Perfect — let's turn this into a **build-ready PRD pack** you can actually use in VS Code this week.

I'm giving you the 4 things you asked for:

1. **Database schema SQL (Supabase + RLS)**
2. **API routes list (Next.js App Router)**
3. **UI wireframe layout (page-by-page)**
4. **Copilot prompt pack (copy/paste ready)**

I'm also aligning this with the official docs stack we picked:

- **Supabase Auth + RLS**
- **Next.js Route Handlers**
- **OpenAI server-side API usage**
- **Stripe Payment Links + webhooks**
- **Vercel preview deployments**
- **Playwright + GitHub Actions**
- **Sentry for Next.js** (Supabase)

---

0) Build assumptions

Stack assumptions (implementation-ready)

- **Next.js App Router** with `app/` directory and `route.ts` handlers for backend endpoints (supports `GET/POST/PATCH/DELETE`, etc.). (Next.js)
- **Supabase Auth** for login (email/password or magic link) and **RLS** for user-scoped access. Supabase docs explicitly recommend RLS on exposed tables. (Supabase)
- **OpenAI API** called from server-side code (Next.js route handlers), using the official SDK shown in the OpenAI quickstart. (OpenAI Developers)
- **Stripe Payment Links** for fast monetization, with a webhook endpoint to sync plan status (`checkout.session.completed`, etc.). Stripe docs describe Payment Links as a no-code entry point and webhooks for async events. (Stripe Docs)
- **Vercel** for deploys + preview URLs per commit/PR. (Vercel)
- **Playwright** for E2E and **GitHub Actions** CI (YAML workflow in `.github/workflows`). (Playwright)
- **Sentry** for Next.js error monitoring and traces. (Sentry Documentation)

---

1) Database schema SQL (Supabase)

Paste this as your first migration in Supabase SQL editor (or a migration file).

```
-- ==========================================
-- Student OS - Initial Schema (MVP)
-- Supabase / Postgres
-- ==========================================

-- Optional but usually available in Supabase
create extension if not exists pgcrypto;

-- ----------------------------------------
-- Utility: updated_at trigger
-- ----------------------------------------
create or replace function public.set_updated_at()
returns trigger
language plpgsql
as $$
begin
  new.updated_at = now();
  return new;
end;
$$;

-- ----------------------------------------
-- Core tables
-- ----------------------------------------

-- Deadlines
create table if not exists public.deadlines (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  title text not null check (char_length(title) between 1 and 200),
  due_date date not null,
  category text not null check (category in ('university', 'rent', 'insurance', 'residence_permit',
'work', 'personal', 'other')),
  status text not null default 'pending' check (status in ('pending', 'done')),
  notes text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
```

```
create index if not exists idx_deadlines_user_id on public.deadlines(user_id);
create index if not exists idx_deadlines_due_date on public.deadlines(due_date);
create index if not exists idx_deadlines_user_due on public.deadlines(user_id, due_date);

create trigger trg_deadlines_updated_at
before update on public.deadlines
for each row execute function public.set_updated_at();

-- Checklist lists (containers/templates per user)
create table if not exists public.checklist_lists (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  title text not null check (char_length(title) between 1 and 120),
  category text not null check (category in ('university', 'housing', 'insurance', 'residence_permit',
'job', 'other')),
  is_template boolean not null default false,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create index if not exists idx_checklist_lists_user_id on public.checklist_lists(user_id);

create trigger trg_checklist_lists_updated_at
before update on public.checklist_lists
for each row execute function public.set_updated_at();

-- Checklist items
create table if not exists public.checklist_items (
  id uuid primary key default gen_random_uuid(),
  list_id uuid not null references public.checklist_lists(id) on delete cascade,
  user_id uuid not null references auth.users(id) on delete cascade,
  label text not null check (char_length(label) between 1 and 200),
  status text not null default 'pending' check (status in ('pending', 'submitted', 'done')),
  notes text,
  sort_order integer not null default 0,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create index if not exists idx_checklist_items_user_id on public.checklist_items(user_id);
create index if not exists idx_checklist_items_list_id on public.checklist_items(list_id);
create index if not exists idx_checklist_items_list_sort on public.checklist_items(list_id,
sort_order);
```

```sql
create trigger trg_checklist_items_updated_at
before update on public.checklist_items
for each row execute function public.set_updated_at();

-- AI email drafts history
create table if not exists public.email_drafts (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  recipient_type text not null check (recipient_type in ('landlord', 'university', 'insurance',
'employer', 'authority', 'other')),
  language text not null check (language in ('en', 'de')),
  tone text not null check (tone in ('formal', 'polite', 'concise')),
  context_input text not null check (char_length(context_input) between 5 and 4000),
  subject text not null,
  body text not null,
  was_copied boolean not null default false,
  created_at timestamptz not null default now()
);

create index if not exists idx_email_drafts_user_id on public.email_drafts(user_id);
create index if not exists idx_email_drafts_created_at on public.email_drafts(created_at);

-- Payment / proof log
create table if not exists public.payment_logs (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  type text not null check (type in ('rent', 'insurance', 'semester_fee', 'deposit', 'transport', 'misc')),
  amount numeric(10,2) not null check (amount >= 0),
  currency text not null default 'EUR' check (char_length(currency) = 3),
  paid_on date not null,
  recipient text not null check (char_length(recipient) between 1 and 120),
  reference_note text,
  proof_url text,
  status text not null default 'paid' check (status in ('paid', 'pending')),
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create index if not exists idx_payment_logs_user_id on public.payment_logs(user_id);
create index if not exists idx_payment_logs_paid_on on public.payment_logs(paid_on);

create trigger trg_payment_logs_updated_at
before update on public.payment_logs
for each row execute function public.set_updated_at();
```

```
-- Subscription / billing state (server-managed)
create table if not exists public.subscriptions (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade unique,
  plan text not null default 'free' check (plan in ('free', 'pro')),
  status text not null default 'inactive' check (status in ('inactive', 'active', 'past_due', 'canceled')),
  stripe_customer_id text,
  stripe_subscription_id text,
  last_checkout_session_id text,
  current_period_end timestamptz,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create index if not exists idx_subscriptions_plan on public.subscriptions(plan);
create index if not exists idx_subscriptions_stripe_customer on
public.subscriptions(stripe_customer_id);
create index if not exists idx_subscriptions_stripe_sub on
public.subscriptions(stripe_subscription_id);

create trigger trg_subscriptions_updated_at
before update on public.subscriptions
for each row execute function public.set_updated_at();

-- AI usage counters (monthly quota tracking)
create table if not exists public.ai_usage_counters (
  id uuid primary key default gen_random_uuid(),
  user_id uuid not null references auth.users(id) on delete cascade,
  usage_month date not null, -- use first day of month e.g. 2026-02-01
  draft_generations integer not null default 0 check (draft_generations >= 0),
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now(),
  unique (user_id, usage_month)
);

create index if not exists idx_ai_usage_user_month on public.ai_usage_counters(user_id,
usage_month);

create trigger trg_ai_usage_updated_at
before update on public.ai_usage_counters
for each row execute function public.set_updated_at();

-- ----------------------------------------
-- Row Level Security (RLS)
-- Supabase best practice for exposed tables
-- ----------------------------------------

alter table public.deadlines enable row level security;
alter table public.checklist_lists enable row level security;
alter table public.checklist_items enable row level security;
alter table public.email_drafts enable row level security;
alter table public.payment_logs enable row level security;
alter table public.subscriptions enable row level security;
alter table public.ai_usage_counters enable row level security;

-- Helper policy pattern: user owns row via user_id
create policy "deadlines_select_own"
on public.deadlines for select
using (auth.uid() = user_id);

create policy "deadlines_insert_own"
on public.deadlines for insert
with check (auth.uid() = user_id);

create policy "deadlines_update_own"
on public.deadlines for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create policy "deadlines_delete_own"
on public.deadlines for delete
using (auth.uid() = user_id);

create policy "checklist_lists_select_own"
on public.checklist_lists for select
using (auth.uid() = user_id);

create policy "checklist_lists_insert_own"
on public.checklist_lists for insert
with check (auth.uid() = user_id);

create policy "checklist_lists_update_own"
on public.checklist_lists for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create policy "checklist_lists_delete_own"
```

```
on public.checklist_lists for delete
using (auth.uid() = user_id);

create policy "checklist_items_select_own"
on public.checklist_items for select
using (auth.uid() = user_id);

create policy "checklist_items_insert_own"
on public.checklist_items for insert
with check (
  auth.uid() = user_id
  and exists (
    select 1
    from public.checklist_lists cl
    where cl.id = checklist_items.list_id
      and cl.user_id = auth.uid()
  )
);

create policy "checklist_items_update_own"
on public.checklist_items for update
using (auth.uid() = user_id)
with check (
  auth.uid() = user_id
  and exists (
    select 1
    from public.checklist_lists cl
    where cl.id = checklist_items.list_id
      and cl.user_id = auth.uid()
  )
);

create policy "checklist_items_delete_own"
on public.checklist_items for delete
using (auth.uid() = user_id);

create policy "email_drafts_select_own"
on public.email_drafts for select
using (auth.uid() = user_id);

create policy "email_drafts_insert_own"
on public.email_drafts for insert
with check (auth.uid() = user_id);
```

```
create policy "email_drafts_update_own"
on public.email_drafts for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create policy "email_drafts_delete_own"
on public.email_drafts for delete
using (auth.uid() = user_id);

create policy "payment_logs_select_own"
on public.payment_logs for select
using (auth.uid() = user_id);

create policy "payment_logs_insert_own"
on public.payment_logs for insert
with check (auth.uid() = user_id);

create policy "payment_logs_update_own"
on public.payment_logs for update
using (auth.uid() = user_id)
with check (auth.uid() = user_id);

create policy "payment_logs_delete_own"
on public.payment_logs for delete
using (auth.uid() = user_id);

-- Subscriptions: users can read only their subscription row.
-- Writes should be server-side (service role / webhook), not client-side.
create policy "subscriptions_select_own"
on public.subscriptions for select
using (auth.uid() = user_id);

create policy "ai_usage_select_own"
on public.ai_usage_counters for select
using (auth.uid() = user_id);

-- Optional: allow client read only; writes from server route
-- If you want client-side incrementing later, add insert/update policies carefully.

-- -----------------------------------------
-- Helpful views (optional, read-only helpers)
-- -----------------------------------------

create or replace view public.deadlines_with_flags as
```

```
select
  d.*,
  (d.status = 'pending' and d.due_date < current_date) as is_overdue,
  (d.status = 'pending' and d.due_date between current_date and current_date + interval '7
days') as is_due_soon
from public.deadlines d;
```

Why this schema is solid

- Clean user ownership model (`user_id` on every row)
- RLS everywhere (important in Supabase for exposed tables) (Supabase)
- Billing is separated and mostly server-managed (safer)
- AI quota tracking is built-in from day 1

---

2) API routes list (Next.js App Router)

Next.js Route Handlers are a great fit here because they live inside `app/` and support standard HTTP methods in `route.ts`files. (Next.js)

Folder structure (API only)

```
app/
  api/
    health/route.ts
    me/route.ts

    deadlines/route.ts
    deadlines/[id]/route.ts

    checklists/route.ts
    checklists/[id]/route.ts
    checklists/[id]/items/route.ts
    checklist-items/[id]/route.ts

    payment-logs/route.ts
    payment-logs/[id]/route.ts

    email-drafts/route.ts
    email-drafts/[id]/route.ts
    ai/
      email-draft/route.ts
      rewrite-draft/route.ts   (v1 optional)

    billing/
      status/route.ts
      payment-link/route.ts    (optional helper endpoint)
      webhook/route.ts         (Stripe webhook)

    telemetry/
      feedback/route.ts        (thumbs up/down on AI drafts)
```

Route specs (MVP-ready)

`GET /api/health`

Purpose: quick health check for debugging/deploy verification

Response

```
{ "ok": true, "service": "student-os-api" }
```

`GET /api/me`

Purpose: return current user + plan status

Response

```
{
  "user": {
    "id": "uuid",
    "email": "user@example.com"
  },
  "subscription": {
    "plan": "free",
    "status": "inactive"
  }
}
```

`GET /api/deadlines`

Purpose: list user deadlines (filter/sort)

Query params (optional)

- `status=pending|done`
- `category=rent|insurance|...`
- `sort=due_date_asc|due_date_desc`

Response

```
{
  "items": [
    {
      "id": "uuid",
      "title": "Rent payment",
      "due_date": "2026-03-01",
      "category": "rent",
      "status": "pending",
      "notes": "Transfer by bank",
      "is_overdue": false,
      "is_due_soon": true
    }
  ]
}
```

`POST /api/deadlines`

Purpose: create deadline

Body

```
{
  "title": "TK insurance payment",
  "due_date": "2026-03-05",
  "category": "insurance",
  "status": "pending",
  "notes": "Monthly payment"
}
```

Response

```
{ "item": { "...": "created deadline row" } }
```

`PATCH /api/deadlines/:id`

Purpose: update deadline

Body (partial)

```
{
  "status": "done"
}
```

`DELETE /api/deadlines/:id`

Purpose: delete deadline

Response

```
{ "ok": true }
```

`GET /api/checklists`

Purpose: list checklist lists with completion summary

Response

```
{
  "items": [
    {
      "id": "uuid",
      "title": "Residence Permit",
      "category": "residence_permit",
      "is_template": false,
      "total_items": 8,
      "done_items": 3,
      "submitted_items": 2,
      "progress_percent": 62
    }
  ]
}
```

POST /api/checklists

Purpose: create checklist list

Body

```
{
  "title": "University Admission Admin",
  "category": "university",
  "is_template": false
}
```

GET /api/checklists/:id

Purpose: get list + items

Response

```
{
  "list": {
    "id": "uuid",
    "title": "Residence Permit",
    "category": "residence_permit"
  },
  "items": [
    {
      "id": "uuid",
      "label": "Passport copy",
      "status": "done",
    "sort_order": 1
  }
 ]
}
```

PATCH /api/checklists/:id

Purpose: rename / change category / toggle template

DELETE /api/checklists/:id

Purpose: delete list (cascades items)

POST /api/checklists/:id/items

Purpose: add checklist item

Body

```
{
 "label": "Meldebescheinigung",
 "status": "pending",
 "notes": "",
 "sort_order": 3
}
```

PATCH /api/checklist-items/:id

Purpose: update item status/text/order

DELETE /api/checklist-items/:id

Purpose: delete item

`GET /api/payment-logs`

Purpose: list payment/proof records

Query params (optional)

- `type=rent`
- `status=paid`
- `month=2026-02`

`POST /api/payment-logs`

Purpose: create payment log record

Body

```
{
  "type": "rent",
  "amount": 450.00,
  "currency": "EUR",
  "paid_on": "2026-02-28",
  "recipient": "Mieterio",
  "reference_note": "Rent Feb 2026",
  "proof_url": "",
  "status": "paid"
}
```

`PATCH /api/payment-logs/:id`

Purpose: edit payment record

`DELETE /api/payment-logs/:id`

Purpose: delete payment record

`GET /api/email-drafts`

Purpose: list generated drafts history (latest first)

Query params

- `limit=20`
- `language=de`
- `recipient_type=landlord`

`PATCH /api/email-drafts/:id`

Purpose: mark metadata (e.g., `was_copied=true`)

Body

{ "was_copied": true }

`POST /api/ai/email-draft`

Purpose: generate AI draft (server-side OpenAI call)

OpenAI's quickstart shows the official JS SDK for server-side environments and a simple request flow, which is exactly the pattern you'll use here. (OpenAI Developers)

Body

```
{
  "recipient_type": "landlord",
  "language": "de",
  "tone": "formal",
  "context": "I paid the rent but forgot to include the reference. I want to confirm the payment and
apologize.",
  "include_deadline": false
}
```

Response

```
{
  "draft": {
    "subject": "Bestätigung der Mietzahlung",
    "body": "Sehr geehrte Damen und Herren, ...",
    "language": "de",
    "tone": "formal"
  },
  "usage": {
    "remaining_free_drafts": 7,
    "plan": "free"
  }
}
```

Validation rules for this endpoint

- Require auth
- Limit `context` length (e.g. 20–2000 chars)
- Enforce monthly quota if `plan = free`
- Save generated output to `email_drafts`
- Return safe error if AI provider fails

`GET /api/billing/status`

Purpose: return user plan + quota status

Response

```
{
  "plan": "free",
  "subscription_status": "inactive",
  "usage": {
    "month": "2026-02-01",
    "draft_generations": 3,
    "free_limit": 10
  }
}
```

`GET /api/billing/payment-link` (optional helper)

Purpose: return the Stripe Payment Link URL from env/config

You can also skip this route and hardcode the link in your frontend config for MVP.

`POST /api/billing/webhook`

Purpose: handle Stripe webhook events

Stripe docs describe webhook endpoints as HTTPS endpoints that receive JSON `Event` payloads for async payment events. (Stripe Docs)

Events to handle first

- `checkout.session.completed` → mark user `pro` (or create subscription row)
- `customer.subscription.updated` → sync status
- `customer.subscription.deleted` → downgrade/cancel
- `invoice.payment_failed` → set `past_due` if using subscriptions
- `invoice.paid` → keep active if needed

`checkout.session.completed` is a listed Stripe event type and is the key one for Payment Link/Checkout completion flows. (Stripe Docs)

Important implementation note

Webhook route should:

- read raw body
- verify Stripe signature
- be idempotent (don't double-process)
- log failures safely

`POST /api/telemetry/feedback` (optional but useful)

Purpose: thumbs up/down for draft quality

Body

```
{
  "draft_id": "uuid",
  "rating": "up",
  "reason": "too formal"
}
```

This is super useful for improving prompts later.

---

3) UI wireframe layout (simple + clean)

These are intentionally minimal so you can ship fast.

A) Login page (`/login`)

```
+----------------------------------------------------+
| Student OS                                          |
| "Stay on top of deadlines, docs, and emails."       |
|                                                    |
|  [ Email address input ___________________________ ]  |
|  [ Continue with Magic Link ]                       |
|                                                    |
|  or                                                |
|                                                    |
|  [ Email ] [ Password ]                             |
|  [ Sign In ]                                        |
|                                                    |
|  Small note: "Your data stays private to your account"|
+----------------------------------------------------+
```

UX notes

- Keep this page very clean
- One clear CTA (Magic Link recommended)
- Add link: “Why Student OS?”

Supabase Auth supports email/password and magic links, so you can start with either. (Supabase)

B) Dashboard (`/dashboard`) — the “home base”

```
+------------------------------------------------------------------------------+
| Top Nav: Student OS | Dashboard | Deadlines | Checklists | Payments | Draft Email|
|                              [Plan: Free] [Upgrade] [User Menu]          |
+------------------------------------------------------------------------------+

+-----------------------------+  +---------------------------------------------+
| Quick Actions               |  | Upcoming Deadlines                          |
| [ + New Deadline ]          |  | 1. Rent payment     | 2026-03-01 | Due soon  |
| [ + Draft Email ]           |  | 2. Insurance doc    | 2026-03-04 | Pending   |
| [ + Payment Log ]           |  | 3. Uni form         | 2026-03-10 | Pending   |
| [ + Checklist ]             |  | [View all]                                  |
+-----------------------------+  +---------------------------------------------+

+-----------------------------+  +---------------------------------------------+
| Overdue (if any)            |  | AI Draft Usage (free tier)                  |
| 0 overdue ✅                |  | 3 / 10 used this month                      |
|                             |  | [Upgrade to Pro]                            |
+-----------------------------+  +---------------------------------------------+

+--------------------------------------------------------------------------+
| Draft Email Quick Panel                                                  |
| Recipient: [Landlord v] Language: [DE v] Tone: [Formal v]                |
| Context: [_____________________________________________]                 |
| [ Generate Draft ]                                                       |
+--------------------------------------------------------------------------+
```

Why this layout works

- One-page overview
- User sees urgency + actions immediately
- “Draft Email” is visible (your killer feature)

C) Deadlines page (`/deadlines`)

```
+------------------------------------------------------------------------------+
| Deadlines                                                  [ + Add Deadline ]  |
+------------------------------------------------------------------------------+
| Filters: [Status v] [Category v] [Sort by due date v] [Search ____ ]            |
+------------------------------------------------------------------------------+

| Title                    | Category          | Due Date    | Status   | Actions |
| Rent payment             | Rent              | 2026-03-01  | Pending  | Edit Del|
| TK document upload        | Insurance         | 2026-03-05  | Done      | Edit Del|
| Residence permit appt     | Residence Permit  | 2026-03-20  | Pending  | Edit Del|
```

Add/Edit modal

Fields:

- Title
- Due date
- Category
- Status
- Notes

D) Draft Email page (`/draft-email`) — your star feature

```
+------------------------------------------------------------------------------+
| Draft Email                                                        |
+------------------------------------------------------------------------------+
| Left Panel (Inputs)                | Right Panel (Generated Draft)            |
|-------------------------------------|------------------------------------------|
| Recipient Type [ landlord v ]        | Subject                                |
| Language      [ de v ]              | [ Bestätigung der Mietzahlung ]          |
| Tone          [ formal v ]          |                                  |
|                                     | Body                             |
| Context                            | [ Sehr geehrte Damen und Herren, ... ]   |
| [____________________________________] |                                    |
| [____________________________________] |                                    |
| [____________________________________] |                                    |
|                                     | [ Copy ] [ Regenerate ] [ Shorter ]       |
| [ Generate Draft ]                  | [ More Formal ] [ Translate EN/DE ]       |
+------------------------------------------------------------------------------+

+------------------------------------------------------------------------------+
| Recent Drafts (history)                                              |
| - Landlord (DE) | 2 min ago | copied ✅                               |
| - University (EN) | yesterday                                        |
+--------------------------------------------------------------------------------+
```

UX notes

- Split layout = feels premium
- Add “Review before sending” note under output
- Track “copied” clicks (good product metric)

E) Checklists page (`/checklists`)

```
+--------------------------------------------------------------------------------+
| Checklists                                          [ + New Checklist ] |
+--------------------------------------------------------------------------------+

+------------------------------+   +--------------------------------------------+
| Checklist Lists              |   | Selected Checklist: Residence Permit        |
|------------------------------|   | Progress: 5 / 8 (62%)                      |
| - Residence Permit (62%)     |   |--------------------------------------------|
| - University (40%)           |   | [ ] Passport copy                          |
| - Insurance (80%)            |   | [✓] Health insurance proof                 |
| - Housing (25%)              |   | [Submitted] Anmeldung certificate          |
|                              |   | [ ] Biometric photo                        |
| [Create from template]       |   |                                            |
+------------------------------+   | [ + Add Item ]                             |
                               +--------------------------------------------+
```

MVP scope tip

Keep checklist items simple:

- checkbox-like interactions
- one-click status change

F) Payments / Proof page (`/payments`)

```
+--------------------------------------------------------------------------------+
| Payments & Proofs                                   [ + Add Payment ]  |
+--------------------------------------------------------------------------------+
| Filters: [Type v] [Status v] [Month v]                                  |
+--------------------------------------------------------------------------------+

| Date       | Type      | Recipient | Amount | Status | Ref Note         | Actions |
| 2026-02-28 | Rent      | Mieterio  | €450   | Paid   | Rent Feb 2026    | Edit Del|
| 2026-02-12 | Insurance | TK        | €125   | Paid   | Monthly premium  | Edit Del|
```

UX notes

- Super simple list
- Later you can add export CSV/PDF (nice upgrade feature)

G) Settings / Billing page (`/settings`)

```
+----------------------------------------------------------+
| Settings                                          |
+----------------------------------------------------------+
| Profile                                         |
| - Email: ayush@example.com                              |
| - Language preference: English / German                 |
|                                                |
| Billing                                         |
| - Plan: Free                                        |
| - AI drafts this month: 3 / 10                          |
| [ Upgrade to Pro ]                                  |
|                                                |
| Data                                            |
| [ Export my data ]   [ Delete account ] (later)          |
+----------------------------------------------------------+
```

---

4) Copilot prompt pack (copy/paste ready)

These are designed for **VS Code Copilot Chat** so you can move fast.

Prompt 0 — Project operating rules (use this first)

Paste this at the start of your project chat so Copilot follows your standards.

You are helping me build a production-quality MVP in Next.js (App Router) + TypeScript + Tailwind + Supabase.

Project: Student OS (for international students)
Features: Auth, deadlines, checklists, AI email drafts, payment log, basic billing status

Coding rules:
- Use clean modular architecture
- Strong TypeScript types for request/response and DB rows
- Server-side API routes in app/api/**/route.ts
- Never expose secrets on the client
- Validate all API inputs (use zod)
- Return consistent JSON responses with { ok, data?, error? }
- Add loading, empty, and error states in UI
- Keep components small and reusable
- Write comments only where needed
- Prefer readable code over clever code
- Prepare code for Supabase RLS user-scoped access

Prompt 1 — Generate the project structure

Create a clean folder structure for a Next.js App Router project called Student OS with:
- app routes for dashboard, deadlines, checklists, payments, draft-email, settings, login
- app/api route handlers for deadlines, checklists, checklist-items, payment-logs, AI email draft, billing status, webhook
- lib folder for supabase clients, auth helpers, validation schemas, AI client, billing utils
- components folder grouped by feature
- types folder for shared interfaces

Return the folder tree and create starter files.

Prompt 2 — Supabase integration + auth guard

Implement Supabase auth in a Next.js App Router app using server and client helpers.
Requirements:
- login page with email magic link
- protected routes for dashboard and all app pages

- middleware or route protection pattern
- helper to get current user on server
- helper to return JSON 401 for API routes if unauthenticated
Use TypeScript and clean abstractions.

Prompt 3 — Deadline feature (full CRUD)

Build the deadlines feature end-to-end.

Backend:
- GET /api/deadlines (filters + sorting)
- POST /api/deadlines
- PATCH /api/deadlines/[id]
- DELETE /api/deadlines/[id]
- zod validation
- user-scoped Supabase queries

Frontend:
- deadlines page with table/list
- add/edit modal
- filters for status/category
- empty and error states
- optimistic updates if practical

Return all code and explain where each file goes.

Prompt 4 — Checklists feature

Build checklist lists + checklist items.

Requirements:
- checklist lists on left side
- selected list and items on right
- create checklist list
- add/edit/delete checklist item
- item status values: pending, submitted, done
- progress percent UI
- API routes:
  - /api/checklists
  - /api/checklists/[id]

  - /api/checklists/[id]/items
  - /api/checklist-items/[id]

Use TypeScript, zod, and reusable components.

Prompt 5 — Payment log feature

Build the payment log module.

Requirements:
- API routes for list/create/update/delete payment logs
- fields: type, amount, currency, paid_on, recipient, reference_note, proof_url, status
- page UI with filter bar and table/list
- add/edit modal form
- basic validation and user-friendly error messages
- all data scoped to current user

Prompt 6 — AI email draft route (OpenAI server-side)

OpenAI quickstart shows the official JS SDK in server-side Node environments — use that pattern in the route handler. (OpenAI Developers)

Create a Next.js API route handler at /api/ai/email-draft that generates an email draft using the OpenAI SDK.

Requirements:
- Read API key from environment variables
- Validate body with zod:
  recipient_type, language (en/de), tone, context
- Build a strong system prompt for formal student/admin emails
- Return structured JSON: { subject, body, language, tone }
- Save the generated draft to Supabase email_drafts table
- Enforce free-tier monthly quota using ai_usage_counters table
- Return clear errors for invalid input, quota exceeded, and provider failure
- Do not expose any secrets in client code

Also create the frontend page for draft generation with:
- input form on left
- generated output on right
- copy/regenerate buttons

- recent drafts list below

Prompt 7 — Billing status + Stripe webhook

Stripe docs say Payment Links are a fast no-code way to start, and webhooks are used for async events. (Stripe Docs)

Implement basic billing for Student OS.

Requirements:
1) GET /api/billing/status
- returns current plan (free/pro), subscription status, and AI usage count

2) POST /api/billing/webhook (Stripe)
- verify Stripe signature from raw request body
- handle at least:
  - checkout.session.completed
  - customer.subscription.updated
  - customer.subscription.deleted
- update subscriptions table accordingly
- make webhook processing idempotent

3) Frontend settings page
- show current plan
- show usage this month
- show Upgrade button linking to Stripe Payment Link (URL from env)

Use TypeScript and clean utility functions.

Prompt 8 — Shared validation and API response utilities

Create reusable utilities for all API routes:
- successResponse(data, status?)
- errorResponse(message, status?, details?)
- requireAuthForRoute()
- parseJsonBody<T>()
- zod error formatter
- consistent typed API response interface

Refactor example routes (deadlines and payment-logs) to use these utilities.

Prompt 9 — Playwright E2E tests

Playwright is built for E2E testing and supports Chromium/WebKit/Firefox locally or in CI. (Playwright)

Set up Playwright E2E tests for this Next.js app.

Create tests for:
1) Login page loads
2) Create a deadline and verify it appears in the list
3) Generate an AI draft (mock the AI API route if needed)
4) Create a payment log entry

Also:
- add test IDs where needed
- use reliable locators
- include one test helper for authenticated state (if practical)

Prompt 10 — GitHub Actions CI workflow

GitHub Actions CI runs builds/tests, and workflow files are YAML in `.github/workflows`. (GitHub Docs)

Create a GitHub Actions CI workflow for this project.

Requirements:
- file path: .github/workflows/ci.yml
- triggers: push and pull_request
- run on ubuntu-latest
- setup Node
- install dependencies
- run lint
- run typecheck
- run tests (unit if present)
- run Playwright tests (or skip with a clear TODO if secrets are missing)
- cache dependencies for speed

Return the YAML and explain any required secrets.

Prompt 11 — Sentry integration (optional but impressive)

Sentry provides a Next.js guide for setup and capturing errors/traces. (Sentry Documentation)

Integrate Sentry into this Next.js project.

Requirements:
- install and configure Sentry for Next.js
- capture API route errors and frontend runtime errors
- add a small helper for logging expected vs unexpected errors
- avoid leaking sensitive request data
- add a test button/page in development to trigger a sample error

Return all changed files and environment variables needed.

Prompt 12 — README + portfolio polish

Write a professional README for Student OS.

Include:
- project summary
- problem statement
- features
- tech stack
- architecture overview
- local setup instructions
- environment variables
- database setup notes
- testing instructions
- deployment steps (Vercel)
- future roadmap
- screenshots placeholders

Tone: clean, recruiter-friendly, not overly long.

5) Bonus — Build order (so you don’t get overwhelmed)

Here’s the best sequence for *you*:

Week 1 (MVP core)

1. **Project scaffold**
2. **Supabase auth**
3. **Deadlines CRUD**
4. **Draft Email page (UI only)**
5. **AI route integration**
6. **Deploy to Vercel**

Vercel gives you unique deployment URLs and preview deployments for branch/PR pushes, which is perfect for quick iteration and sharing demos. (Vercel)

Week 2 (make it look real)

1. Checklists
2. Payment log
3. Billing status page + Upgrade button (Stripe Payment Link)
4. Playwright test (at least 1–2)
5. GitHub Actions CI
6. README + screenshots

Week 3 (portfolio polish)

1. Sentry
2. Better UI polish
3. AI output improvements (shorter/more formal buttons)
4. User feedback loop (thumbs up/down)
5. Soft launch to 5 friends/classmates

---

6) Tiny but important implementation tips

- **Don’t overbuild billing first.** Start with a Stripe Payment Link and a simple webhook sync.
- **Keep AI prompts narrow.** Your app wins because it solves a *specific* email drafting use case.
- **Track usage from day 1.** Even a simple `ai_usage_counters` table makes your app feel “product-minded.”
- **Ship ugly, then polish.** Functionality first. Then make it beautiful.

If you want, I can do the **next step immediately** and generate the actual **starter code files** for:

- `app/api/ai/email-draft/route.ts`
- `lib/validation/emailDraft.ts`
- `app/draft-email/page.tsx`
- `lib/supabase/server.ts`

So you can paste them straight into VS Code and run.