents: 0,

intentId: pi.id,

},

});

}

break;

}

case "charge.refunded":

case "charge.dispute.created":

// TODO: update donation, ledger, notifications

break;

default:

break;

}

return new Response("ok");

}

**8) Stripe Connect (organizer onboarding)**

**src/app/api/stripe/connect/onboard/route.ts**:

import { stripe } from "@/lib/stripe";

import { NextResponse } from "next/server";

export async function POST() {

const account = await stripe.accounts.create({ type: "express", country: "US", business\_type: "individual" });

const link = await stripe.accountLinks.create({

account: account.id,

refresh\_url: `${process.env.NEXT\_PUBLIC\_APP\_URL}/portal/connect/refresh`,

return\_url: `${process.env.NEXT\_PUBLIC\_APP\_URL}/portal/connect/return`,

type: "account\_onboarding",

});

return NextResponse.json({ url: link.url, accountId: account.id });

}

**9) Cloudinary signed upload**

**src/app/api/cloudinary/sign/route.ts**:

import { cloudinary } from "@/lib/cloudinary";

import { NextResponse } from "next/server";

export async function POST() {

const ts = Math.round(Date.now() / 1000);

const signature = cloudinary.utils.api\_sign\_request({ timestamp: ts }, process.env.CLOUDINARY\_API\_SECRET!);

return NextResponse.json({

cloudName: process.env.CLOUDINARY\_CLOUD\_NAME,

apiKey: process.env.CLOUDINARY\_API\_KEY,

timestamp: ts,

signature,

});

}

Add Cloudinary domains to **next.config.js**:

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

images: { remotePatterns: [{ protocol: "https", hostname: "res.cloudinary.com" }] },

};

module.exports = nextConfig;

**10) Pusher: basic DM send endpoint**

**src/app/api/dm/send/route.ts**:

import { pusherServer } from "@/lib/pusher";

import { NextResponse } from "next/server";

import { z } from "zod";

const Body = z.object({ threadId: z.string(), text: z.string().min(1).max(2000) });

export async function POST(req: Request) {

const { threadId, text } = Body.parse(await req.json());

await pusherServer.trigger(`thread-${threadId}`, "message:new", { text, ts: Date.now() });

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

}

Client example (somewhere in your DM UI):

"use client";

import { pusherClient } from "@/lib/pusher";

import { useEffect } from "react";

export function useThread(threadId: string, onMsg: (m: any) => void) {

useEffect(() => {

const ch = pusherClient.subscribe(`thread-${threadId}`);

ch.bind("message:new", onMsg);

return () => {

ch.unbind\_all(); pusherClient.unsubscribe(`thread-${threadId}`);

};

}, [threadId, onMsg]);

}

**11) Typesense indexing stub**

**src/app/api/search/index-campaign/route.ts**:

import { typesense } from "@/lib/typesense";

import { prisma } from "@/server/db";

export async function POST() {

const campaigns = await prisma.campaign.findMany({ take: 100 });

const collection = "campaigns";

try {

await typesense.collections(collection).retrieve();

} catch {

await typesense.collections().create({

name: collection,

fields: [

{ name: "id", type: "string" },

{ name: "title", type: "string" },

{ name: "city", type: "string" },

{ name: "state", type: "string", facet: true },

{ name: "isAon", type: "bool", facet: true },

],

default\_sorting\_field: "id",

});

}

const docs = campaigns.map(c => ({ id: c.id, title: c.title, city: c.city, state: c.state, isAon: c.isAon }));

await typesense.collections(collection).documents().import(docs, { action: "upsert" });

return new Response("ok");

}

**12) Notifications: Postmark + Twilio helpers**

**src/lib/postmark.ts**:

import { ServerClient } from "postmark";

export const postmark = new ServerClient(process.env.POSTMARK\_SERVER\_TOKEN!);

export async function sendReceiptEmail(to: string, subject: string, htmlBody: string) {

return postmark.sendEmail({ From: "noreply@yourdomain.com", To: to, Subject: subject, HtmlBody: htmlBody });

}

**src/lib/twilio.ts**:

import twilio from "twilio";

const client = twilio(process.env.TWILIO\_ACCOUNT\_SID!, process.env.TWILIO\_AUTH\_TOKEN!);

export async function sendSms(toE164: string, body: string) {

return client.messages.create({ to: toE164, messagingServiceSid: process.env.TWILIO\_MESSAGING\_SERVICE\_SID!, body });

}

**13) Trigger.dev (scheduled jobs & nudges)**

Install:

pnpm add @trigger.dev/sdk @trigger.dev/nextjs

**src/app/api/trigger/route.ts**:

export { handler as GET, handler as POST } from "@trigger.dev/nextjs";

**src/jobs/nudges.ts**:

import { client } from "@trigger.dev/sdk";

import { sendSms } from "@/lib/twilio";

client.defineJob({

id: "reward-nudge-14d",

name: "Reward Nudge 14 Days",

version: "1.0.0",

trigger: {

cron: { cron: "0 15 \* \* \*" }, // daily 15:00 UTC

},

run: async (payload, io, ctx) => {

// TODO: query pending rewards older than 14d; send SMS/email

await sendSms("+15555550123", "Reminder: You have unfulfilled rewards to process.");

},

});

In your Trigger.dev dashboard, point the endpoint to /api/trigger.

**14) Admin (react‑admin minimal mount)**

pnpm add react-admin ra-data-simple-rest @mui/material @emotion/react @emotion/styled

**src/app/admin/page.tsx**:

"use client";

import { Admin, Resource, ListGuesser } from "react-admin";

import simpleRestProvider from "ra-data-simple-rest";

export default function AdminPage() {

const dataProvider = simpleRestProvider("/api/admin"); // build endpoints later

return (

<div style={{ height: "100vh" }}>

<Admin dataProvider={dataProvider}>

<Resource name="users" list={ListGuesser} />

<Resource name="campaigns" list={ListGuesser} />

</Admin>

</div>

);

}

**15) Package scripts**

Update **package.json**:

{

"scripts": {

"dev": "next dev",

"build": "next build",

"start": "next start",

"typecheck": "tsc --noEmit",

"lint": "next lint",

"prisma:generate": "prisma generate",

"prisma:push": "prisma db push",

"seed": "tsx prisma/seed.ts"

}

}

Optional **postinstall**:

"postinstall": "prisma generate"

**16) Tailwind & global styles**

Already created by create-next-app --tailwind. You can start using utility classes immediately.

**17) Run it**

pnpm prisma:push

pnpm dev

Open [**http://localhost:3000**](http://localhost:3000).

**What you have now**

* **Auth** (Clerk provider + protected routes via middleware).
* **DB** (Prisma + Postgres + seed‑ready).
* **Core libs** wired (Stripe, Cloudinary, Mux, Typesense, Pusher, Postmark, Twilio, GrowthBook).
* **API stubs** for donations (PaymentIntent), Stripe webhooks, Cloudinary signed uploads, DM send, Typesense indexing, Connect onboarding.
* **Jobs** entry for Trigger.dev (14‑day reward nudge example).
* **Admin** mount stub (react‑admin) ready to connect to your REST endpoints.
* **Public campaign page** + **portal** placeholder.

**Next 1–2 days (fill‑ins)**

* Attach **actual user id** on donation (donor must be logged in; pass donorId in metadata or resolve server‑side).
* Expand Prisma models to match our Section 12/14 data model (RewardTier, RewardClaim, Team, Payout, Ledger).
* Implement **Stripe Connect** flows end‑to‑end (account creation, account links, transfer on campaign end+hold).
* Add **Cloudinary widget** or your own upload UI for campaign images/video (Mux direct uploads).
* Build **search** UI bound to Typesense (facets/sorts per Section 16).
* Flesh out **Admin** endpoints (/api/admin/\*) and RBAC with CASL.
* Wire **Postmark** templates and Twilio quiet‑hours logic (Section 18).
* Add **report abuse** endpoint + moderation queue (Section 20).