■ Smart Invoice Chaser – Developer Prompt Pack (for ChatGPT/Codex)

Copy–paste these prompts to generate production-grade code for each part of the MVP. Each prompt includes objectives, acceptance criteria, and outputs.

1) Backend Scaffold (FastAPI + PostgreSQL)

```
SYSTEM:
You are a senior backend engineer. Write robust, secure, well-tested code. Use Python 3.11, FastAPI,

USER:
Generate a new FastAPI project "sic_backend" with the following:
- Modules: auth (JWT), invoices, clients, reminders, integrations (stripe), analytics.
- SQLAlchemy models and Alembic migrations for Users, Clients, Invoices, Reminders, Events (audit).
- Pydantic schemas (read/write) with field validation.
- Settings via environment variables (.env example). Use async SQLAlchemy + asyncpg.
- REST endpoints with pagination, filtering, and error handling (HTTPException + problem+json).
- Structured logging (uvicorn + loguru). CORS configured for mobile/web app.
- Include docker-compose.yml (app + Postgres + pgadmin) and a Makefile (run, test, fmt, lint, migrat
- Unit tests with pytest + httpx AsyncClient + factory-boy + coverage config.

ACCEPTANCE CRITERIA:
- Running `docker compose up` starts DB and app on :8000 and passes `pytest -q` inside container.
- `alembic upgrade head` works cleanly.
- Endpoints include: /auth/register, /auth/login; /clients CRUD; /invoices CRUD; /reminders CRUD; /a
- Pagination via limit/offset; consistent response envelope {"data": ..., "meta": ...}.
- Provide a README with setup steps.
```

2) Database Schema & Migrations

```
SYSTEM:
You are a database architect. Optimize for correctness first, then performance.

USER:
Design SQLAlchemy models and Alembic migrations for:
- users(id, email unique, password_hash, role, created_at)
- clients(id, user_id FK, name, email, timezone, notes, payment_behavior_score, created_at)
- invoices(id, user_id FK, client_id FK, amount_cents, currency, due_date, status(enum: draft,pendin
- reminders(id, invoice_id FK, template_id FK nullable, send_at, channel(enum: email,sms,whatsapp), 
- templates(id, user_id FK, name, tone(enum:friendly,neutral,firm), subject, body_markdown)
- events(id, user_id FK, entity_type, entity_id, event_type, payload JSONB, created_at)

Add indexes for common queries: by user_id, by status, by due_date, by send_at.
Include server defaults, updated_at triggers, and DB check constraints (amount > 0, due_date in futu
Provide Alembic scripts and example seed data SQL.
```

3) Stripe Integration (Invoice import + Pay Now link)

```
SYSTEM:
You are an integrations engineer. Follow Stripe best practices.

USER:
Implement Stripe integration module:
- OAuth connect flow for Standard accounts.
- Webhooks: invoice.created, invoice.finalized, invoice.payment_succeeded, invoice.payment_failed.
- Import invoices into our DB; map fields; idempotency via external_id.
- Generate hosted payment link for manual invoices using Stripe Payment Links and attach to our invo
- Secure webhook endpoint with signature verification.
- Retry logic + dead letter table for failures.
- Provide Postman collection for testing and a README with ngrok instructions.
ACCEPTANCE:
```

```
- Running with STRIPE_TEST keys, I can connect an account, see imported invoices, and open Pay Now li
```

4) Email Delivery & Templates

```
SYSTEM:
You are an email platform engineer. Use best practices to ensure deliverability.

USER:
Add email sending via Postmark (preferred) with a provider interface to allow swapping SES later.
- Markdown templates rendered to HTML (Jinja2), with variables: {client_name, invoice_number, amount,
- Pre-send preview endpoint /reminders/preview.
- DKIM/SPF instructions in README.
- Link tracking and message_id stored in reminders.meta.
- Expose /reminders/send-now to trigger immediate send for testing.
- Unit tests mocking Postmark API.
Include 3 tone presets: friendly, neutral, firm.
```

5) Reminder Scheduler (Adaptive)

```
SYSTEM:
You are a backend engineer focusing on scheduled jobs.

USER:
Implement a background scheduler with APScheduler or Celery+Redis (choose one, justify).
- Jobs: enqueue reminder sends at send_at; compute adaptive schedules nightly.
- Adaptive logic: for each client, compute avg_lateness = avg(paid_at - due_date). Choose send windo
- Respect client timezone from clients.timezone.
- Ensure exactly-once execution using advisory locks or Redis setnx.
- Admin endpoint to requeue failed jobs; metrics via Prometheus counters.

Tests must cover idempotency and timezone correctness.
```

6) Analytics API

```
SYSTEM:
You are building analytics endpoints with SQL-first mindset.

USER:
Create /analytics endpoints:
- /analytics/summary: totals by status, expected_payments_next_30d, avg_days_to_pay, top_late_client
- /analytics/timeseries?metric=payments&interval=week
- Use SQL GROUP BYs; avoid N+1s.
- Return {data, meta} with consistent ISO dates.
- Add unit tests with fixtures covering edge cases (no invoices, all paid, etc.).
```

7) Mobile-First Web App (React + Vite)

```
SYSTEM:
You are a front-end lead. Write clean, accessible React with TypeScript and TailwindCSS.

USER:
Build a mobile-first SPA "sic_app":
- Views: Auth, Dashboard (KPIs + lists), Invoice Detail (timeline + reminders), Client Detail, Setti
- Components: KPI cards, Status badges, EmailPreview modal, ReminderSchedule timeline.
- State management with React Query; API client with axios interceptors (JWT).
- Form validation with Zod + react-hook-form.
- Dark mode + accessible (ARIA labels, keyboard nav).
- Include unit tests (Vitest + Testing Library) and basic e2e (Playwright).
- Provide `npm run dev`, `build`, `test`, `e2e` scripts and a README.

ACCEPTANCE:
- On mobile viewport, critical flows are one-hand usable.
```

8) Onboarding Flow

```
SYSTEM:
You design frictionless onboarding.

USER:
Implement a 3-step onboarding wizard:
1) Create account + email verification.
2) Connect Stripe (or choose Manual Invoices).
3) Import first invoices and preview first reminder.
Include progress save, helpful tips, and a sample dataset if user skips integration.
Track onboarding completion events in Events table.
```

9) AI: Tone Presets & Schedule Heuristics

```
SYSTEM:
You are an ML engineer prototyping pragmatic heuristics (no overkill).

USER:
Implement first-pass ML-lite features:
- For each client, compute distribution of payment weekdays/hours; schedule reminders to precede thes
- Tone selector: default=friendly; escalate to neutral after 3 days overdue; firm after 10 days.
- Export functions `choose_tone(client_id, overdue_days)` and `next_send_times(invoice_id)` with docs
- Keep model code modular to swap in a learned model later.
```

10) Subscription Billing (Stripe)

```
SYSTEM:
You implement SaaS billing with Stripe.

USER:
Add subscription tiers: Free (2 reminders/mo), Basic ($9), Pro ($19), Agency ($29).
- Use Stripe Checkout + Billing Portal.
- Enforce plan limits server-side (rate-limit reminder creation/sends).
- Webhooks to sync subscription status; lock premium features when past_due.
- Add coupon support and 'pause plan' flag.
- Tests for limit enforcement and webhook flows.
```

11) Referral Program (MVP)

```
SYSTEM:
You ship a simple, trackable referral.

USER:
Add referrals:
- Each user has a referral_code.
- When a new user signs up with a code, store referral and grant 1 month credit to referrer (post-pa
- Admin endpoint to adjust credits.
- UI: share link, referral count, credits balance.
- Tests for abuse prevention (self-referrals blocked).
```

12) CI/CD, Linting, Security

```
SYSTEM:
You configure CI like a pro.

USER:
Create GitHub Actions workflows:
```

```
- backend: run black, isort, flake8, mypy, pytest with coverage >= 85%.
- frontend: eslint, type-check, vitest, build.
- docker build and push to GHCR on tags.
- Trivy container scan; Bandit for Python security.
Add pre-commit hooks and a CONTRIBUTING.md.
```

13) Deployment (Docker + Fly.io/Vercel)

```
SYSTEM:
You deploy reliably with minimal ops.

USER:
Provide deployment manifests:
- Backend Dockerfile (multi-stage), Fly.io app config with Postgres addon.
- Frontend deploy on Vercel with env var wiring.
- Secrets management documentation.
- Runbooks for rotating keys and restoring DB from snapshot.
```

14) End-to-End Happy Path Script

```
SYSTEM:
You are QA. Define an executable test plan.

USER:
Write a Playwright e2e spec that:
- Registers a user, verifies email (mock link), logs in.
- Connects Stripe in sandbox (mock OAuth), imports sample invoices (fixture).
- Creates a reminder, previews email, schedules send.
- Advances clock, asserts email webhook received, marks invoice paid.
- Checks analytics summary updates.
Include commands to run in CI.
```

15) Docs (Docusaurus)

```
SYSTEM:
You write clear docs.

USER:
Spin up Docusaurus docs with pages for: Overview, Quickstart, API Reference (autogenerated from Open
Add code samples (curl + JS + Python). Publish via GitHub Pages on push to main.
```