**Flight Booking Website — SDLC Plan**

*A pragmatic, solo‑developer SDLC tailored for a production MVP you can deploy.*

**Phase 1 — Requirements & Planning**

**1.1 Vision**

Build a lean, production‑ready web app where customers can search flights, book, pay, and receive e‑tickets. Use an aggregator (Amadeus Self‑Service) for content and ticketing; Stripe for payments. Deploy on managed hosting to keep ops light for a solo developer.

**1.2 Stakeholders**

* Business owners (family)
* Solo developer (you)
* End users (customers)
* External providers: Amadeus, Stripe, Email (Resend/SendGrid), Hosting (Vercel), DB (Neon/Railway)

**1.3 Scope (MVP)**

* Search flights (one‑way/round trip) ↔ Amadeus Offers Search
* Offer select → re‑price/verify
* Passenger details → optional seat/baggage notes (no live seat map in MVP)
* Card payment via Stripe (you as MoR) using Payment Intents + 3DS
* Order Create + Ticketing ↔ Amadeus (PNR + e‑ticket numbers)
* Confirmation page + email with itinerary and ticket numbers
* User accounts (view bookings/history, download receipts)
* Basic admin view (search bookings, cancel/refund)

**1.4 Out of Scope (v1)**

* Multi‑city searches; complex ancillaries; loyalty; live seat maps; B2B agent tools; localized currencies beyond EUR; gift cards; mobile apps.

**1.5 Assumptions**

* Single market launch (EU, prices in EUR)
* One supplier: Amadeus Self‑Service APIs (test → production)
* Merchant of Record: your business (Stripe)
* Email sender domain available (e.g., mail.yourdomain.com)

**1.6 Constraints**

* Solo developer capacity
* PCI scope minimized (Stripe Elements)
* Supplier API quotas & pricing
* GDPR compliance for customer PII

**1.7 Success Criteria / KPIs**

* ✅ Customers can complete a booking end‑to‑end (search → pay → ticket) without human help
* P95 page response < 600 ms (server) under 20 RPS
* Payment success rate ≥ 90% after 3DS challenges
* Error budget: availability ≥ 99.9% (month)
* Refund/cancel handled within the app (void same‑day if possible)

**1.8 High‑Level Risks & Mitigations**

* **Supplier mismatch / price change** → Always re‑price before payment; handle sold‑out gracefully.
* **Paid but not ticketed** → Use idempotency; on ticketing failure, auto‑void and refund via webhook.
* **PII exposure** → Encrypt sensitive fields; restrict admin access; audit trail.
* **Quota/rate limits** → Cache search results; exponential backoff; circuit breaker.

**1.9 Milestones (sequence, no dates)**

1. Scaffold + Auth + DB
2. Search & Re‑price
3. Checkout (passengers → review)
4. Stripe Payment + Webhooks
5. Order Create + Ticketing
6. User Portal & Emails
7. Admin + Refunds + Monitoring

**Phase 2 — System Design**

**2.1 Architecture Overview**

* **Frontend**: Next.js 14 (App Router, TS, Tailwind)
* **Backend**: Next.js Route Handlers (server actions) + background jobs via webhooks
* **DB**: Postgres + Prisma ORM
* **Payments**: Stripe (Payment Intents, Checkout/Elements)
* **Supplier**: Amadeus Self‑Service (Flight Offers Search, Price, Orders)
* **Email**: Resend/SendGrid
* **Hosting**: Vercel (app), Neon/Railway (Postgres)

**2.2 Components**

* Web UI: Search, Results, Checkout, Confirmation, Account, Admin
* API: /api/search, /api/offers/:id/reprice, /api/bookings, /api/payments/intent, /api/stripe/webhook, /api/orders/ticket, /api/bookings/:id (GET/DELETE)
* Webhooks: Stripe events (payment\_intent.succeeded/failed); supplier callbacks if provided
* Services: Email service, PDF ticket/receipt generator (optional v1)

**2.3 Data Model (initial)**

* **User**(id, email, name, role, createdAt)
* **Booking**(id, userId, status[pending|confirmed|canceled|refunded], currency, totalCents, pnr, itinerary JSON, createdAt)
* **Payment**(id, bookingId, provider, providerId, amountCents, currency, status)
* **Ticket**(id, bookingId, airline, ticketNumber, pdfUrl)
* **Passenger**(id, bookingId, firstName, lastName, dob, docType, docNumber, nationality, seatNote, baggageNote)
* **SearchCache**(hash, resultsJSON, ttl)
* **AuditLog**(id, actor, action, meta, createdAt)

**2.4 Sequence (happy path)**

1. Search → /api/search → Amadeus Offers
2. Select → /api/offers/:id/reprice
3. Collect passengers → create Booking(pending)
4. Create Stripe PaymentIntent → client confirms
5. Stripe payment\_intent.succeeded → webhook → call Supplier **Order Create + Ticket**
6. Update Booking(confirmed) + save PNR/ticket numbers → send email → show success

**2.5 Error/Compensation Flows**

* Payment fails → keep booking pending, allow retry
* Ticketing fails after payment → auto refund (Stripe) + set booking canceled
* Price change on re‑price → show new price; require user confirm

**2.6 Non‑Functional Requirements**

* **Security**: No card data stored; PII encryption at rest; RBAC for admin
* **Performance**: Server P95 < 600 ms; search results cached 5–10 min
* **Reliability**: Idempotent endpoints; retry with backoff; at‑least‑once webhooks
* **Observability**: Structured logs, Sentry for errors, tracing on critical flows
* **Compliance**: GDPR (data retention policy; delete on request)

**Phase 3 — Implementation Plan**

**3.1 Backlog (MVP tasks)**

* **Scaffold** Next.js app; Tailwind; layout
* **Prisma** schema; DB migrations; seed admin user
* **Auth** with NextAuth (email/password)
* **Search API** → Amadeus Offers + caching
* **Results page** with offer cards; select & re‑price
* **Checkout**: passenger form (Zod validation), review page
* **Payments**: PaymentIntent endpoint + client Elements; 3DS handling
* **Webhook**: verify signature; on success call Supplier Order Create + Ticket
* **Persistence**: save PNR + ticket numbers; update statuses
* **Email**: confirmation with itinerary/tickets
* **Portal**: My bookings list/detail; download receipt
* **Admin**: bookings table; cancel/refund action
* **Monitoring**: Sentry; basic health endpoint

**3.2 Definition of Done (per feature)**

* Unit tests for utilities/services
* Integration tests for API routes (mock external)
* E2E happy path (Playwright) covering search→ticket
* Logs without PII; env vars documented; runbook updated

**Phase 4 — Verification & Testing**

**4.1 Test Types**

* **Unit** (formatters, validators, pricing calculations)
* **Contract** (mocks vs Amadeus/Stripe schemas)
* **Integration** (API routes with mocked providers)
* **E2E** (Playwright: full booking flow)
* **Security** (headers, authZ, rate limits, secrets)

**4.2 Test Data & Environments**

* **Sandbox** keys for Stripe + Amadeus
* Seed data for users and example offers
* Feature flags to stub ticketing in dev

**4.3 Acceptance Criteria (MVP)**

* A new user can:
  1. Search and see offers
  2. Select an offer and see final price
  3. Enter passengers and pay with a test card
  4. Receive confirmation with PNR + at least one ticket number
  5. View booking in portal

**Phase 5 — Deployment & Release**

**5.1 Environments**

* **Dev** (sandbox keys) → **Prod** (live keys)

**5.2 Secrets & Config**

* AMADEUS\_API\_KEY/SECRET, STRIPE\_SECRET, STRIPE\_WEBHOOK\_SECRET, DATABASE\_URL, EMAIL\_API\_KEY, BASE\_URL

**5.3 Deployment Steps**

1. Run DB migrations
2. Deploy app to Vercel
3. Configure Stripe webhook → /api/stripe/webhook
4. Configure domain + email sender
5. Smoke test: health, search, test booking

**5.4 Go‑Live Checklist**

* Monitoring dashboards live
* Error rate < 1% on API
* Rollback plan (revert to last deploy)
* Support contact and runbook

**Phase 6 — Operations & Maintenance**

* Incident response runbook (payment ok, ticketing failed → auto refund)
* Data retention & GDPR deletion requests
* Rotating keys and backups
* Quarterly dependency updates

**Phase 7 — Documentation & Training**

* README (setup, run, deploy)
* API docs (OpenAPI) for internal routes
* Admin guide (refunds/cancel)

**Open Decisions (to confirm when ready)**

* Branding/name and domain
* Exact refund policy (fees, windows)
* Supported markets/currencies beyond EUR
* Seat map integration timeline (post‑MVP)

**Next Actions (Start Here)**

1. Create repos and environment (.env spec)
2. Set up Neon/Railway Postgres; run initial Prisma migration
3. Set up Stripe + Amadeus sandbox accounts
4. Implement /api/search + Results page with mock data → swap to real API
5. Build checkout + Stripe; wire webhook and stub ticketing
6. Wire Amadeus Order Create + Ticketing; finalize happy path