

BriskRecall Quick map — what we'll set up

1. Local machine: Node.js + Git
 2. Supabase project: create project, get Project URL, and anon public key
 3. Database schema in Supabase (SQL provided) — `profiles`, `decks`, `flashcards` + foreign keys + RLS policies
 4. Google Generative AI (Gemini): create API key (Google Cloud), enable Generative AI API
 5. Local repo: install frontend (Vite React) and backend (Express) deps
 6. Create `.env` files (frontend and server) with correct variable names
 7. Run backend `node server/server.js` and frontend `npm run dev` (Vite)
 8. Test flows: signup/signin, upload PDF, verify generated flashcards in Supabase, open DeckView
-

1) Prepare your machine

- Install Node.js (LTS recommended): <https://nodejs.org>

Confirm:

```
node -v  
npm -v
```

- Install Git if needed: <https://git-scm.com>
 - Confirm: `git --version`

2) Clone project / get code

From project root (where your `flashcards-project` folder lives) — if using git:

```
git clone <repo-url>
cd flashcards-project
```

We'll assume repo layout:

```
flashcards-project/
package.json      # frontend (root Vite app)
src/              # frontend source
server/           # backend server
  server.js
  package.json    # backend (if separate)
```

3) Create Supabase project and get API keys

1. Sign up / login: <https://app.supabase.com>
2. Create a new project, choose DB password and region, enable RLS.
3. After project created → go to **Project Settings** → **Data API and API Keys**:
 - Copy `Project URL` (call it `VITE_SUPABASE_URL`)
 - Copy `anon public` key (call it `VITE_SUPABASE_ANON_KEY` for frontend)

4) SQL: create tables + foreign keys + cascade + RLS policies

Open Supabase SQL editor (Project → SQL) and run the SQL below. It creates **profiles**, **decks**, **flashcards**, sets FK cascade, and example RLS policies.

NOTE: table column names need to match your code

```
-- Create profiles (optional: mirror auth user id)
create table if not exists public.profiles (
  id uuid primary key,    -- matches auth.uid()
  username text,
  created_at timestamptz default now()
);

-- Create decks
create table if not exists public.decks (
  id uuid default gen_random_uuid() primary key,
  user_id uuid not null,  -- the auth.user id (foreign to profiles.id / auth.users)
  title text,
  created_at timestamptz default now()
);

-- Create flashcards
create table if not exists public.flashcards (
  id uuid default gen_random_uuid() primary key,
  deck_id uuid not null references public.decks(id) on delete cascade,
  question_text text,
  answer_text text,
  created_at timestamptz default now()
);

-- Add extension for gen_random_uuid if not present (Postgres)
create extension if not exists "pgcrypto";

-- RLS: enable for tables (recommended)
alter table public.profiles enable row level security;
alter table public.decks enable row level security;
alter table public.flashcards enable row level security;

-- Profiles policies
create policy "Profiles: users can insert own profile"
  on public.profiles for insert to anon using (auth.uid() = id) with check (auth.uid() = id);

create policy "Profiles: users can view own profile"
  on public.profiles for select to anon using (auth.uid() = id);
```

```
-- Decks policies
create policy "Decks: users can insert decks for themselves"
  on public.decks for insert to anon with check (auth.uid() = user_id);

create policy "Decks: users can view their decks"
  on public.decks for select to anon using (auth.uid() = user_id);

create policy "Decks: users can update/delete own decks"
  on public.decks for update, delete to anon using (auth.uid() = user_id) with check (auth.uid() =
user_id);

-- Flashcards policies
-- Let authorized users insert flashcards only for decks they own
create policy "Flashcards: insert if deck belongs to user"
  on public.flashcards for insert to anon with check (
    exists (select 1 from public.decks d where d.id = deck_id and d.user_id = auth.uid())
  );

create policy "Flashcards: user can view flashcards for their decks"
  on public.flashcards for select to anon using (
    exists (select 1 from public.decks d where d.id = deck_id and d.user_id = auth.uid())
  );

create policy "Flashcards: update/delete if deck belongs to user"
  on public.flashcards for update, delete to anon using (
    exists (select 1 from public.decks d where d.id = deck_id and d.user_id = auth.uid())
  );
```

- Run the SQL. Deleting a **deck** will cascade-delete its **flashcards** (due to **on delete cascade**).

5) Google Generative AI (Gemini) — create API key

1. Create a Google Cloud account and project: <https://console.cloud.google.com>
2. Enable the **Generative AI API** (or "Generative Language API") in your project.
3. Create an API key or service account and set appropriate access — follow Google docs.

4. Save the key to `VITE_GEMINI_API_KEY` (server environment variable).

Note: Google may require billing and specific project configuration, but their Free Tier Plan should be available. Follow their docs and check the exact model names available by calling `listModels` or checking docs. Current Code uses `gemini-2.5-flash`, if error arises check for modern models available

6) Install dependencies (frontend + server)

From project root:

Frontend (root Vite app)

```
cd flashcards-project
npm install
# start dev server
npm run dev
# the Vite dev URL will appear, e.g. http://localhost:5173
```

If your `package.json` root is set to `vite` scripts, `npm run dev` starts frontend.

List of dependencies listed on `package.json`:

```
"@picocss/pico": "^2.1.1",

"@supabase/supabase-js": "^2.80.0",

"react": "^19.1.1",

"react-dom": "^19.1.1",

"react-router-dom": "^7.9.5"
```

Check for missing libraries and install them with `npm install "Library_name"`

Backend (server folder)

Open another terminal:

```
cd flashcards-project/server
npm install
# (install any missing modules used in server.js)
# Example:
# npm install express multer cors @supabase/supabase-js pdf2json dotenv
# @google/generative-ai
# Start server:
node server.js
# or if you use nodemon:
npx nodemon server.js
```

List of dependencies listed on `package.json`:

```
"@google/generative-ai": "^0.24.1",
"@supabase/supabase-js": "^2.76.1",
"cors": "^2.8.5",
"dotenv": "^17.2.3",
"express": "^5.1.0",
"multer": "^2.0.2",
"node": "^25.2.1",
"node-fetch": "^3.3.2",
"nodemon": "^3.1.11",
"pdf2json": "^4.0.0"
```

Check for missing libraries and install them with `npm install "Library_name"` inside the server folder.

7) Create **.env** files — exact content (two places)

A) Frontend **.env** (root Vite app)

File: `flashcards-project/.env` (Vite must see `VITE_` prefixed variables)

```
VITE_SUPABASE_URL=https://<your-project-ref>.supabase.co  
VITE_SUPABASE_ANON_KEY=<your-anon-public-key>  
VITE_PORT=5000 # backend PORT used by upload form
```

- After changing `.env`, restart Vite (`npm run dev`).

B) Backend **.env** (server folder)

File: `flashcards-project/server/.env`

```
VITE_PORT=5000  
VITE_SUPABASE_URL=https://<your-project-ref>.supabase.co  
VITE_SUPABASE_ANON_KEY=<your-anon-role-key> # server-only secret  
VITE_GEMINI_API_KEY=<your-google-generative-ai-key>
```

Troubleshooting Tip: If backend doesn't work, and error message in terminal doesn't recognize the variables in `.env`, change the names of the variables **JUST** inside your `flashcards-project/server/.env` by removing `VITE_` and adjust the name of the variables in `server/server.js` to match those of the `.env`

Important:

- Frontend variables *must* start with `VITE_` to be accessible in client code (`import.meta.env.VITE_...`). It may not be the exact same case for the Backend.
 - Backend and Frontend variables **must not** be committed to git; include `.env` in `.gitignore`.
-

8) Start backend and frontend

1. Start backend first (so API endpoint exists):

```
cd flashcards-project/server
node server.js
# expected output: "✅ Server running on http://localhost:5000"
```

2. Start frontend (in a separate terminal):

```
cd flashcards-project
npm run dev
# Vite opens at http://localhost:5173 (or similar)
```

3. Open <http://localhost:5173> in browser.
-

9) Test end-to-end flows

A. Create a Supabase auth user (from UI)

- Go to your site <http://localhost:5173> → Signup page → create account.
- Remember the Password.

B. Sign in

- Sign in with the credentials (Email and Password).

C. Upload PDF

- Go to Upload page → choose a PDF → give a deck title → click Upload.
- The frontend will call **POST** <http://localhost:5000/api/upload> (backend).
- Backend will:

- Verify Supabase token (header `Authorization: Bearer <access_token>`)
- Extract text from PDF (pdf2json)
- Call Gemini to generate 15 Q/A (ensure model name valid)
- Return JSON `{ deckTitle, cards: [{question, answer}, ...] }`
- Frontend will:
 - Create `decks` row for the authenticated user
 - Insert `flashcards` rows in `flashcards` table
 - Navigate to Decks / DeckView

D. Verify Supabase tables

- In Supabase Dashboard → Table Editor → `decks` / `flashcards` → see rows.
 - Deleting a deck should remove flashcards (cascade).
-

10) How to get API endpoints / keys in Supabase UI

- Project → Settings → API: `Project URL`, `anon key`
 - Auth → Settings → Email templates: paste HTML templates
 - Table Editor → view table rows, run SQL in SQL editor
-

11) Where to get Google Gemini info

- Google Gemini (Generative AI): Google Cloud Console → Enable API → Create API key or service account → copy key → check model naming (call `listModels` or read docs).

12) How to test endpoints manually (curl)

- Test backend health:

```
curl http://localhost:5000/  
# should return "Backend API is running 
```

- Test upload endpoint (requires a valid supabase token and a pdf file). Example (replace token and file):

```
curl -X POST "http://localhost:5000/api/upload" \  
-H "Authorization: Bearer <ACCESS_TOKEN_FROM_SUPABASE>" \  
-F "title=Test Deck" \  
-F "file=@/path/to/file.pdf"
```

To get a test token, use supabase client in browser console after login:
`supabase.auth.getSession()` or inspect `localStorage`.

13) Common errors & fixes (quick)

- **supabaseKey required / undefined envs**
 - Fix: ensure backend `.env` exists, variables names correct, and dotenv loaded at top of `server.js`. Run server from `server` folder.

Add debug logs at top of `server.js`:

```
console.log('VITE_SUPABASE_URL', process.env.SUPABASE_URL);
```

- - **ERR_CONNECTION_REFUSED** to `/api/upload`

- **PDF parsing errors (URI malformed)**
 - pdf2json can produce strings that fail `decodeURIComponent`. Use `safeDecode` wrapper (try/catch) as we already implemented.
 - **models/... not found from Gemini**
 - Fix: check model name; use `gemini-2.5-flash-latest` or call `genAI.listModels()` to see available models.
 - **No API key found in request** when calling Supabase REST
 - Fix: ensure `supabase` client created with correct key and that frontend calls have `Authorization` header where required.
 - **CORS**
 - Your backend uses `cors()` by default. If you customized, ensure `cors({ origin: 'http://localhost:5173' })` allows your dev host.
 - Run project as admin
-

14) Security reminders

- Never commit `.env` files; add to `.gitignore`.
 - If sharing repo with collaborators, give them instructions to create their own `.env` with keys.
-

15) Minimal checklist someone should follow (condensed)

1. Install Node.js + Git
2. Clone repo
3. Create Supabase project, copy `Project URL`, and `anon public key`

4. Run SQL above in Supabase SQL editor
5. Get Google Gemini API key
6. Create `flashcards-project/.env` (frontend) with `VITE_SUPABASE_URL`, `VITE_SUPABASE_ANON_KEY`, `VITE_PORT`
7. Create `flashcards-project/server/.env` (backend) with `VITE_SUPABASE_URL`, `VITE_GEMINI_API_KEY`, `VITE_SUPABASE_ANON_KEY`, `VITE_PORT`
8. `cd flashcards-project && npm install`
9. `cd server && npm install`
10. Start server: `node server/server.js`
11. Start frontend: `npm run dev` (in project root)
12. Open <http://localhost:5173> → Signup → Upload PDF → check `decks` + `flashcards` in Supabase