

JS Apps with Supabase

Database Apps with Users and Roles,
Supabase Auth, Supabase Storage



Svetlin Nakov, PhD
Co-founder @ SoftUni

Agenda

1. **Database apps with users**: auth, users, register, login, logout
2. **Database migrations**: concepts, SQL migration scripts, Supabase migrations infrastructure
3. **Locations on the Map**: example of JS app with database, auth, users, RLS policies, DB schema migrations
4. **Row-Level Security (RLS)** in Supabase
5. **Users and roles** with Supabase Auth and RLS
6. **Edge functions**: server-side JS logic in the backend
7. **Supabase storage**: file uploads and downloads



Sli.do Code

#Soft-Tech-AI

Join at

slido.com

#Soft-Tech-AI



Breaks

20:00 / 21:00



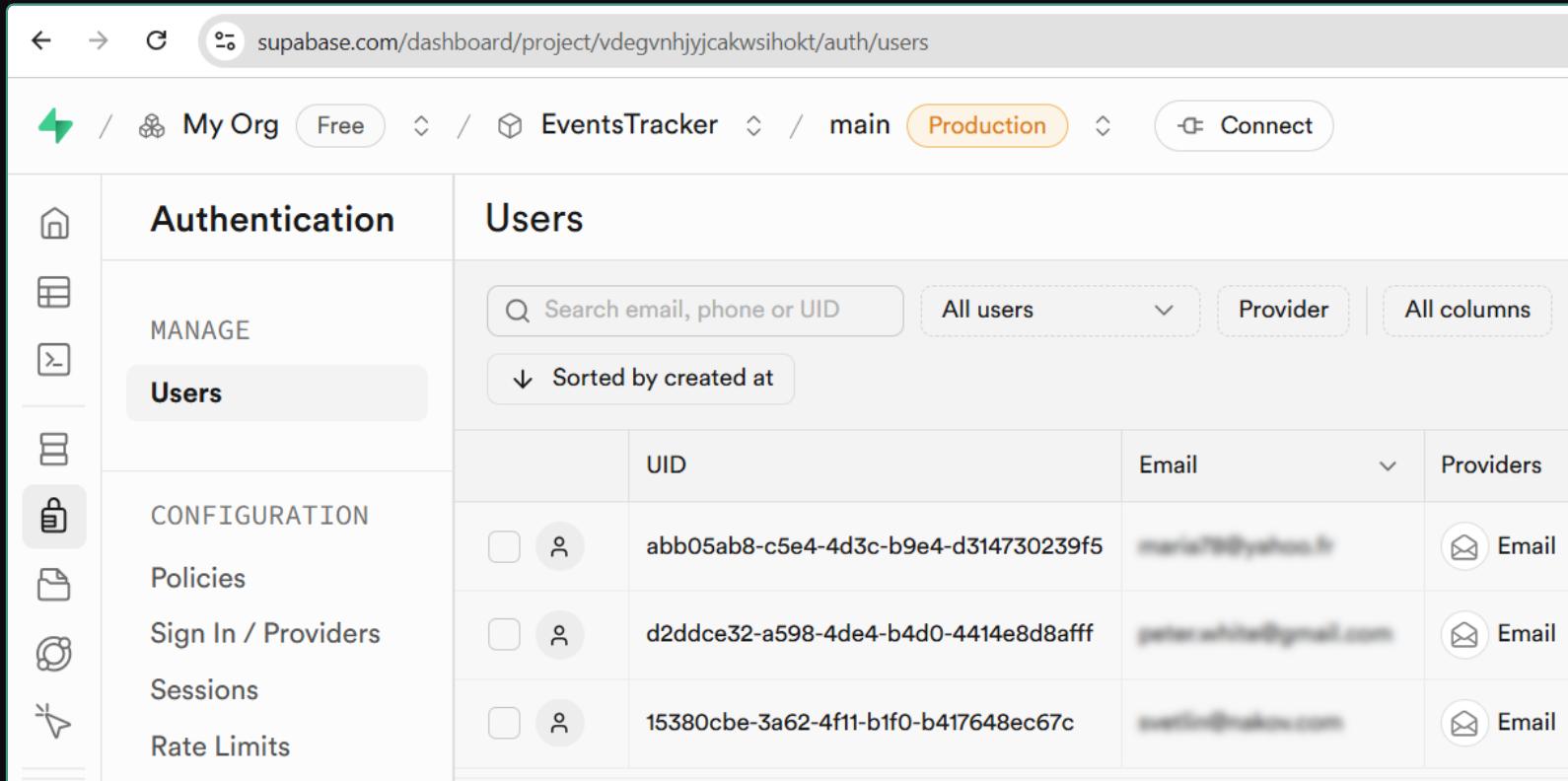
Database Apps with Users

Auth, Users, Register / Login / Logout



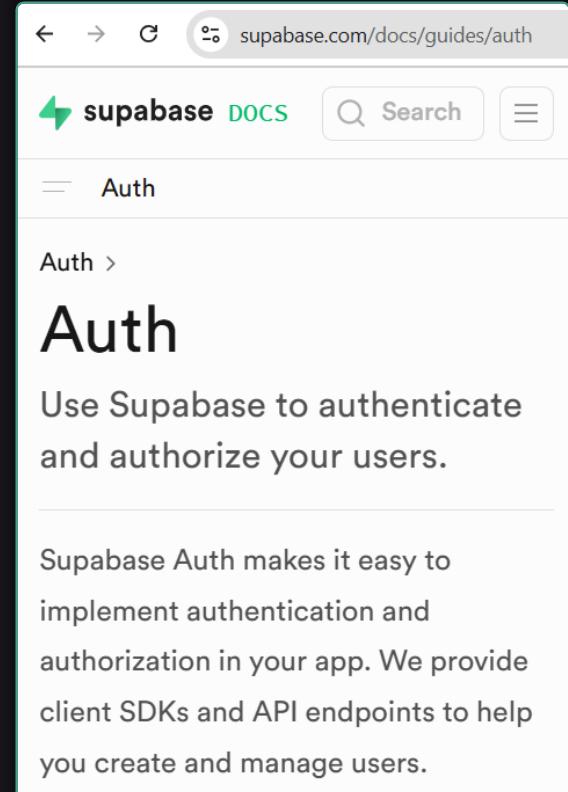
Authentication in Supabase

- Supabase has **built-in authentication** back-end
 - Users, roles, external login providers, multi-factor auth, ...



The screenshot shows the Supabase dashboard for managing users. The URL in the address bar is `supabase.com/dashboard/project/vdegvnjhjycakwsihokt/auth/users`. The navigation bar includes 'My Org' (Free), 'EventsTracker', 'main', 'Production' (selected), and 'Connect'. On the left sidebar, under 'MANAGE', 'Users' is selected. The main table lists three users with columns for UID, Email, and Providers (all listed as Email). The table is sorted by created at.

	UID	Email	Providers
<input type="checkbox"/>	abb05ab8-c5e4-4d3c-b9e4-d314730239f5	maria@yahoo.it	Email
<input type="checkbox"/>	d2ddce32-a598-4de4-b4d0-4414e8d8afff	peter.white@gmail.com	Email
<input type="checkbox"/>	15380cbe-3a62-4f11-b1f0-b417648ec67c	sue@live.com	Email

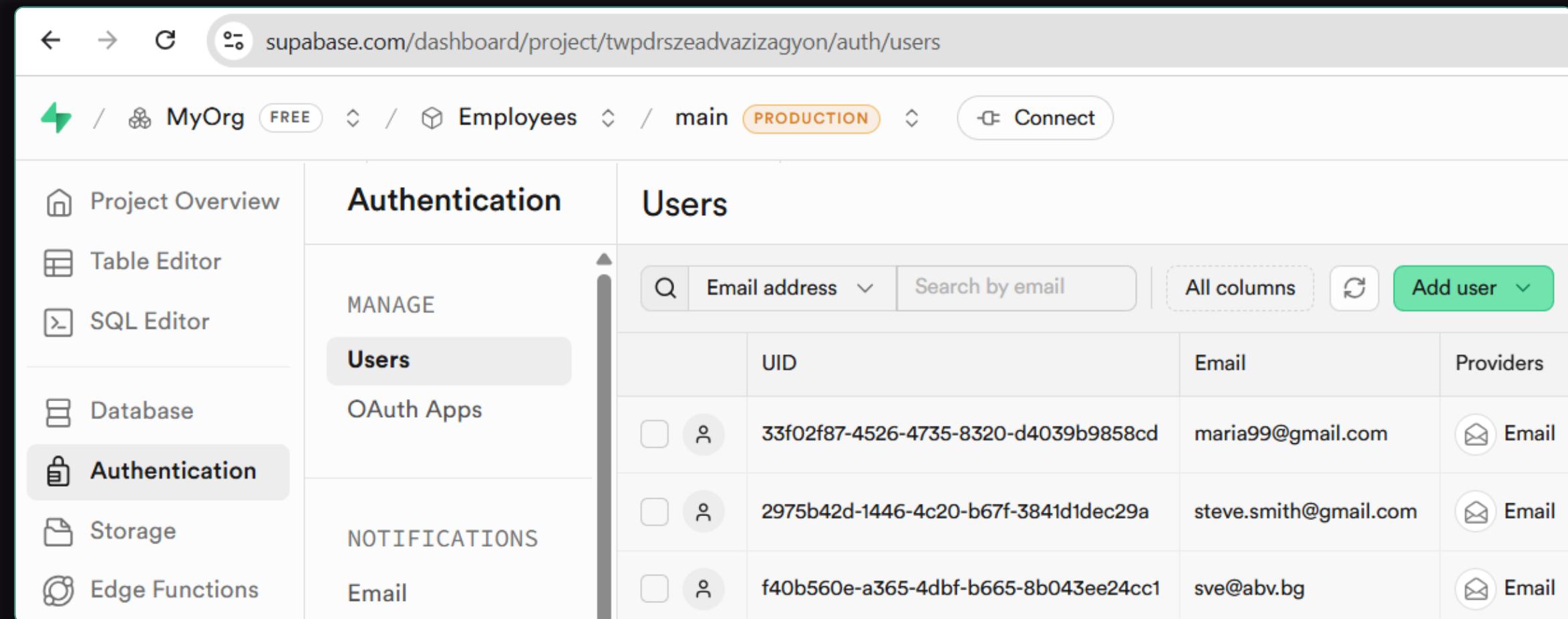


The screenshot shows the Supabase documentation page for Auth. The URL is `supabase.com/docs/guides/auth`. The page title is 'Auth'. It contains the following text:
Auth
Use Supabase to authenticate and authorize your users.

Supabase Auth makes it easy to implement authentication and authorization in your app. We provide client SDKs and API endpoints to help you create and manage users.

Supabase Auth → Users

- Dashboard → Authentication → Users



The screenshot shows the Supabase dashboard interface. The left sidebar has navigation links: Project Overview, Table Editor, SQL Editor, Database, **Authentication** (which is selected and highlighted in grey), Storage, and Edge Functions. The main area has a breadcrumb navigation: MyOrg / Employees / main. The top right shows a PRODUCTION environment switch and a Connect button. The central part is titled "Authentication" and "Users". Under "MANAGE", there are tabs for "Users" (selected) and "OAuth Apps". The "NOTIFICATIONS" section includes an "Email" tab. On the right, a table lists three users:

	UID	Email	Providers
<input type="checkbox"/>	33f02f87-4526-4735-8320-d4039b9858cd	maria99@gmail.com	Email
<input type="checkbox"/>	2975b42d-1446-4c20-b67f-3841d1dec29a	steve.smith@gmail.com	Email
<input type="checkbox"/>	f40b560e-a365-4dbf-b665-8b043ee24cc1	sve@abv.bg	Email

At the top right of the table, there are buttons for "All columns", a refresh icon, and a green "Add user" button with a dropdown menu.

Supabase Auth Configuration

- Create a new user
- Switch off "Confirm email"

Create a new user

Email address

User Password

Auto Confirm User?
A confirmation email will not be sent when creating a user via this form.

Create user

Authentication

- MANAGE
- Users
- OAuth Apps

NOTIFICATIONS

- Email

CONFIGURATION

- Policies
- Sign In / Providers**
- OAuth Server
- Sessions
- Rate Limits
- Multi-Factor

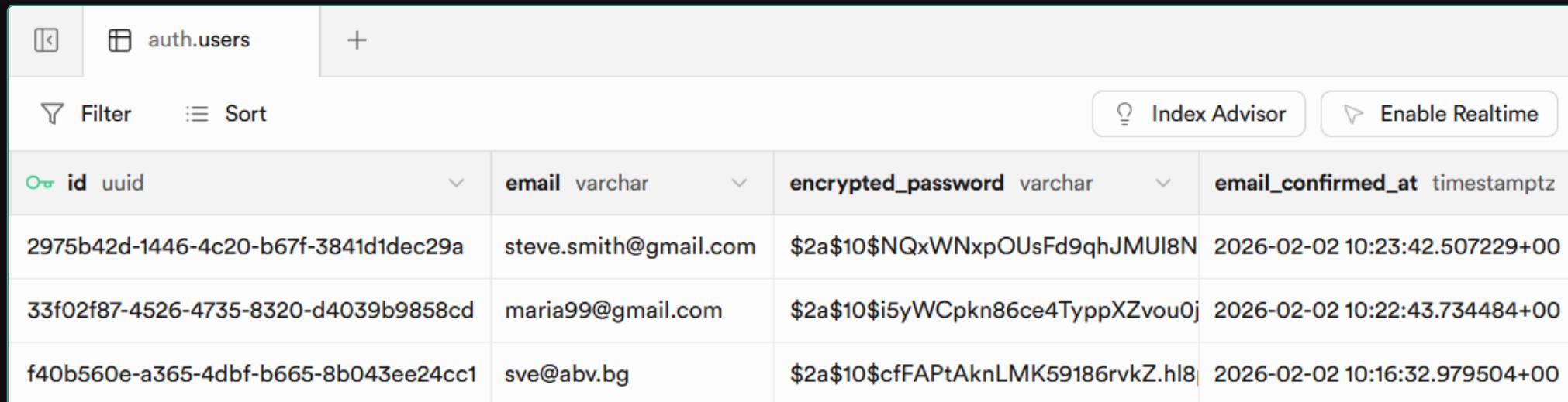
User Signups

- Allow new users to sign up
If this is disabled, new users will not be able to sign up to your application
- Allow manual linking
Enable [manual linking APIs](#) for your project
- Allow anonymous sign-ins
Enable [anonymous sign-ins](#) for your project
- Confirm email
Users will need to confirm their email address before signing in for the first time

Cancel **Save changes**

Where are Users Stored in DB?

- Users, managed by Supabase Auth, are stored in the `auth.users` DB table:



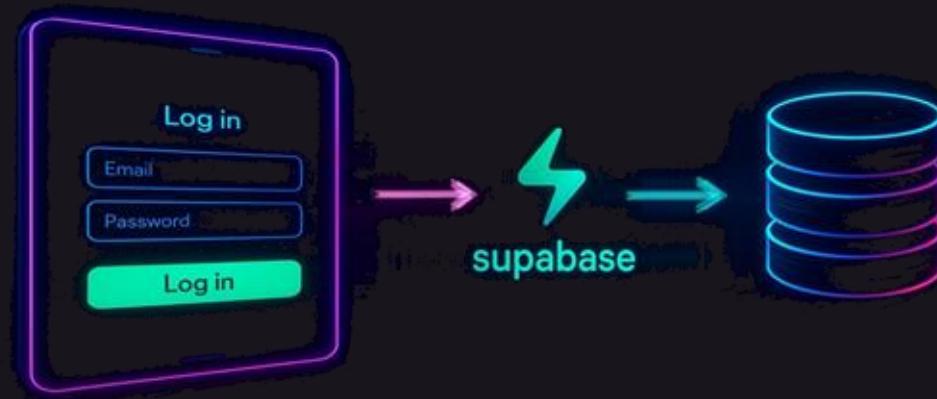
The screenshot shows the Supabase PostgreSQL interface with the 'auth.users' table selected. The table has four columns: id (uuid), email (varchar), encrypted_password (varchar), and email_confirmed_at (timestamptz). There are three rows of data:

id	email	encrypted_password	email_confirmed_at
2975b42d-1446-4c20-b67f-3841d1dec29a	steve.smith@gmail.com	\$2a\$10\$NQxWNxpOUUsFd9qhJMUI8N	2026-02-02 10:23:42.507229+00
33f02f87-4526-4735-8320-d4039b9858cd	maria99@gmail.com	\$2a\$10\$i5yWCpkn86ce4TypXZvouoj	2026-02-02 10:22:43.734484+00
f40b560e-a365-4dbf-b665-8b043ee24cc1	sve@abv.bg	\$2a\$10\$cfFAPtAknLMK59186rvkZ.hl8	2026-02-02 10:16:32.979504+00

- Passwords are encrypted with bcrypt
 - Your app can compare a password with the DB, but cannot decrypt a password back to plaintext

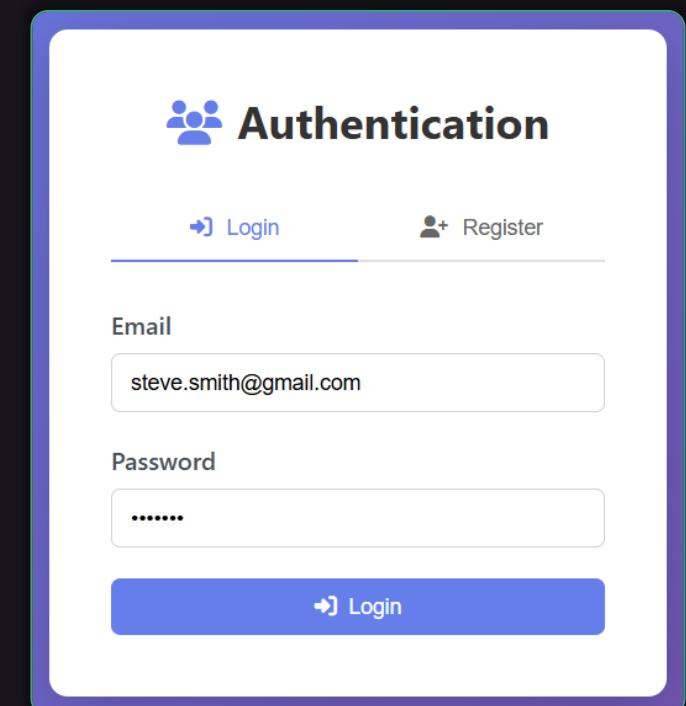
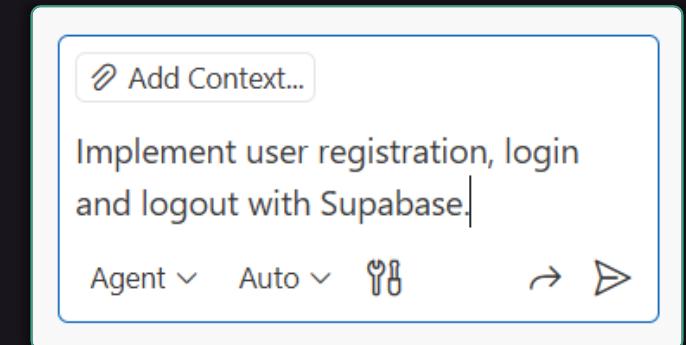
Adding Supabase Authentication

- Implementing **Supabase authentication**
 - User registration, login, logout



- Just run an AI prompt:

Implement **user registration**,
login and **logout** with Supabase.



Contactbook App with Users

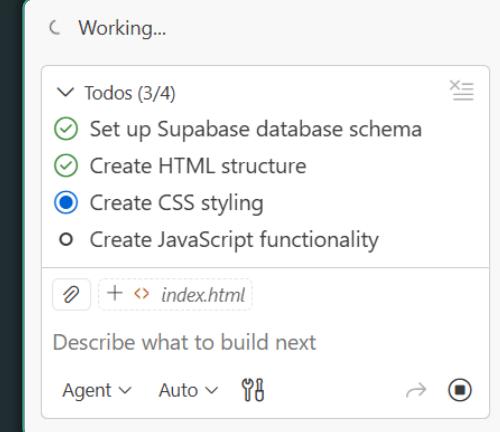
- We want to create a "Contactbook" app
 - Users **register** / **login** and manage their own contacts in their accounts in the app
 - **Contacts** hold **name + phone + email + comments**
- **Step 1:** Create a **Supabase project**
 - Supabase Dashboard → Projects → New project
- **Step 2:** Connect **Supabase MCP** to VS Code
 - Supabase Dashboard → Connect → MCP → VS Code

Contactbook App with Users (2)

- **Step 3:** Create the **Contactbook** app with users:

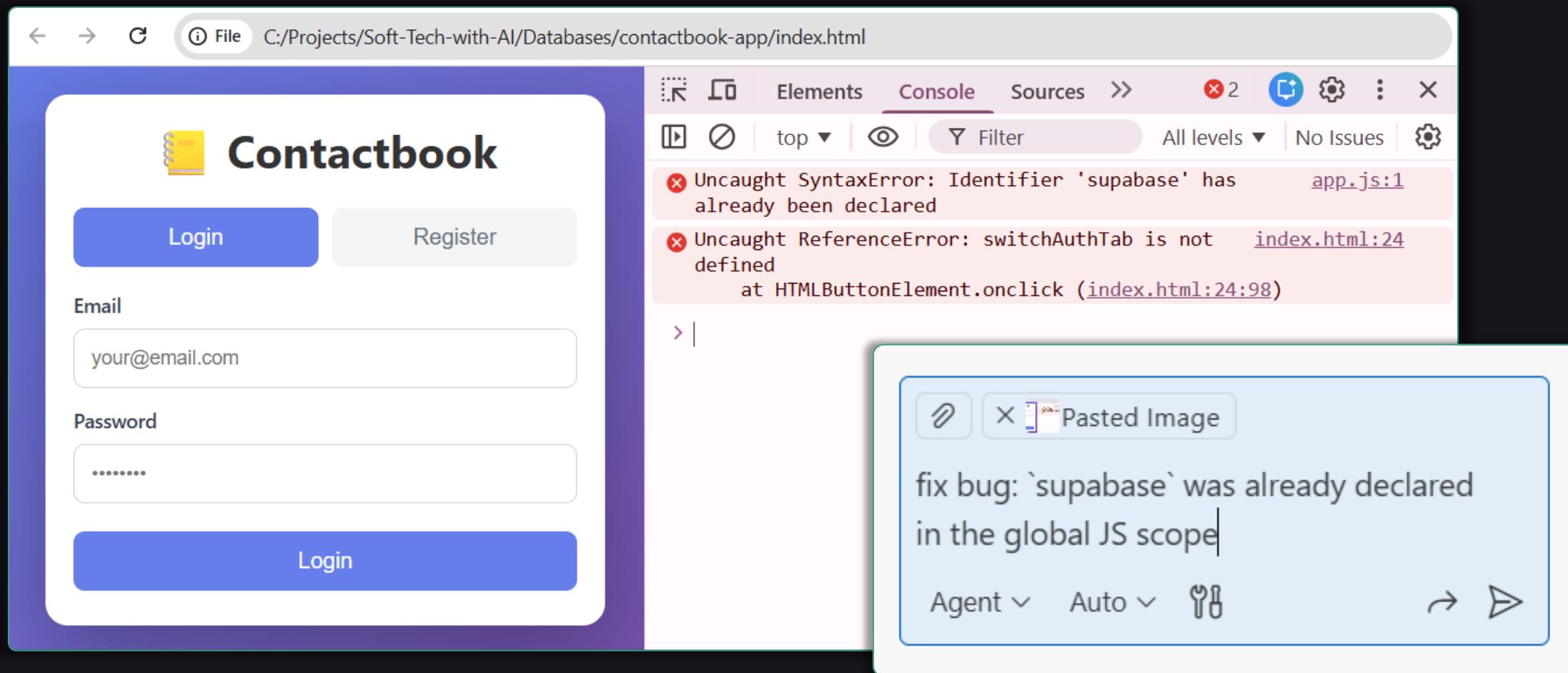
Create a "Contactbook" app: login, view / add / edit / delete contacts, logout

- Use **Supabase as backend** (acess it through the Supabase MCP)
 - Each user has **own list of contacts**, displayed as cards
 - Each **contact** holds **name + email + phone + comments**
- Implement user **register / login / logout** using Supabase Auth
- Use simple **HTML + CSS + JavaScript** code
 - At startup display the **login / register** form
 - After login, **list contacts** as cards (with view / edit / delete buttons)
 - Implement **view, add, edit** and **delete** confirm with **popups**
 - Use **toast notifications** for errors and info messages (auto-close)
 - Implement a **loading spinner** at the top of the app screen



Contactbook App with Users (3)

- Step 4: Run the app and **fix bugs** (if any)



The screenshot shows a web browser window displaying a login page for a "Contactbook" application. The page has a purple header with the "Contactbook" logo and two buttons: "Login" (blue) and "Register" (gray). Below the header is a form with "Email" and "Password" fields, both containing placeholder text ("your@email.com" and "....."). A large blue "Login" button is at the bottom of the form.

The browser's address bar shows the path: C:/Projects/Soft-Tech-with-AI/Databases/contactbook-app/index.html.

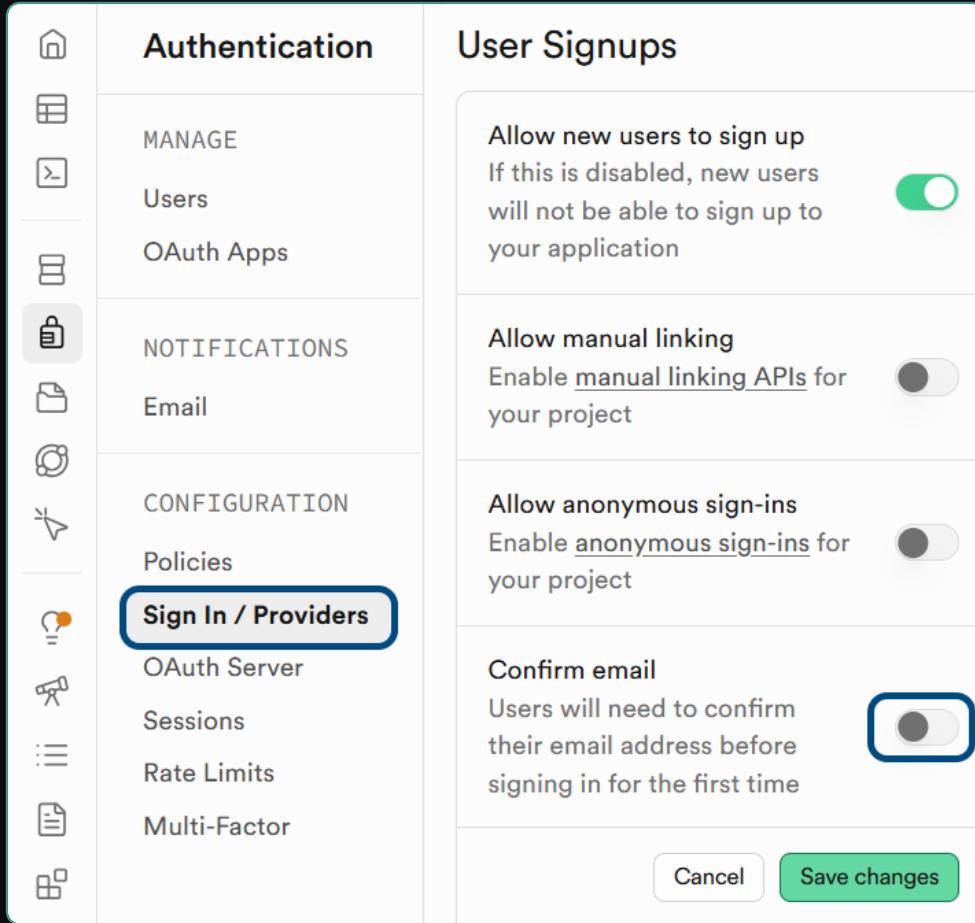
The developer tools' "Console" tab is active, showing two error messages:

- Uncaught SyntaxError: Identifier 'supabase' has already been declared (app.js:1)
- Uncaught ReferenceError: switchAuthTab is not defined (index.html:24)
at HTMLButtonElement.onclick (index.html:24:98)

A callout box in the bottom right corner contains the text: "fix bug: `supabase` was already declared in the global JS scope".

Register and Login in the App

- Switch off "Confirm email" in Supabase



Authentication

MANAGE

- Users
- OAuth Apps

NOTIFICATIONS

- Email

CONFIGURATION

- Policies
- Sign In / Providers**

- OAuth Server
- Sessions
- Rate Limits
- Multi-Factor

User Signups

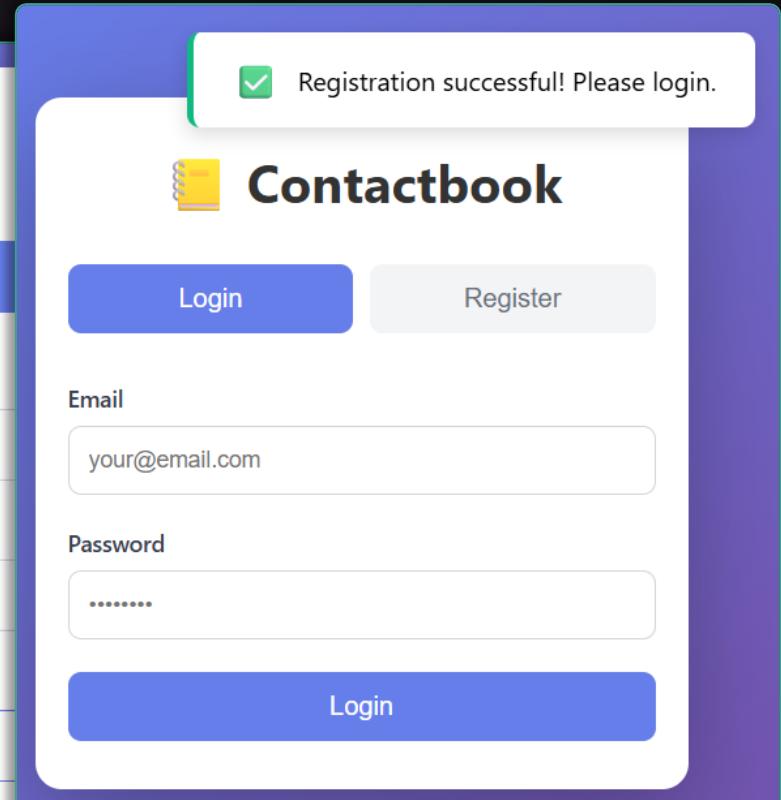
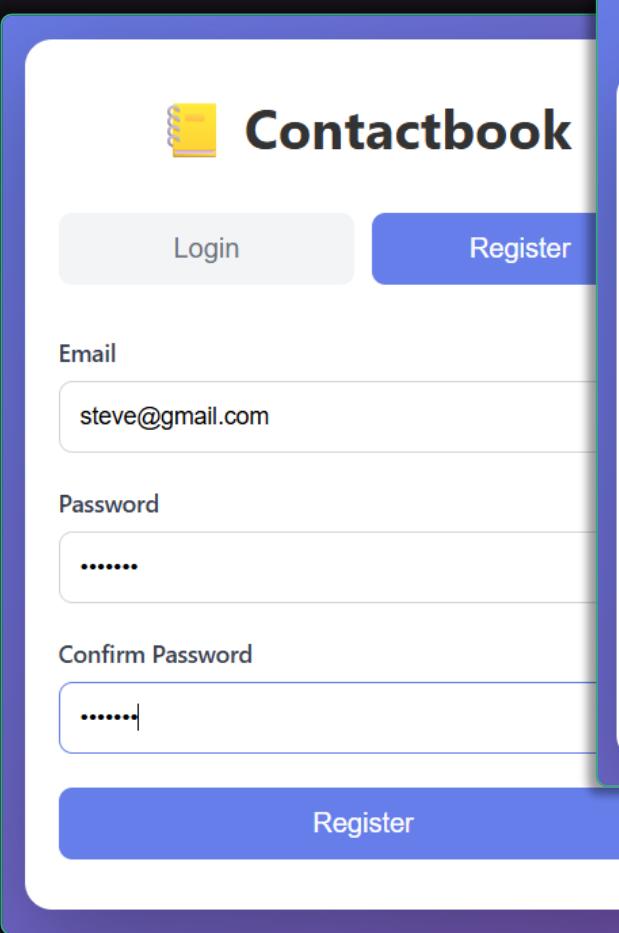
Allow new users to sign up
If this is disabled, new users will not be able to sign up to your application

Allow manual linking
Enable [manual linking APIs](#) for your project

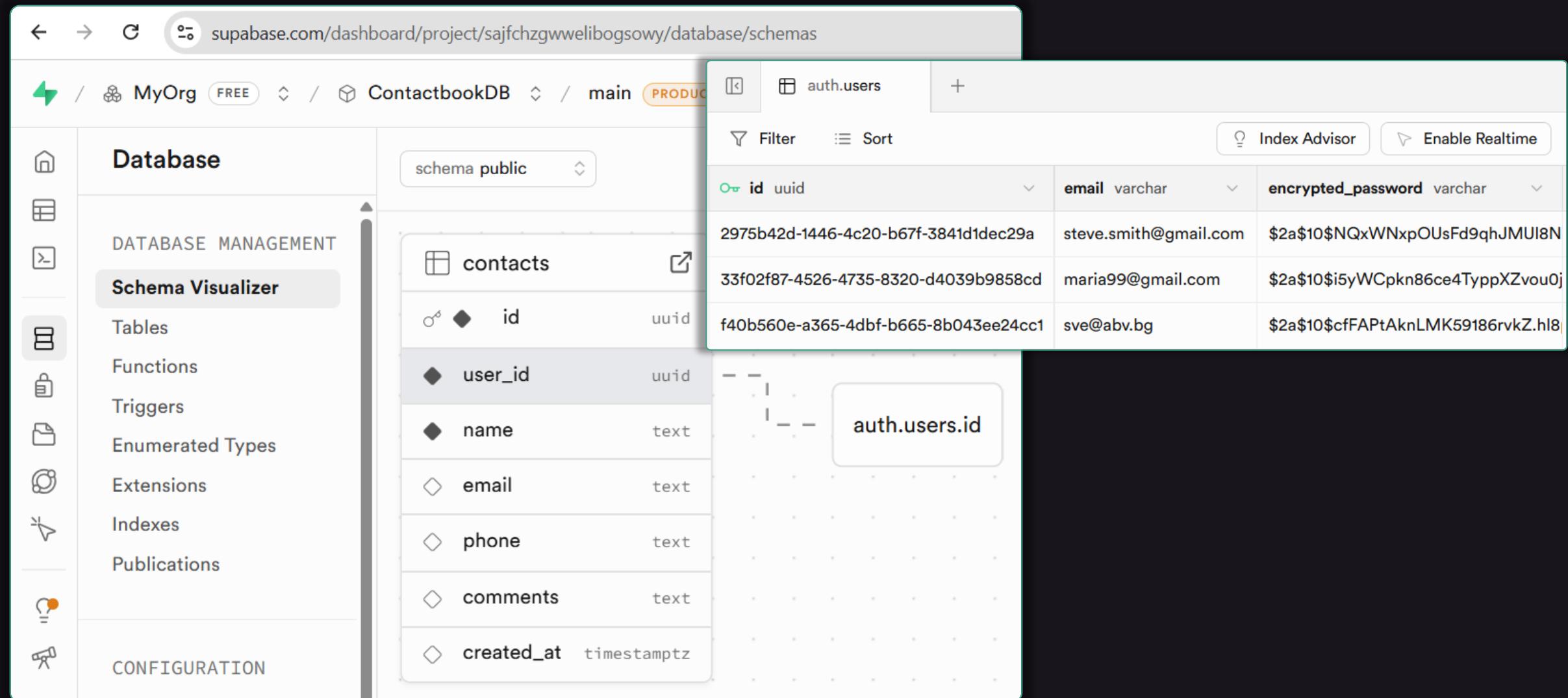
Allow anonymous sign-ins
Enable [anonymous sign-ins](#) for your project

Confirm email
Users will need to confirm their email address before signing in for the first time

Cancel **Save changes**



App Tables in Supabase



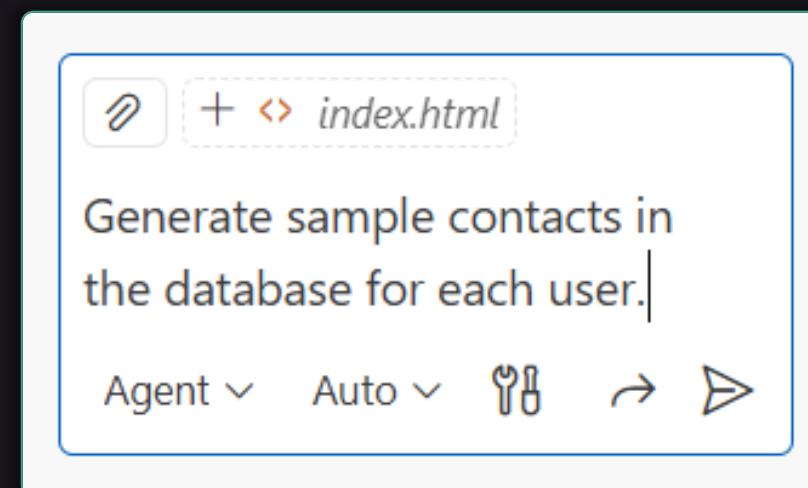
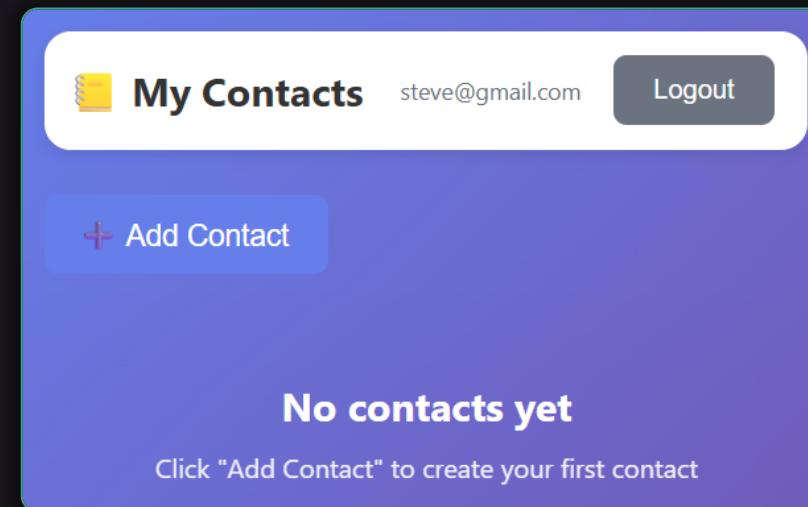
The screenshot shows the Supabase dashboard interface. On the left, there's a sidebar with various management options like Database, Functions, Triggers, and Indexes. The main area shows the schema structure for the 'public' schema. A modal window is open over the 'auth.users' table, which has columns: id (uuid), email (varchar), and encrypted_password (varchar). The modal displays three rows of data:

id	email	encrypted_password
2975b42d-1446-4c20-b67f-3841d1dec29a	steve.smith@gmail.com	\$2a\$10\$NQxWNxpOUSFd9qhJMUI8N
33f02f87-4526-4735-8320-d4039b9858cd	maria99@gmail.com	\$2a\$10\$i5yWCpkn86ce4TyppXZvou0j
f40b560e-a365-4dbf-b665-8b043ee24cc1	sve@abv.bg	\$2a\$10\$cffFAPtAknLMK59186rvkZ.hl8

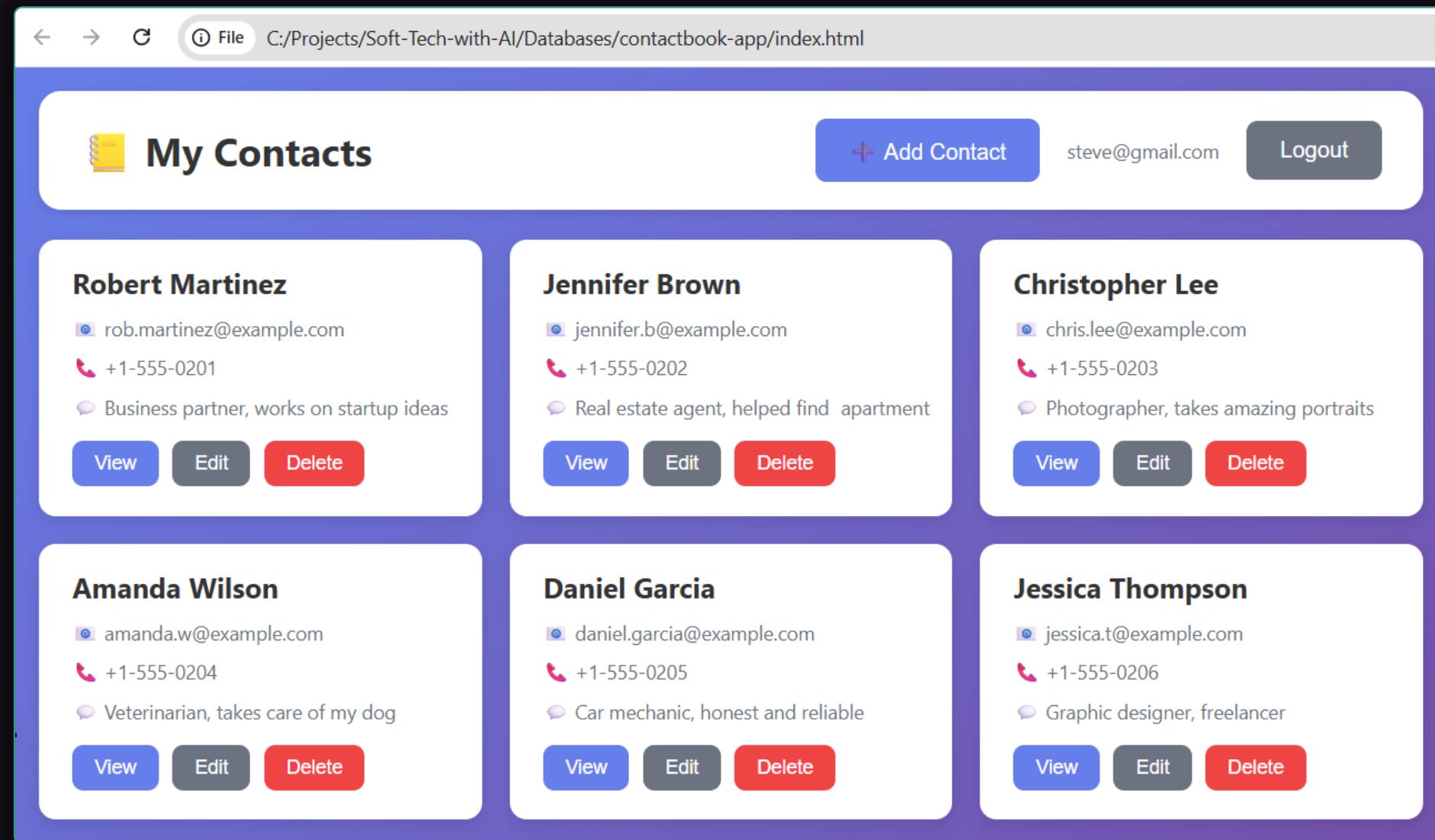
Contactbook App after Login

- The database is initially **empty** (no contacts yet)
- Register a 2-3 **users**
- You can **generate sample data** in the DB with an AI prompt:

Generate **sample contacts** in the database for each user.



Contactbook with Sample Contacts

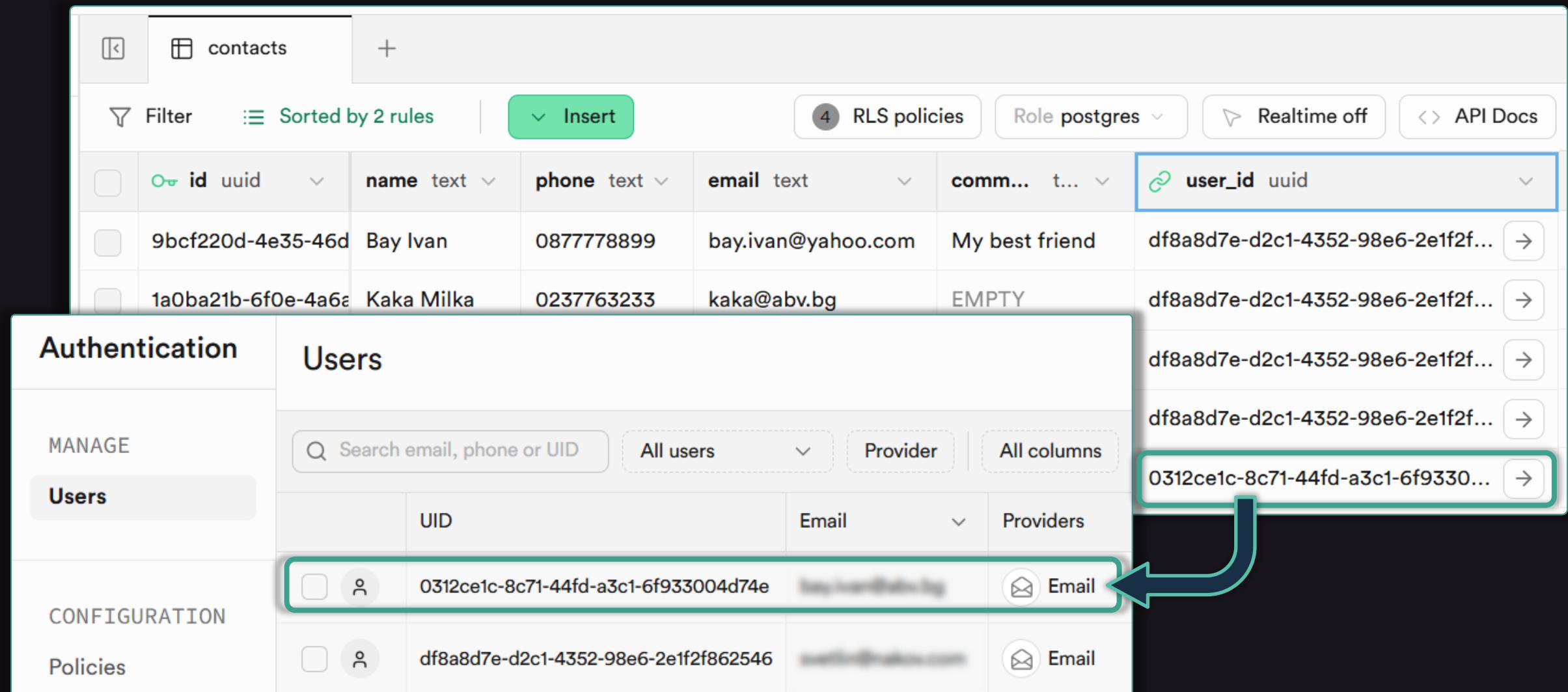


The screenshot shows a web-based contact management application with a purple header and footer. The header includes a back/forward button, a file menu, and the path C:/Projects/Soft-Tech-with-AI/Databases/contactbook-app/index.html. On the right of the header are buttons for 'Add Contact' (with a plus sign), 'steve@gmail.com', and 'Logout'. The main content area displays six contact cards arranged in two rows of three. Each card contains the contact's name, email, phone number, a brief description, and three action buttons: 'View' (blue), 'Edit' (grey), and 'Delete' (red).

Contact Data:

Contact	Email	Phone	Description	Action Buttons
Robert Martinez	rob.martinez@example.com	+1-555-0201	Business partner, works on startup ideas	View Edit Delete
Jennifer Brown	jennifer.b@example.com	+1-555-0202	Real estate agent, helped find apartment	View Edit Delete
Christopher Lee	chris.lee@example.com	+1-555-0203	Photographer, takes amazing portraits	View Edit Delete
Amanda Wilson	amanda.w@example.com	+1-555-0204	Veterinarian, takes care of my dog	View Edit Delete
Daniel Garcia	daniel.garcia@example.com	+1-555-0205	Car mechanic, honest and reliable	View Edit Delete
Jessica Thompson	jessica.t@example.com	+1-555-0206	Graphic designer, freelancer	View Edit Delete

Users and Contacts Tables in the DB



The screenshot shows two database interfaces side-by-side.

Top Interface (Database View):

- Table name: contacts
- Columns: id (uuid), name (text), phone (text), email (text), comments (text), user_id (uuid)
- Data rows:
 - id: 9bcf220d-4e35-46d, name: Bay Ivan, phone: 0877778899, email: bay.ivan@yahoo.com, comments: My best friend, user_id: df8a8d7e-d2c1-4352-98e6-2e1f2f...
 - id: 1a0ba21b-6f0e-4a6a, name: Kaka Milka, phone: 0237763233, email: kaka@abv.bg, comments: EMPTY, user_id: df8a8d7e-d2c1-4352-98e6-2e1f2f...

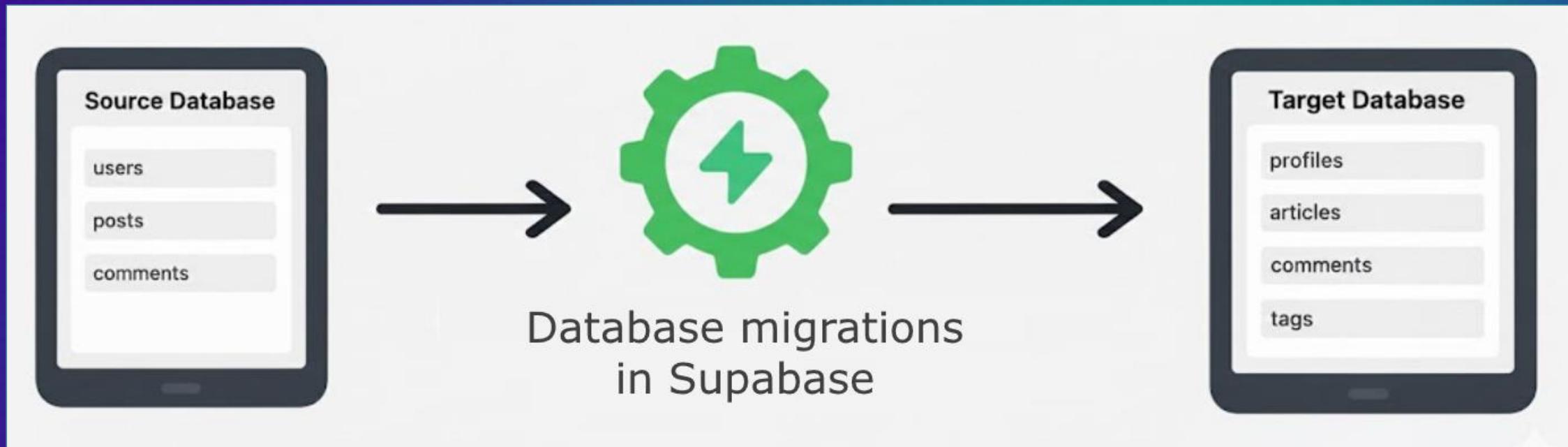
Bottom Interface (Authentication View):

- Section: Authentication
- Sub-section: Users
- Buttons: Manage, Users (highlighted)
- Search bar: Search email, phone or UID
- Filter: All users, Provider, All columns
- Table columns: UID, Email, Providers
- Data rows:
 - UID: 0312ce1c-8c71-44fd-a3c1-6f933004d74e, Email: [redacted], Providers: [redacted]
 - UID: df8a8d7e-d2c1-4352-98e6-2e1f2f862546, Email: [redacted], Providers: [redacted]

A blue arrow points from the highlighted row in the bottom table to the corresponding row in the top table, indicating a relationship between the user and contact records.

Database Migrations

Tracking DB Schema
Modifications in Supabase



Database Migrations

- **Database migrations** are how developers safely version, share, and deploy changes to the PostgreSQL schema
 - Track changes in tables, columns, functions, policies, etc. are stored as **SQL scripts**, ordered by **time**

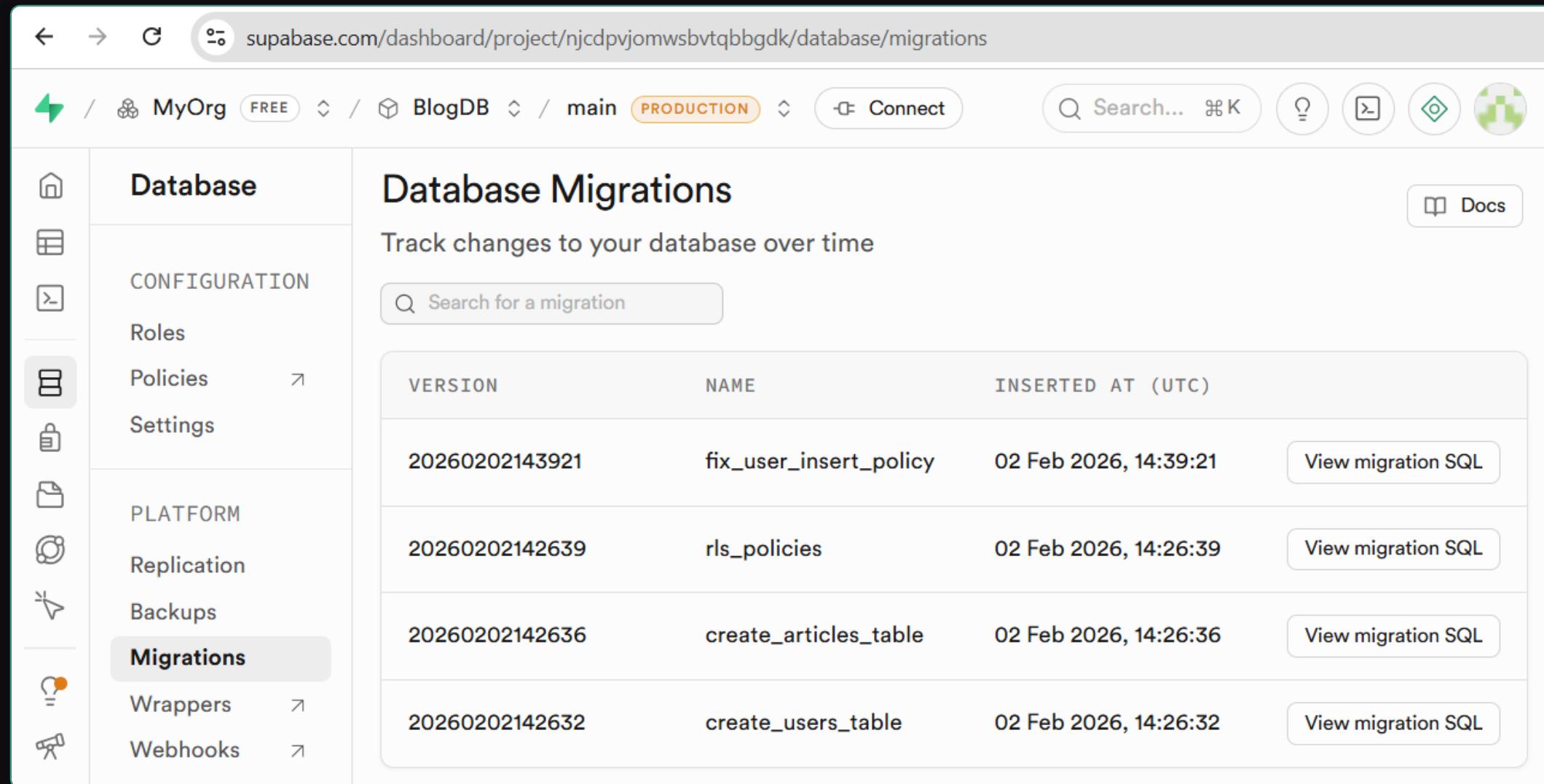
```
✓ migrations
  001_create_users_table.sql
  002_create_articles_table.sql
  003_rls_policies.sql
  004_fix_user_insert_policy.sql
```

```
migrations > 002_create_articles_table.sql
1  -- Create articles table
2  CREATE TABLE articles (
3    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
4    owner_id UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
5    title TEXT NOT NULL,
6    content TEXT NOT NULL,
7    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
8    updated_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
9  );
10
11  -- Create indexes for better query performance
12  CREATE INDEX idx_articles_owner_id ON articles(owner_id);
13  CREATE INDEX idx_articles_created_at ON articles(created_at DESC);
14
15  -- Enable Row Level Security
16  ALTER TABLE articles ENABLE ROW LEVEL SECURITY;
```

Supabase Migrations

- Supabase has its own **migration infrastructure**
 - Local **SQL files** in the `supabase/migrations` folder
 - These files are part of your project **source code**
 - It is committed to **Git** and version tracked
 - Once created, a migration SQL file is **never changed**
 - **Migration history table** in the DB:
supabase_migrations.schema_migrations
 - It holds the migrations already **applied** to the DB schema

Supabase Migrations – Example



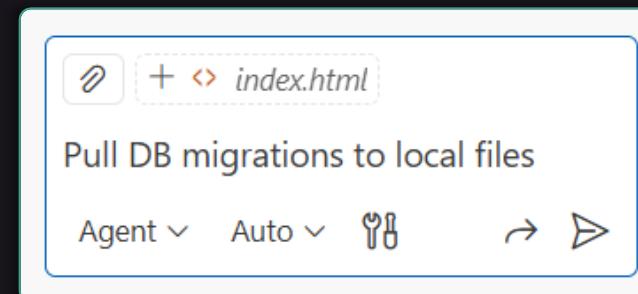
The screenshot shows the Supabase Migrations dashboard. The URL in the browser is supabase.com/dashboard/project/njcdpvjomwsbvtqbbgdk/database/migrations. The interface includes a navigation bar with project details (MyOrg, FREE), database selection (BlogDB, main, PRODUCTION), and connectivity options. The left sidebar features a navigation menu with icons for Home, Configuration, Roles, Policies, Settings, Platform, Replication, Backups, Migrations (which is highlighted in red), Wrappers, and Webhooks. The main content area is titled "Database Migrations" and "Track changes to your database over time". It includes a search bar and a table listing four migrations:

VERSION	NAME	INSERTED AT (UTC)	Actions
20260202143921	fix_user_insert_policy	02 Feb 2026, 14:39:21	View migration SQL
20260202142639	rls_policies	02 Feb 2026, 14:26:39	View migration SQL
20260202142636	create_articles_table	02 Feb 2026, 14:26:36	View migration SQL
20260202142632	create_users_table	02 Feb 2026, 14:26:32	View migration SQL

Working with Migrations

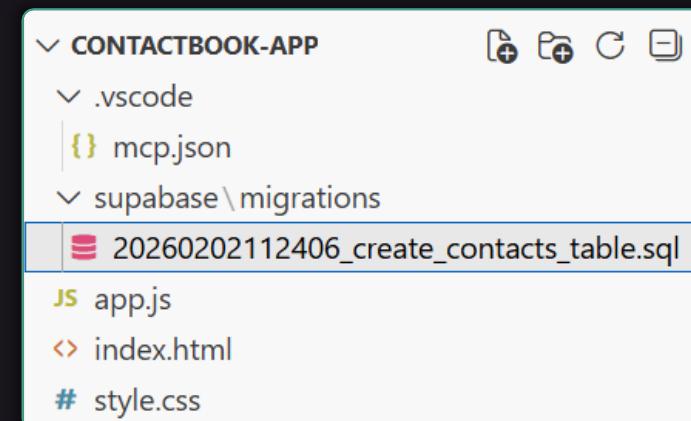
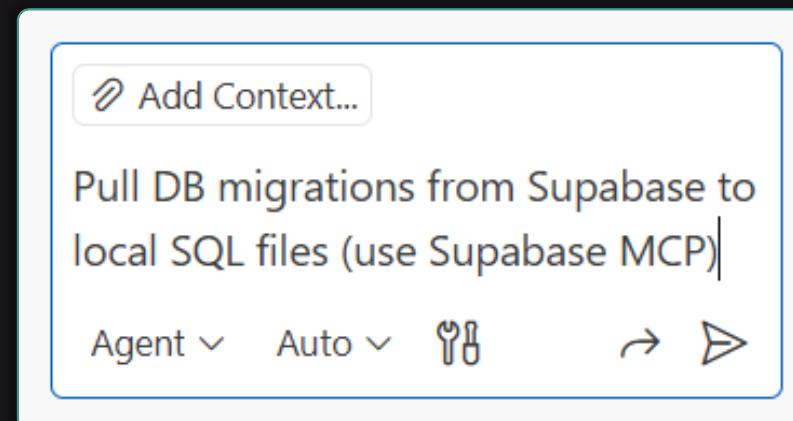
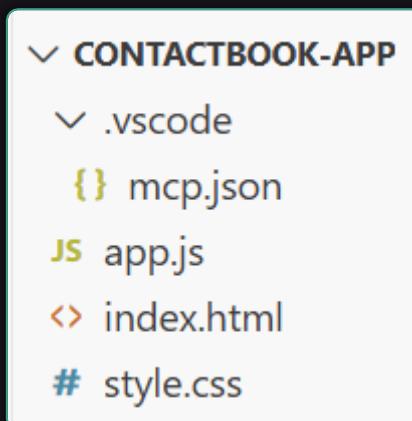
- Supabase MCP typically creates **migrations** for each change it makes in the DB schema
- If not explicitly requested, Supabase migrations stay inside the **database only** (local SQL files are missing)
 - This is **wrong**: DB schema migrations are part of **the code**
- Run this **AI prompt** to store all DB migrations to **local files**

Pull DB migrations
to local files



Example: Contactbook Migrations

- Let's take the **Contactbook app** from the previous example
 - Simple app, which holds contacts in **Supabase** and implements **login, view / add / edit / delete contacts, logout**
 - The app code does not have any DB migrations scripts:
 - Let's **pull the migrations** from Supabase to local SQL files:

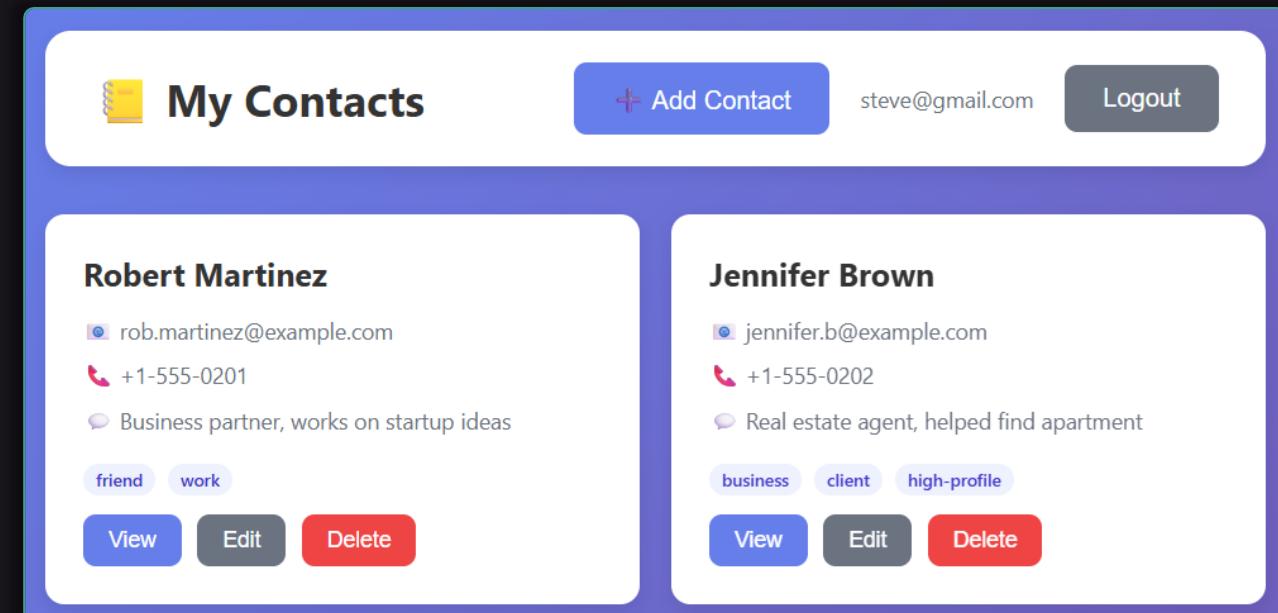


Example: Creating New Migrations

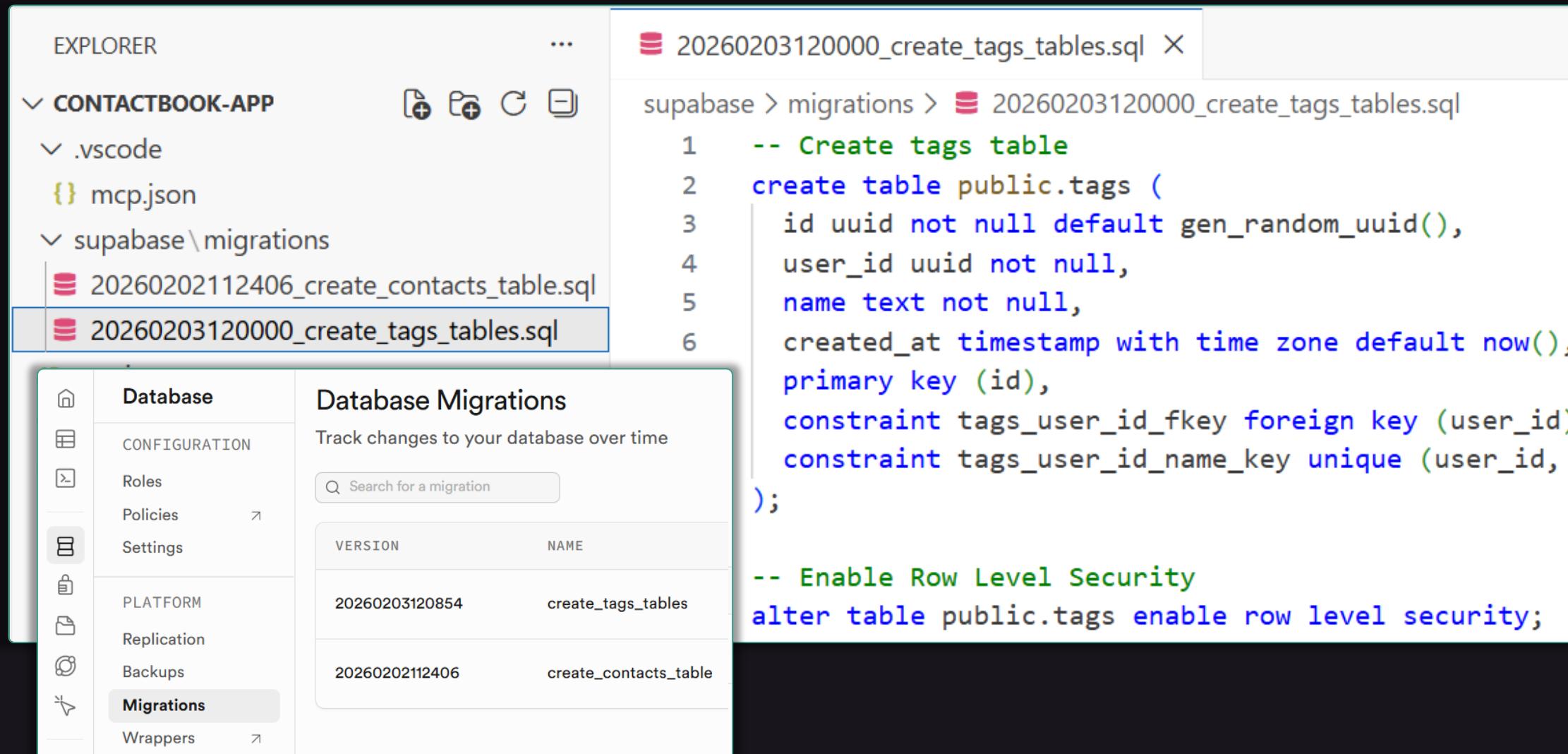
- Let's **implement tags** in the Contactbook app
 - This will create a **migration** to add **new tables** in the DB
 - It will also **extend the app UI** to display and edit tags

Implement tags in the Contactbook **contacts**:

- Add and run migrations to add relevant tables to the DB
- Extend the app **UI** to display and edit tags



Migrations as SQL in the DB



The screenshot shows a development environment with a code editor and a database interface.

EXPLORER pane:

- CONTACTBOOK-APP
- .vscode
- mcp.json
- supabase\migrations
 - 20260202112406_create_contacts_table.sql
 - 20260203120000_create_tags_tables.sql**

Code Editor pane:

File path: supabase > migrations > 20260203120000_create_tags_tables.sql

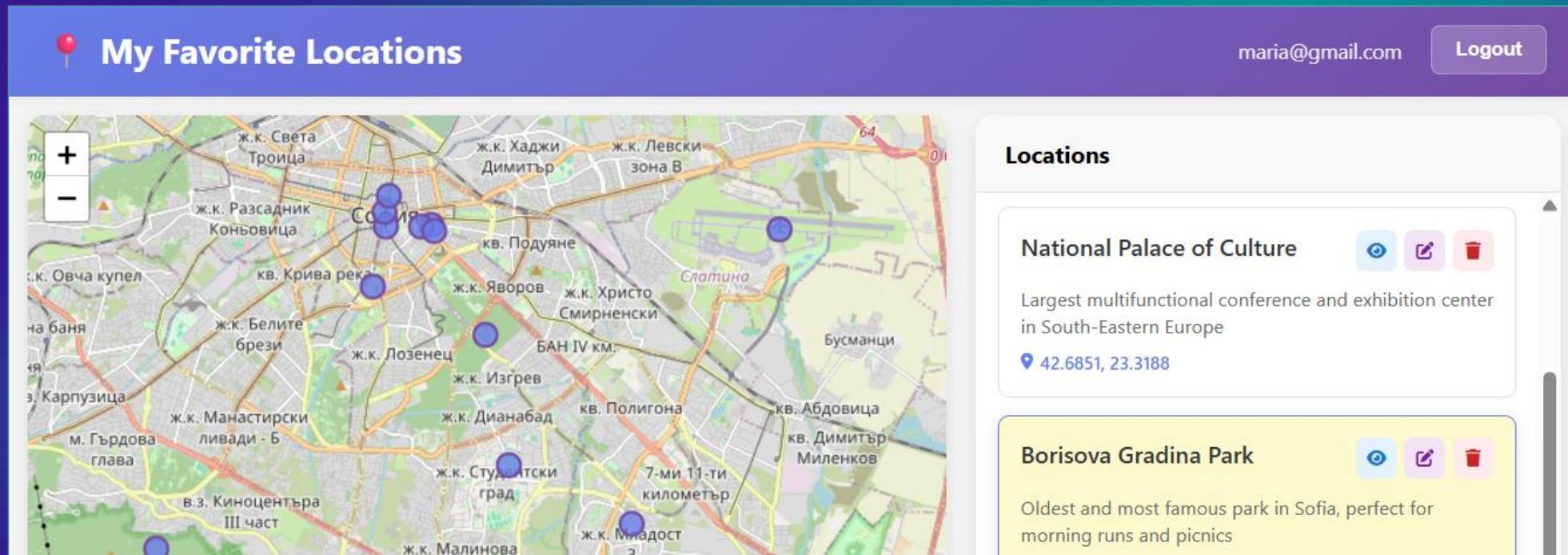
```
1 -- Create tags table
2 create table public.tags (
3   id uuid not null default gen_random_uuid(),
4   user_id uuid not null,
5   name text not null,
6   created_at timestamp with time zone default now(),
primary key (id),
constraint tags_user_id_fkey foreign key (user_id)
constraint tags_user_id_name_key unique (user_id,
);
-- Enable Row Level Security
alter table public.tags enable row level security;
```

Database Migrations interface:

- Database
- VERSION NAME
- 20260203120854 create_tags_tables
- 20260202112406 create_contacts_table

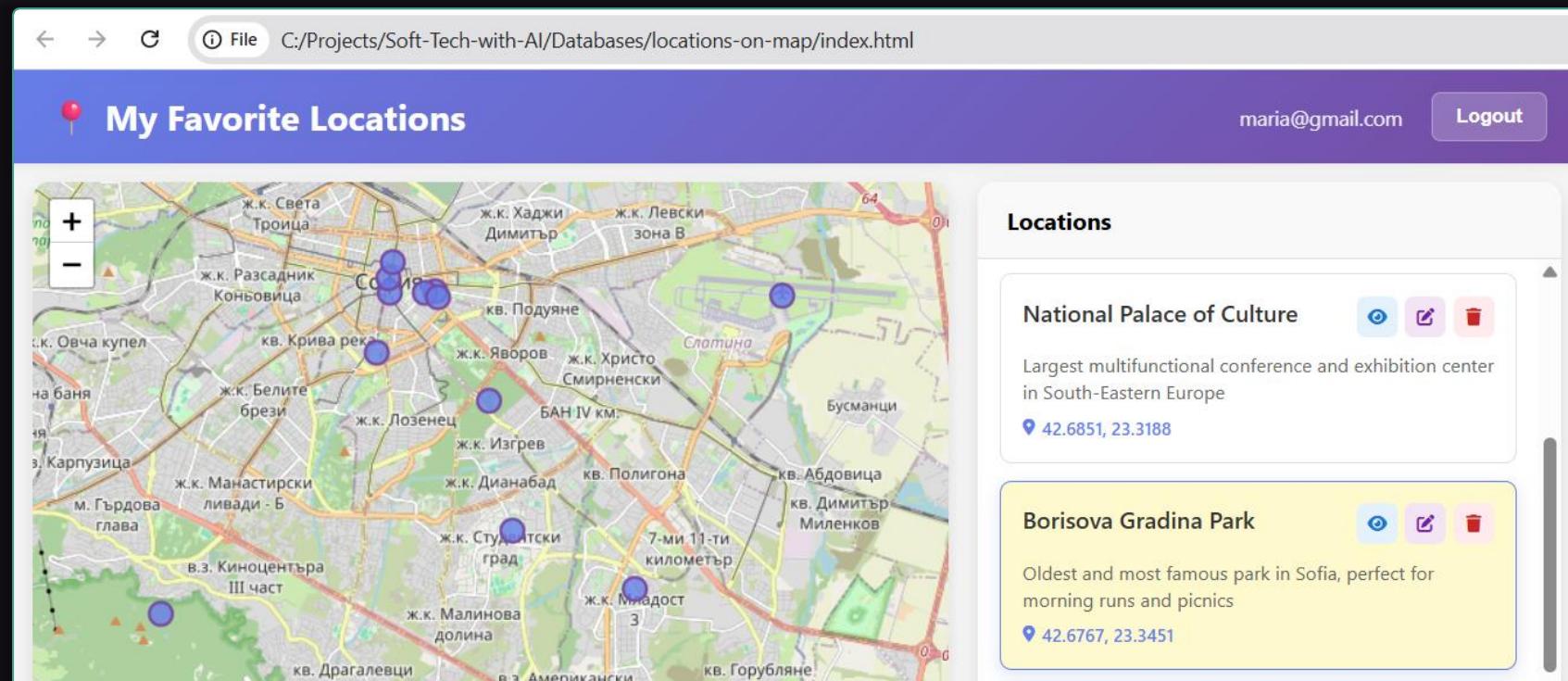
Locations on the Map

Building an App with Supabase Back-End
with Users, Data, RLS Policies and Migrations



Example: Locations on the Map

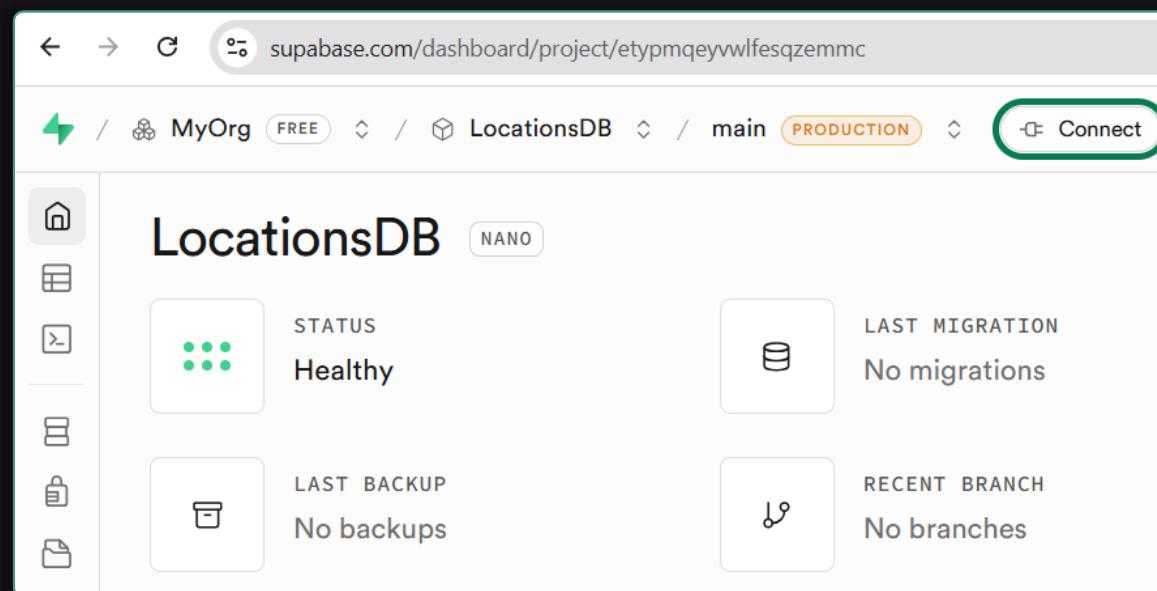
- Let's create an app with **Supabase MCP** and **DB migrations**
- Locations on the Map** app: users log-in and pin locations on a visual map with locations editor of the right



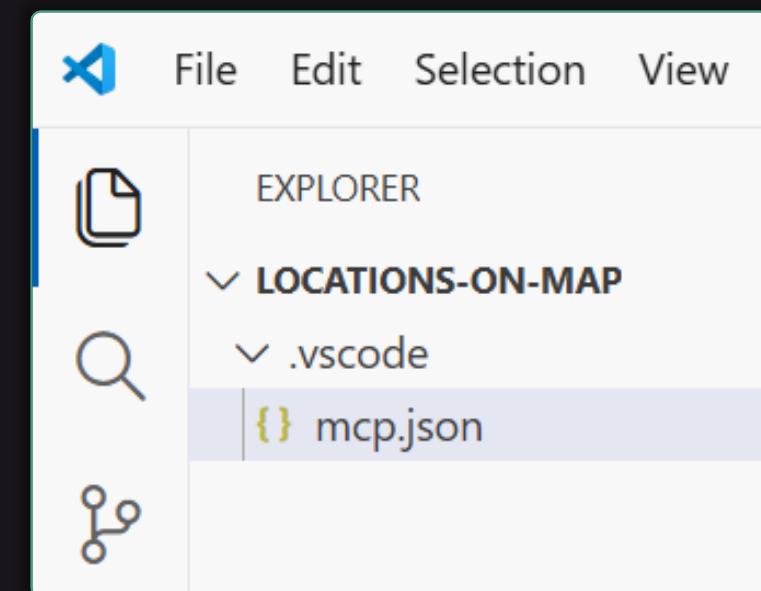
Locations on the Map: Supabase DB

- Step 1: Supabase Project + MCP Server

- Create a new **project in Supabase**
- Install the **Supabase MCP**: Supabase Dashboard → Connect → MCP → VS Code → Install in Workspace → Authorize



The screenshot shows the Supabase Dashboard for a project named "LocationsDB". The status is listed as "Healthy". Other metrics shown include "LAST MIGRATION" (No migrations), "RECENT BRANCH" (No branches), and "LAST BACKUP" (No backups). The "Connect" button is highlighted with a green oval.



The screenshot shows the VS Code Explorer sidebar. It lists a folder named "LOCATIONS-ON-MAP" which contains ".vscode" and "mcp.json". The "mcp.json" file is currently selected, indicated by a blue highlight.

Locations on the Map: DB Schema

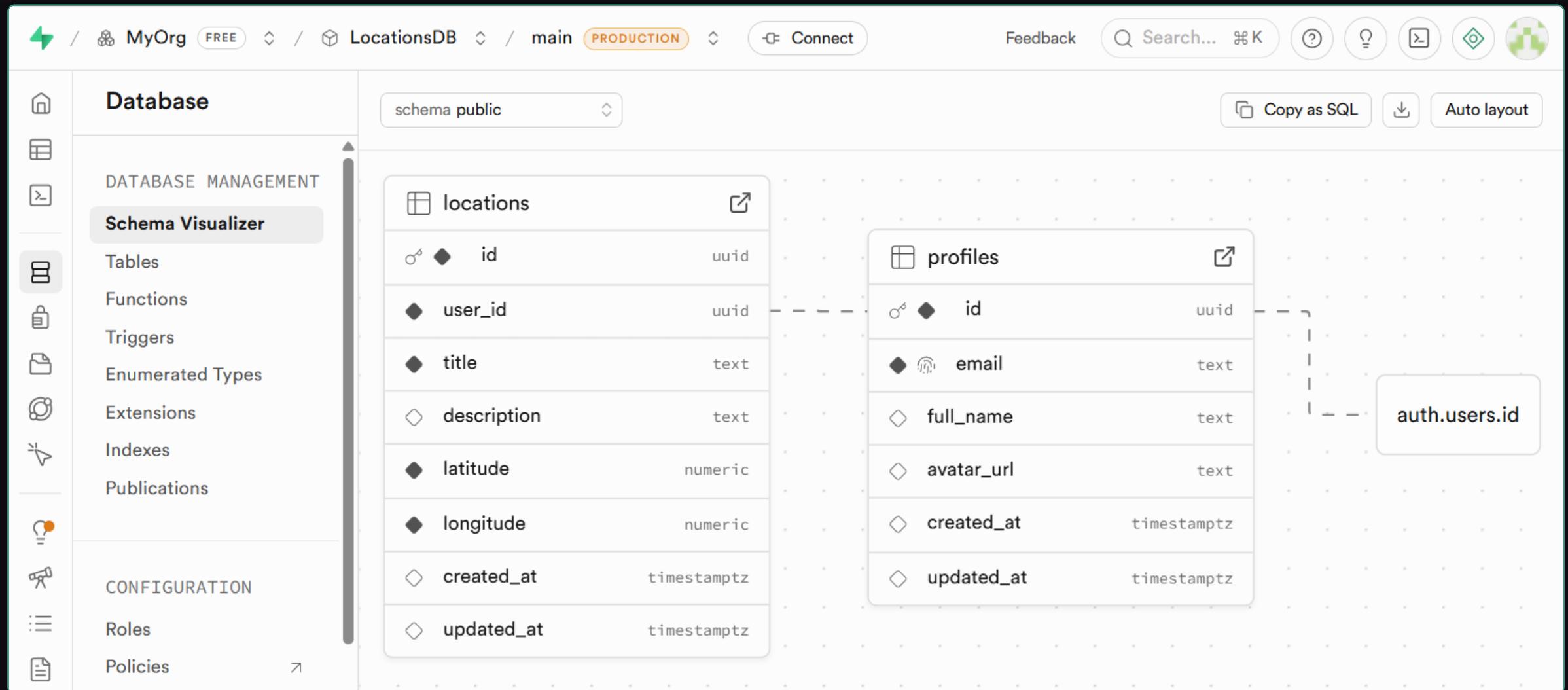


- Step 2: Create the DB Schema for the app

Create the DB schema for the app "Locations on the Map"

- Use Supabase as backend to store users and locations
- Create a DB schema to hold locations on a map
 - Users have their own lists of locations
 - Each location holds GPS coordinates + title + description
- Use the Supabase MCP to access the DB directly
- Keep track of DB migrations as local SQL files
- Define RLS policies to protect users' data

Locations DB Schema – Example



Locations on the Map: Create Users



- Step 3: Create **users** in Supabase
 - Using Supabase Studio, **register sample users** in the Supabase Auth system:
 - maria@gmail.com / pass123
 - steve@gmail.com / pass123
 - **Note:** Supabase MCP cannot register users

The image shows two screenshots of the Supabase Studio interface. The top screenshot is a modal titled 'Add user' with options to 'Send invitation' or 'Create new user'. It includes a date and time stamp: 'Tue 05 Feb 2024 10:25:24 GMT +0200'. The bottom screenshot shows the 'Authentication' section of the 'Users' table. The table has columns for UID, Email, and Providers. Two users are listed: one with UID 'c887c115-7b96-43b0-b8b4-eb0def4af001' and Email 'maria@gmail.com', and another with UID '8b81f735-3306-43a4-b695-9847c32c4ca' and Email 'steve@gmail.com'. The 'Users' tab under 'MANAGE' is selected.

UID	Email	Providers
c887c115-7b96-43b0-b8b4-eb0def4af001	maria@gmail.com	Email
8b81f735-3306-43a4-b695-9847c32c4ca	steve@gmail.com	Email

Locations on the Map: Sample Data



- Step 4: Generate **sample data**
 - Using Supabase MCP, **generate sample data** in the DB:

Generate **sample data** in the DB:

- You have already 2 **existing users**
- Generate **5-6 locations** for each user:
- Choose **GPS locations** in Sofia (Bulgaria)
- Assign relevant **titles** and **descriptions**

The screenshot shows the Supabase MCP interface with a table titled "locations". The table has columns: id (uuid), title (text), description (text), latitude (num...), and longitude (num...). There are three rows of data:

id	title	description	latitude	longitude
03f8750e-796d-46ba-90a9-db549fdd469	Vitosha Boulevard	Main shopping street with cafes, restaurants, and parks.	42.69525000	23.32189000
2eddae58-1d4c-49af-8c45-4e4691c98eb5	Students' City	Large student residential complex with modern facilities.	42.65417000	23.35083000
2ee9616e-704e-478c-928e-57d1dedf003f	Serdika Metro Station	Metro station with ancient Roman ruins visible.	42.69783000	23.32144000

The screenshot shows a "Add Context..." dialog box. It contains the same instructions as the main slide: "Generate sample data in the DB:" followed by the four bullet points. At the bottom, there are buttons for "Agent" (set to "Auto"), a "Save" icon, and two navigation arrows.

Locations on the Map: Create the App

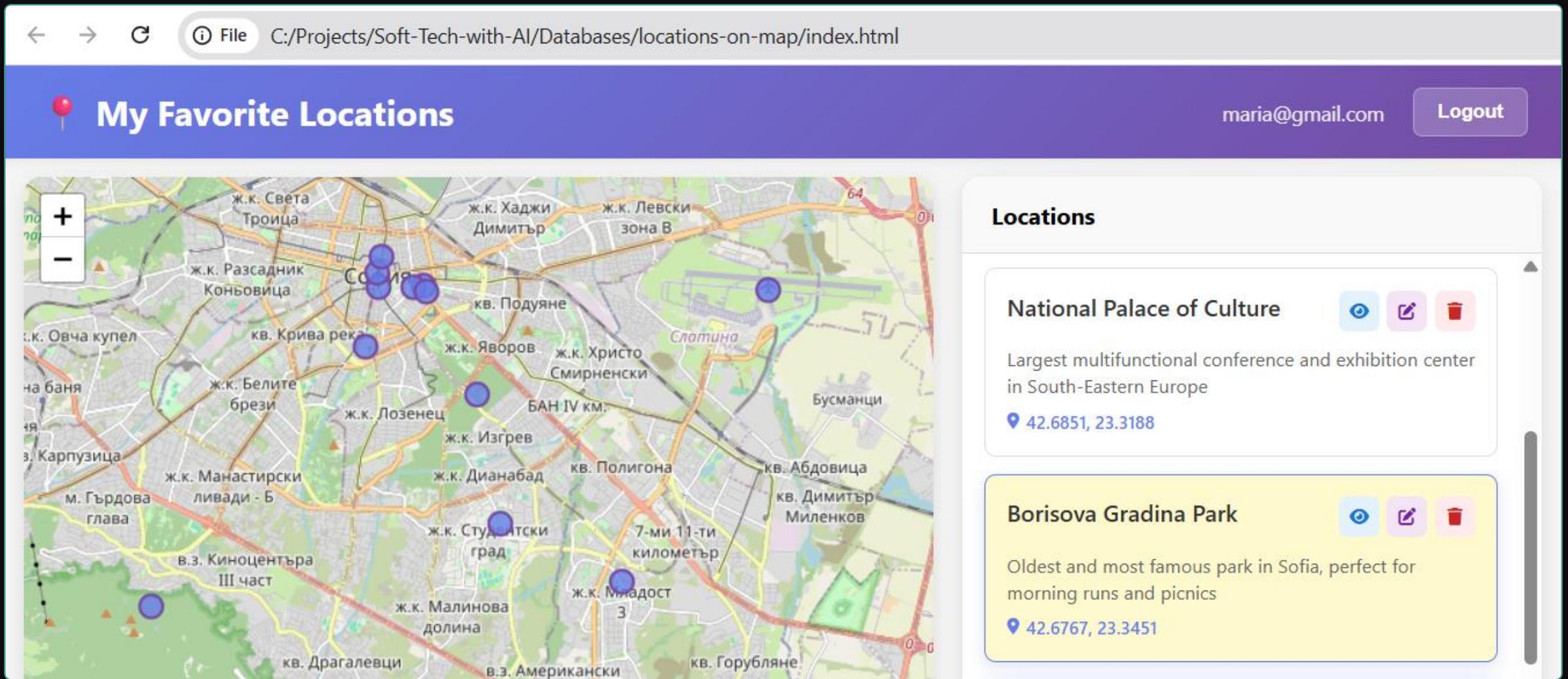


- **Step 5:** Create the app "**Locations on the Map**"

Create an app to visualize favorite locations on a map

- Features: **login / logout**, **manage** locations, **visualize** as pins
- Use simple **HTML + CSS + JavaScript** code
- Use a standard **map UI** component like **Leaflet**
- Display the map visually with **pins** (pinned locations)
- All **locations** are shown on the right side as list of **cards**
 - Click on the map to **create a pin** at certain location (use a popup)
 - Implement **view**, **edit** and **delete** location cards with popups
 - **Pin click** shows the location on the right; **location click** shows the pin
 - Use **icons** to make view, edit and delete buttons small

Locations on the Map – The App



The screenshot shows a web application interface for managing favorite locations. At the top, there's a header bar with navigation icons (back, forward, refresh), a file menu, and the URL <C:/Projects/Soft-Tech-with-AI/Databases/locations-on-map/index.html>. On the left, a sidebar titled "My Favorite Locations" displays a map of Sofia, Bulgaria, with several blue circular markers indicating saved locations. The map includes labels for neighborhoods like "София", "ж.к. Света Троица", "ж.к. Разсадник Конювица", "ж.к. Белите брези", "ж.к. Манастирски ливади - Б", "ж.к. Студентски град", "ж.к. Малинова долина", "ж.к. Младост", "ж.к. Изгрев", "ж.к. Лозенец", "ж.к. Яворов", "ж.к. Христо Смирненски", "кв. Полигона", "кв. Абдoviца", "кв. Димитър Миленков", and "кв. Горубляне". On the right, a sidebar titled "Locations" lists two saved locations: "National Palace of Culture" and "Borisova Gradina Park". Each location entry includes a description, coordinates, and three small circular icons for edit, delete, and other actions.

My Favorite Locations

Logout

maria@gmail.com

National Palace of Culture

Largest multifunctional conference and exhibition center in South-Eastern Europe

42.6851, 23.3188

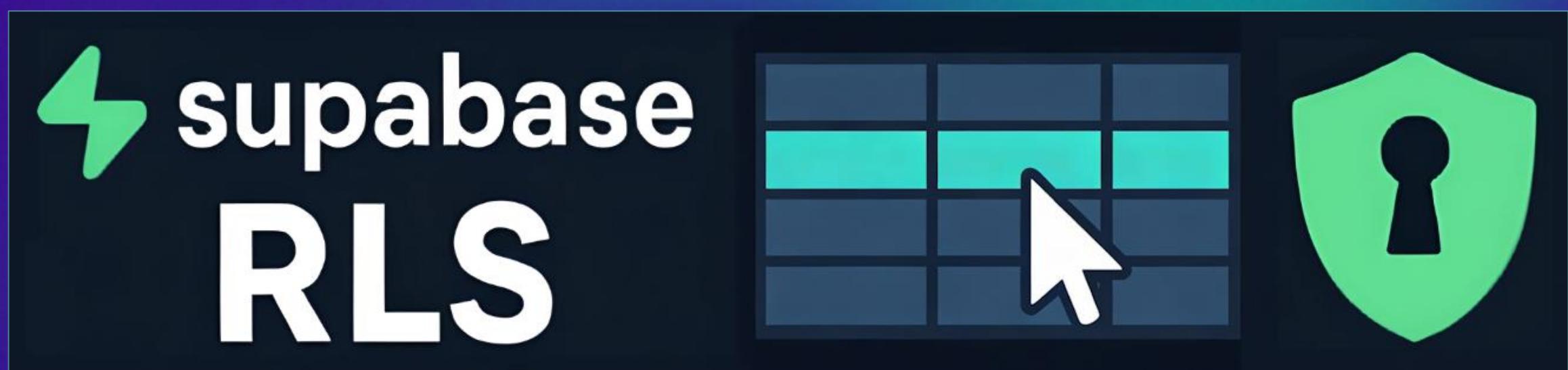
Borisova Gradina Park

Oldest and most famous park in Sofia, perfect for morning runs and picnics

42.6767, 23.3451

Row-Level Security (RLS)

Defining Database-Level Access Control
SQL CREATE POLICY



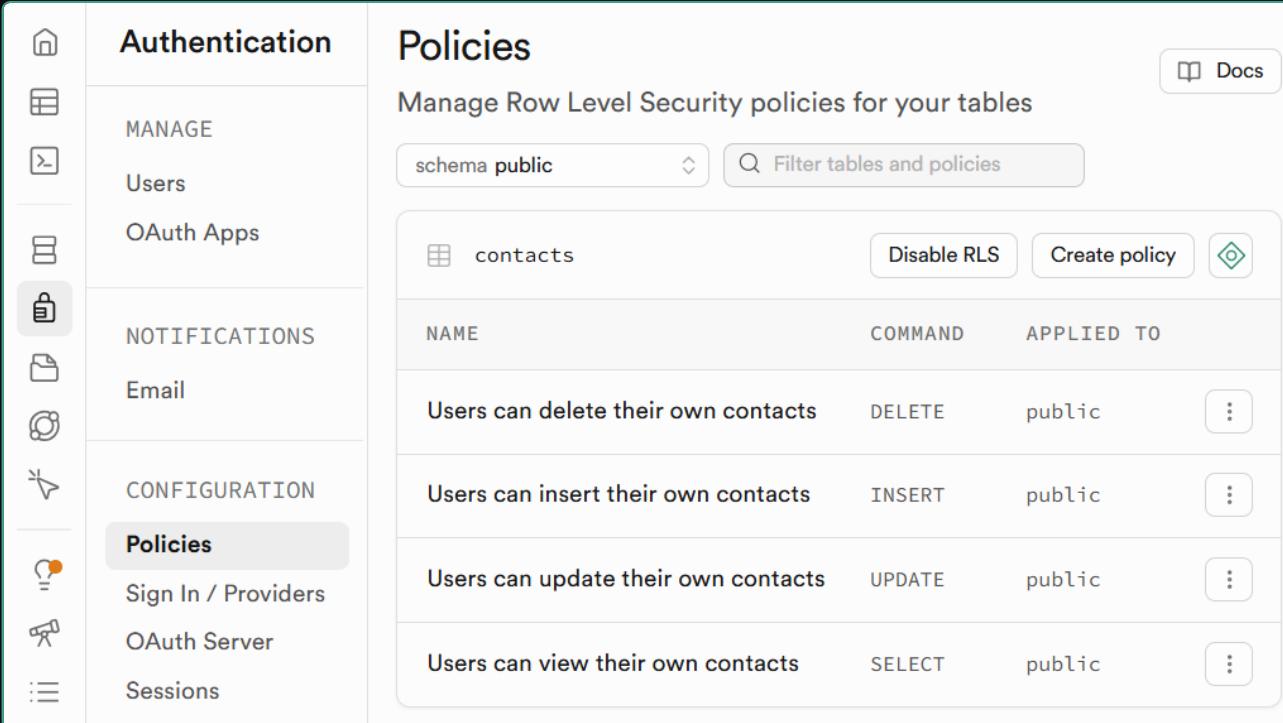
Row-Level Security (RLS)



- **Row-Level Security** (RLS) defines which rows a user can read or write (access control)
 - Defined by **security policies** in the database (SQL code)
 - Based on the **logged-in user**: anonymous / authenticated
- Typical RLS policy:
 - **Anonymous users**: can **read all public data**, e. g. products
 - **Authenticated users**: can read / write their **own data**, e. g. own products, own messages, own user profile
 - **Admin users**: can read / write **any data** in the DB

Sample RLS Policies

- Sample **RLS policies** in Supabase:



The screenshot shows the Supabase RLS Policies interface. On the left, there's a sidebar with icons for Home, Tables, OAuth Apps, Notifications, Configuration, Policies (which is selected), Sign In / Providers, OAuth Server, and Sessions. The main area has a title "Policies" and a subtitle "Manage Row Level Security policies for your tables". It shows a table for the "contacts" table with four rows of policies:

NAME	COMMAND	APPLIED TO
Users can delete their own contacts	DELETE	public
Users can insert their own contacts	INSERT	public
Users can update their own contacts	UPDATE	public
Users can view their own contacts	SELECT	public

CREATE POLICY "Users can update their contacts" ON contacts FOR UPDATE USING ((auth.uid() = user_id));

Blog System with Users and RLS

- We want to create a "**Blog**" web site
 - Site **visitors** can **view** and **read** all published articles
 - Visitors can **register / login**
 - Logged-in **users** can **post** articles, **edit / delete** own articles
 - **Articles** hold **title + owner + date + content**
- **Step 1:** Create a **Supabase project**
 - Supabase Dashboard → Projects → New project
- **Step 2:** Connect **Supabase MCP** to VS Code
 - Supabase Dashboard → Connect → MCP → VS Code

Blog System with Users and RLS (2)



- **Step 3:** Create the **Blog web site** with users and RLS:

Create a "**Blog**" web site to view / publish blog articles:

- Site **visitors** can **view** and **read** all published articles
- **Articles** hold **title** + **owner** + **date** + **content**
- Site visitors can **register** / **login**
- Logged-in **users** can **post** articles, **edit** / **delete** own articles
- Hold **app data** (users, articles) in **Supabase DB**
 - Keep **DB migrations** in a local folder, synced with the DB
 - Define **RLS policies**: users create, edit, delete their own data
 - Generate **sample data** in the DB: a few blog articles (anonymous)
 - Use simple **HTML** + **CSS** + **JS**

Blog System with Users and RLS (3)



← → ⌛ File C:/Projects/Soft-Tech-with-AI/Databases/blog-system/public/index.html

Blog

Welcome, Svetlin Nakov

Logout

Published Articles

Vibe Coding: Build a Product in One Weekend

Svetlin Nakov
February 2, 2026

Few days ago I was a speaker at the Crossroads 2025 startup conference in Sofia. My talk was about vibe coding a how it changes the startup landscape....

[Read More](#) [Edit](#) [Delete](#)

Debugging Techniques Every Developer Needs

Carol Williams
February 2, 2026

Debugging is an essential skill for every developer. Whether you're using browser developer tools, IDE debuggers, or logging statements, effective deb...

[Read More](#)

Delete Article

Are you sure you want to delete this article?
This action cannot be undone.

Edit Article

Title: Vibe Coding: Build a Product in One Weekend

Content: Few days ago I was a speaker at the Crossroads 2025 startup conference in Sofia. My talk was about vibe coding a how it changes the startup landscape.

About the Session
Not long ago, building software meant hiring developers or spending years mastering complex technical skills. Today, the vibe coding paradigm is redefining what it means to create apps. With just

[Save Changes](#)

Blog System: RLS Policies

```
-- RLS Policies for articles table
-- Anyone can read published articles
CREATE POLICY "Anyone can read articles" ON articles
    FOR SELECT USING (true);

-- Users can only insert articles for themselves
CREATE POLICY "Users can create their own articles" ON articles
    FOR INSERT WITH CHECK (auth.uid() = owner_id);

-- Users can only update their own articles
CREATE POLICY "Users can update their own articles" ON articles
    FOR UPDATE USING (auth.uid() = owner_id);

-- Users can only delete their own articles
CREATE POLICY "Users can delete their own articles" ON articles
    FOR DELETE USING (auth.uid() = owner_id);
```

Policies

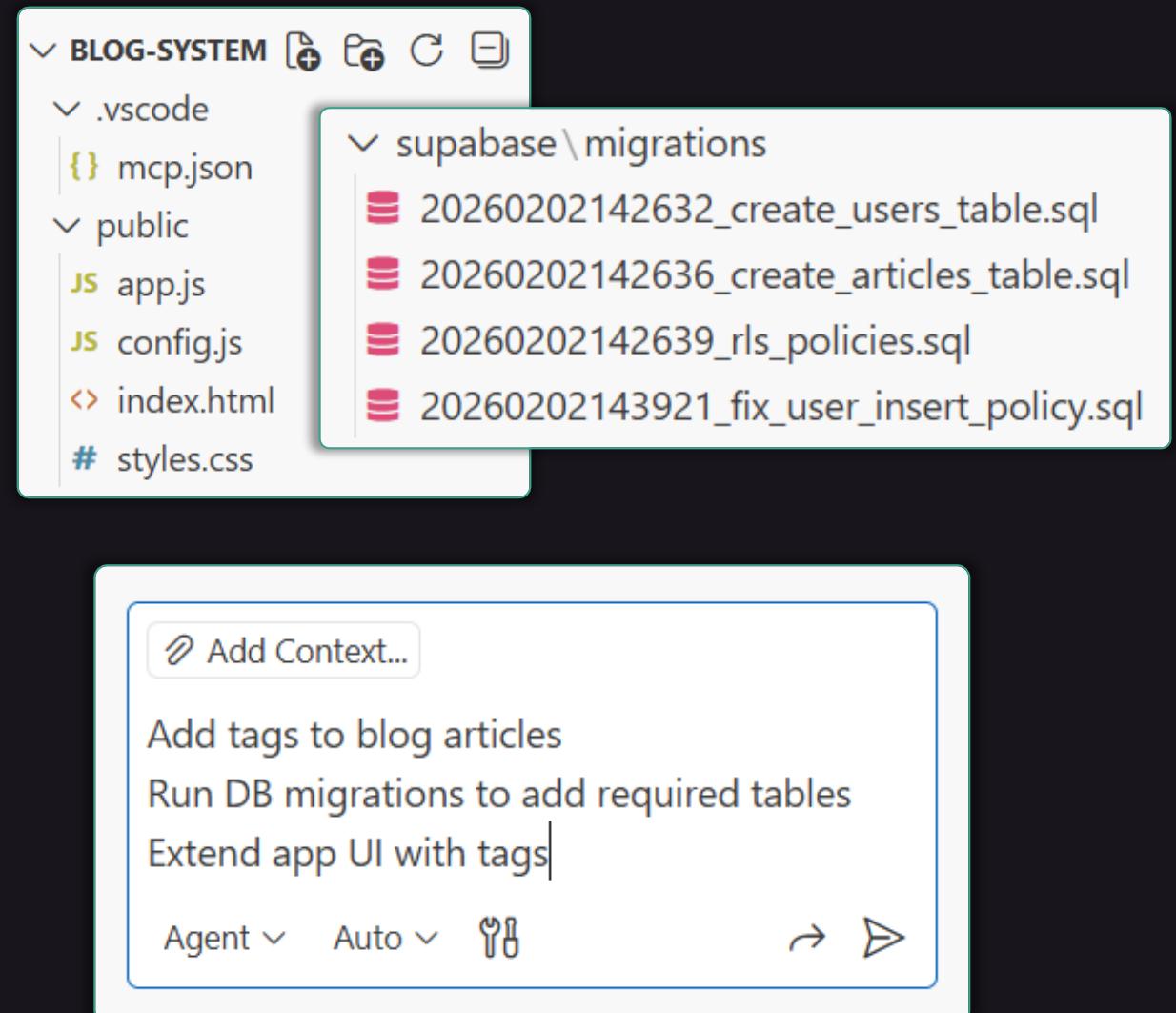
Manage Row Level Security policies for your tables

schema public

NAME	COMMAND	APPLIED TO	⋮
Anyone can read articles	SELECT	public	<input type="button" value="⋮"/>
Users can create their own articles	INSERT	public	<input type="button" value="⋮"/>
Users can delete their own articles	DELETE	public	<input type="button" value="⋮"/>
Users can update their own articles	UPDATE	public	<input type="button" value="⋮"/>

Example: Blog System Migrations

- The source code now holds:
 - **App code** (HTML, CSS, JS)
 - **DB migrations** (SQL)
- Now, let's implement **new functionality** in the blog:
 - **Add tags** to blog articles
 - This will add **DB migrations**
 - Will extend **app UI** with tags



The screenshot shows a software interface for managing database migrations. At the top, there is a file tree for a project named "BLOG-SYSTEM". The tree includes a ".vscode" folder containing an "mcp.json" file, a "public" folder containing "app.js", "config.js", and "index.html", and a "supabase\migrations" folder containing four SQL files:

- 20260202142632_create_users_table.sql
- 20260202142636_create_articles_table.sql
- 20260202142639_rls_policies.sql
- 20260202143921_fix_user_insert_policy.sql

Below the file tree, there is a large text input field with the following content:

```
Add Context...
Add tags to blog articles
Run DB migrations to add required tables
Extend app UI with tags|
```

At the bottom of the interface, there are several buttons and dropdowns:

- Agent ▾
- Auto ▾
- ↻
- ✖
-
-

Example: Blog with Tags

 **Blog**

[Logout](#) Welcome, Svetlin Nakov

Published Articles

[Publish](#)

Vibe Coding: Build a Product in One Weekend

Svetlin Nakov

February 2, 2026

[ai](#) [coding](#) [software](#)

Few days ago I was a speaker at the Crossroads 2025 startup conference in Sofia. My talk was about vibe coding a how it changes the startup landscape....

[Read More](#) [Edit](#) [Delete](#)

Debugging Techniques Every Developer Needs

Carol Williams

February 2, 2026

Debugging is an essential skill for every developer. Whether you're using browser developer tools, IDE debuggers, or logging statements, effective deb...

[Read More](#)

Blog System: RLS Policies

supabase\migrations

- 20260202142632_create_users_table.sql
- 20260202142636_create_articles_table.sql
- 20260202142639_rls_policies.sql
- 20260202143921_fix_user_insert_policy.sql
- 20260203112941_create_tags_table.sql
- 20260203112949_create_article_tags_table.sql
- 20260203112958_tags_rls_policies.sql**

contact_tags

NAME	COMMAND	APPLIED TO
Users can delete their own c...	DELETE	public
Users can insert their own c...	INSERT	public
Users can update their own ...	UPDATE	public
Users can view their own co...	SELECT	public

Disable RLS Create policy

supabase > migrations > 20260203112958_tags_rls_policies.sql

```

1  -- RLS Policies for tags table
2  -- Anyone can read tags
3  CREATE POLICY "Anyone can read tags" ON tags
4    | FOR SELECT USING (true);
5
6  -- Authenticated users can create tags
7  CREATE POLICY "Authenticated users can create tags" ON tags
8    | FOR INSERT WITH CHECK (auth.uid() IS NOT NULL);
9
10 -- No one can update or delete tags (once created, they persist)
11 -- If you want to allow updates/deletes, uncomment these:
12 -- CREATE POLICY "Authenticated users can update tags" ON tags
13 --   | FOR UPDATE USING (auth.uid() IS NOT NULL);
14 -- CREATE POLICY "Authenticated users can delete tags" ON tags
15 --   | FOR DELETE USING (auth.uid() IS NOT NULL);
16
17 -- RLS Policies for article_tags table
18 -- Anyone can read article-tag relationships
19 CREATE POLICY "Anyone can read article tags" ON article_tags
20   | FOR SELECT USING (true);
21
22 -- Users can only add tags to their own articles
23 CREATE POLICY "Users can tag their own articles" ON article_tags
24   | FOR INSERT WITH CHECK (
25     | EXISTS (
26       |   SELECT 1 FROM articles
27       |   WHERE articles.id = article_id
28       |   AND articles.owner_id = auth.uid()
29     )
30 );

```

Users and Roles

Database Apps with Users and Roles.
Visitors, Regulars Users, Admins



Users and Roles

- **Access control** in most apps works with **users** and **roles**
 - Simple apps use hard-coded **roles** (e. g. **user** / **admin**)
 - Complex apps assign **permissions** for each role (e. g. "*view payments*" permission, "*edit payments*" permission)
- Typical access control policy:
 - **Anonymous users**: can **read all public data**, e. g. articles
 - **Authenticated users**: can read / write their **own data**, e. g. own articles, own messages, own user profile
 - **Admin users**: can read / write **any data** in the DB

JWT Tokens

- JWT (JSON Web Token) is modern **authentication** and **authorization** standard
 - Holds info about the **currently logged-in user**: **user_id** / **username**, **roles** and **permissions** (optional)
 - Holds a **digital signature**, created by the token issuer
- The **JWT token** is:
 - **Issued at login** (after credentials are verified)
 - **Sent with each request** to identify and authorize the user
 - **Removed** from the client at **logout** (to forget logged-in user)

Inside JWT Tokens

Name	X	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
index.html								
styles.css								
config.js								
app.js								
supabase-js@2								
users?select=display_name&id=eq.cd7509b3-7161-40f2-bffe-e9f94144101f								
users?select=display_name&id=eq.cd750...								



header

```
{  
  "alg": "ES256",  
  "typ": "JWT"  
}
```

payload

```
{  
  "sub": "jonny86",  
  "iat": 1516239022  
}
```

signature

```
KMUFsIDTnF  
myG3nMiG  
M6H9FNFUR  
Of3wh7S...
```

jwt.io

JWT Debugger

JSON WEB TOKEN (JWT)

Valid JWT

Signature Verified

eyJhbGciOiJFUzI1NiIsImtpZCI6ImYwNDM1ZTkyLWExMTAtNDY1MC1hZjYzLTgwODIzN2VkJNjg5NyIsInR5cI6IkpxVCJ9.eyJpc3MiOiJodHRwczovL25qY2RwdmpvbXdzYnZ0cWJiZ2RrLnN1cGFiYXNlLmNvL2F1dgvdjeiLCJzdWIoiIjZC1MDIiMy03MTYxLTQzJItYmZmZS1lOWy5NDE0NDEwMWYiLCJhdWQiOioiJhdXRozW50alWNhdGVkIiwiZXhwIjoaNzcmAjAwODA1LCJpYXQiOjE3NzAxOTcyMDU5ImVtYwlsIjoic3ZldGxpzbBuYTvtvd5ijb20iLCJwaG9uZSI6IiIsImFwcF9tZXRhZGFOYStI6eyJwcm92aWRlcI6ImVtYwlsIwiwiHJvdmIkZXJiZjpbImVtVwlsIi19LCJ1c2Vx21ldgFkYXrhIjp7ImRpC3BsyxlfbmFTZSI6IlN2ZXRsaw4gTmFrB3yilCJlbWFpbCI6InN2ZXRsaw5AbmFrB3YuY29tIiwiZw1haWxfdmVyalWzPzWQiOnRydWUsInBob251X3ZlcmlmaWVkiJpmYWxzzSwic3ViIjoiY2Q3NTA5YjMtNZE2MS0MGYyLwJmzmlTzlmOTQxNDQxMDFmIn05InJvbGuioiJhdXRoZw50aWhndGVkIiwiYWFsiIjoiYWFMSMsImFtcI6W3sbw0aG9kijoicGFzc3dvcmQiLCJ0aW1c3RhbxAi0je3NzAxOTcyMDV9XSwic2Vz21vlpZCI6Jm50Dc4ZDZl1TyXOTiTNGZjzi1ZTcwLTZiNGRiM2QyYzg3ZSiSmIzX2Fub25bi91cyI6Zmfsc2V9.4907csSRvBaK6EoKSwUiij-10X3KzHEWcsm7fvPFnaEcgkhbucuAiaZ1znqXv6otjNTcu18lpv8jHW52viw

JSON CLAIMS TABLE

```
{  
  "alg": "ES256",  
  "kid": "f0435e92-a110-4650-af63-808237ed6897",  
  "typ": "JWT"  
}
```

DECODED PAYLOAD

JSON CLAIMS TABLE

```
{  
  "iss": "https://njcdpvjomwsbtqbkgdk.supabase.co/auth/v1",  
  "sub": "cd7509b3-7161-40f2-bffe-e9f94144101f",  
  "aud": "authenticated",  
  "exp": 1770200805,  
  "iat": 1516239022  
}
```

Users and Roles in Supabase



- Supabase Auth have built-in **user management** (register, login, external login, magic link login, JWT tokens, logout)
 - Supabase Auth does not provide **user roles** (e. g. admin)
 - Developers should implement **roles** by **custom code**
- Several **approaches** to define user roles in Supabase:
 - Use **auth.users.app_metadata** → simple, but can't edit roles after signup (only with edge functions)
 - Use **user_roles** table + **RLS joins** → easy and correct
 - Use **user_roles** table + **RLS** + **Auth hooks** → better performance

Users and Roles in Supabase

- Recommended approach to **implement roles** in Supabase:
 - Enum: **app_role (user | admin)**
 - Table: **user_roles (user_id, user_role)**
 - Define **RLS policies**:
 - Everyone can read **user_roles**
 - Database function **is_admin()**
 - Only admins can write **user_roles**
 - **Everyone can read public app data**
 - Only **owners** and **admins** can **edit app data**



Example: Blog with Admins

- We already have a **blog system**:
 - **Visitors** list and **view** blog articles
 - **Users** log-in, **create / edit** their **own articles**
- Now we want to **implement admins**:
 - **Admins** will have **unrestricted read / write access** to all app data
 - View / edit / delete user's articles
 - **Manage users**: view / create / edit / delete users



Implementing Blog Admins in DB

- Step 1: Define **users** and **roles** in the Supabase DB

Implement **user roles** in Supabase:

- Enum: **app_role** (**user** | **admin**)
- Table: **user_roles** (**user_id**, **user_role**)

user_id	user_role
311ab519-1500-4023-b9...	admin
cd7509b3-7161-40f2-bff...	admin

Define **RLS policies**:

- Everyone can **read user_roles**
- Database function **is_admin()**
- Only **admins** can **write user_roles**
- **Everyone** can **read** public app data
- Only **owners** and **admins** can **edit** app data



Implementing Admin Panel



- **Step 2:** Implement the Admin Panel

Implement the app **Admin Panel**:

- Use separate files: **admin.html**, **admin.css**, **admin.js**
- The admin panel should be available **only for admins**
 - Admins should have **[Admin]** button in the app header
 - Two tabs: **Article Admin** | **User Admin**
 - **Article Admin** tab: view all articles in a table with **[Edit]** and **[Delete]** buttons, implemented with popups
 - **User Admin** tab: view all users, implement **[Make Admin]** / **[Remove Admin]** buttons

The Admin Panel in Action

- First, **create an admin** from Supabase Studio
- Next, **login** with the admin user
- Open the **Admin Panel**

Admin Panel

Back to Blog Logout Admin (Admin)

Article Admin User Admin

All Articles Refresh

Title	Author	Created	Tags	Actions
From Steve Nak	Steve Nak	Feb 4, 2026	steve nak	Edit Delete
Vibe Coding: Build a Product in One Weekend	Svetlin Nakov	Feb 2, 2026	ai coding software	Edit Delete
Debugging Techniques Every Developer Needs	Carol Williams	Feb 2, 2026	No tags	Edit Delete

Blog Admin Logout Welcome, Admin

Admin Panel

Back to Blog Logout Admin (Admin)

Article Admin User Admin

All Users Refresh

Display Name	Email	Role	Joined	Actions
Steve Nak	steve@abv.bg	User	Feb 4, 2026	Make Admin
Admin	admin@admin.com	ADMIN	Feb 4, 2026 (You)	
Svetlin Nakov	svetlin@nakov.com	ADMIN	Feb 2, 2026	Remove Admin

Add new row to user_roles

user_id

311ab519-1500-4023-b902-d22b19c19299



uuid

Has a foreign key relation to public.users.id

user_role

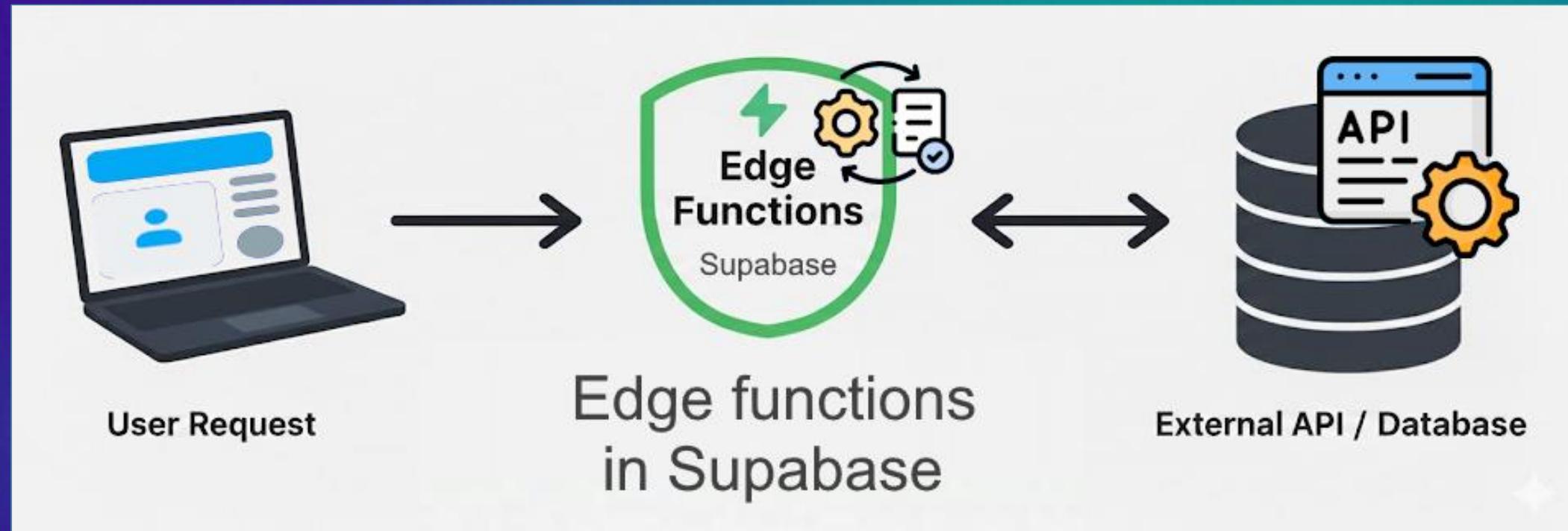
admin



app_role

Edge Functions

JS Logic in the Supabase Backend



Supabase Edge Functions

- **Supabase Edge Functions**
 - **Serverless functions**, acting as a lightweight back-end
 - Implemented in **JavaScript**, accessed through the API
 - Implement **payments**, **emails**, **webhooks** (Web callbacks)
 - Have **unrestricted DB access**, skipping the RLS policies
- **When** to use edge functions?
 - DB operations with **elevated privileges**: e. g. delete user
 - Access **external APIs**, e. g. send email, accept a payment, integrate an LLM (like ChatGPT or Gemini)



Serverless Example

- We already have a **blog system** with **admin panel**
 - **Visitors** read / view blog articles
 - **Users** log-in and **create** articles, **edit** / **delete** own articles
 - **Admins** have **admin panel**: edit articles and manage users
- Now we want to **implement full user management**
 - **Create** / **edit** / **delete** users
 - This requires **elevated privileges** for the DB
 - Can be implemented with **edge functions**

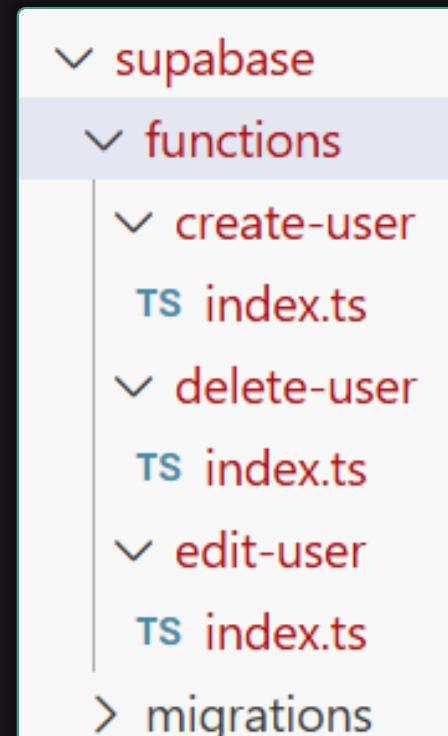
Manage Users with Edge Functions



- Implement **full user management** with **edge functions**:

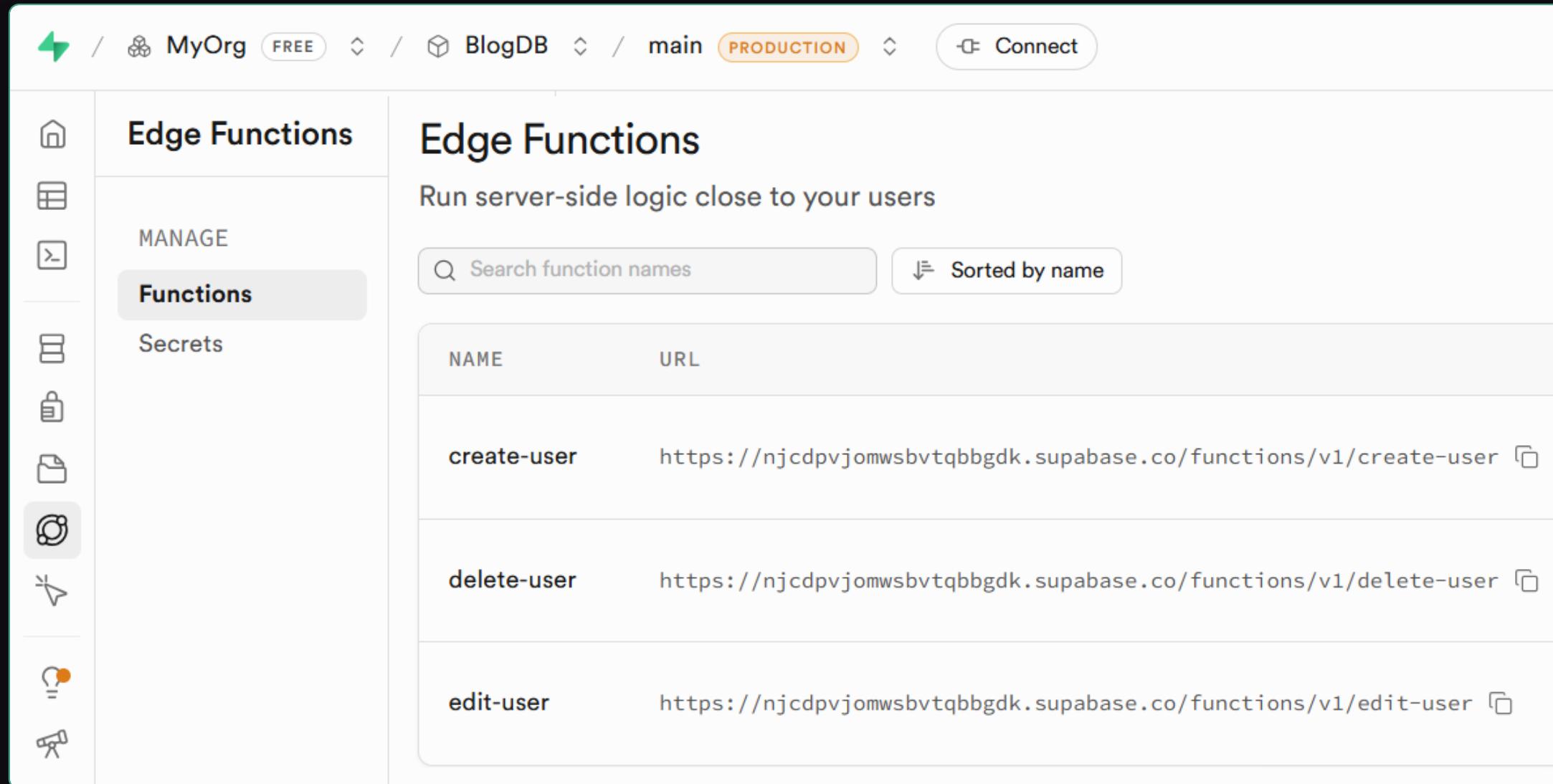
Implement user management in the **Admin Panel**:

- In the **User Admin** table add buttons [**Add**] / [**Edit**] / [**Delete**]
- Implement "Create user", "Edit user", "Delete user" with **edge functions** in the Supabase backend
- Use **popup dialogs** for add / edit / delete confirm
- Choose **user role** when creating / editing users



Deploy the edge functions to Supabase

Deployed Edge Functions



The screenshot shows the Supabase Edge Functions dashboard. The left sidebar has icons for Home, Tables, Functions (which is selected), Secrets, and other management tools. The main area title is "Edge Functions" with the subtitle "Run server-side logic close to your users". It includes a search bar and a sorting dropdown. A table lists three functions:

NAME	URL
create-user	https://njcdpvjomwsbvtqbbgdk.supabase.co/functions/v1/create-user 🔗
delete-user	https://njcdpvjomwsbvtqbbgdk.supabase.co/functions/v1/delete-user 🔗
edit-user	https://njcdpvjomwsbvtqbbgdk.supabase.co/functions/v1/edit-user 🔗

CORS Problems

- Your app may encounter the following problem:

✖ Access to fetch at '[admin.html:1 https://njcdpvjomwsbvtqbbgdk.supabase.co/functions/v1/create-user](https://njcdpvjomwsbvtqbbgdk.supabase.co/functions/v1/create-user)' from origin 'null' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

✖ Failed to load resource: <njcdpvjomwsbvtqbbgdk...ns/v1/create-user:1>  

✖ ► Error creating user: TypeError: Failed to fetch at handleAddUser (<admin.js:454:28>) <admin.js:479>

- The edge functions **do not handle CORS** correctly → ask the AI agent to fix and re-deploy them

Admin Manage Users in Action

 Admin Panel

[Back to Blog](#) [Logout](#) Admin (Admin)

Article Admin User Admin

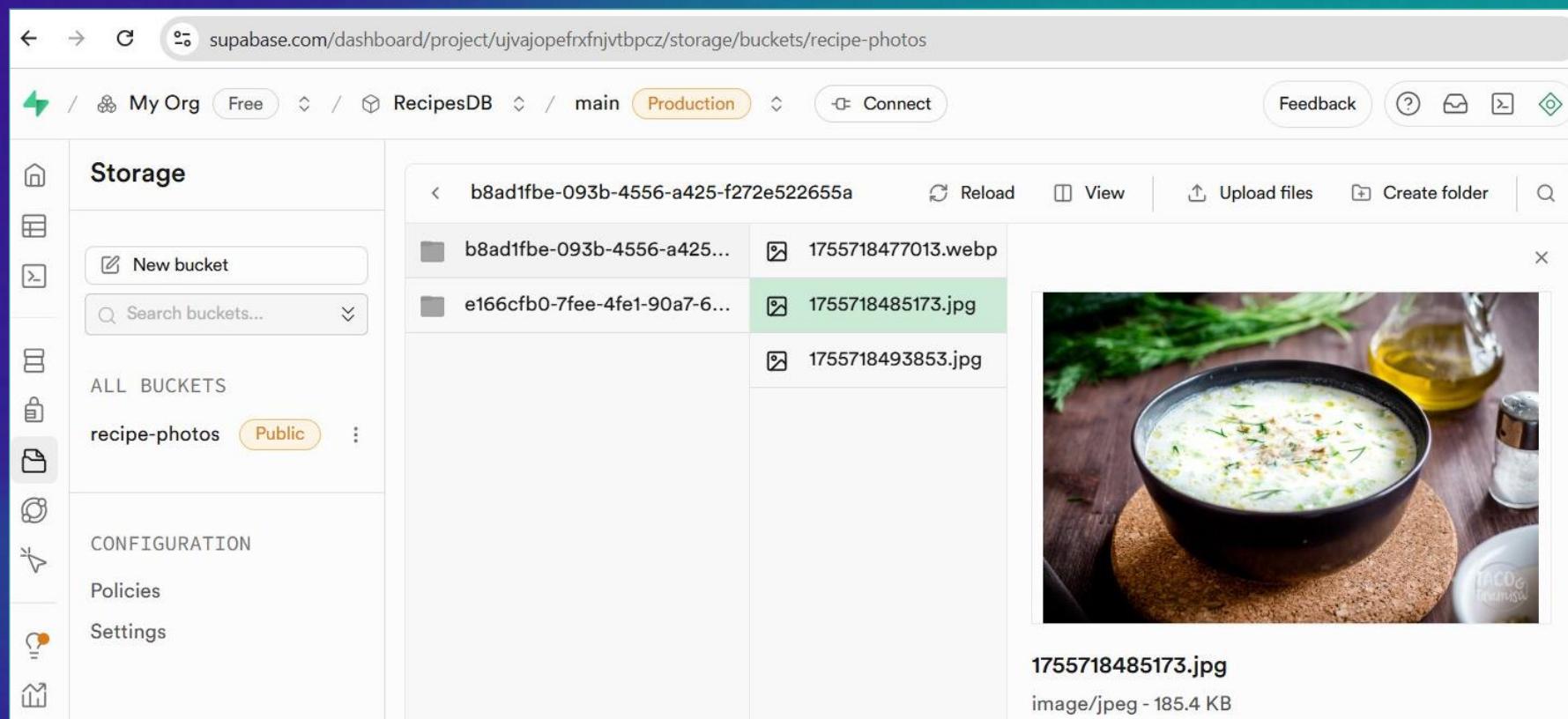
All Users

[Add User](#) [Refresh](#)

DISPLAY NAME	EMAIL	ROLE	JOINED	ACTIONS
Bay Ivan	ivan@abv.bg	ADMIN	Feb 4, 2026	Edit Delete
Steve Nak	steve@abv.bg	USER	Feb 4, 2026	Edit Delete

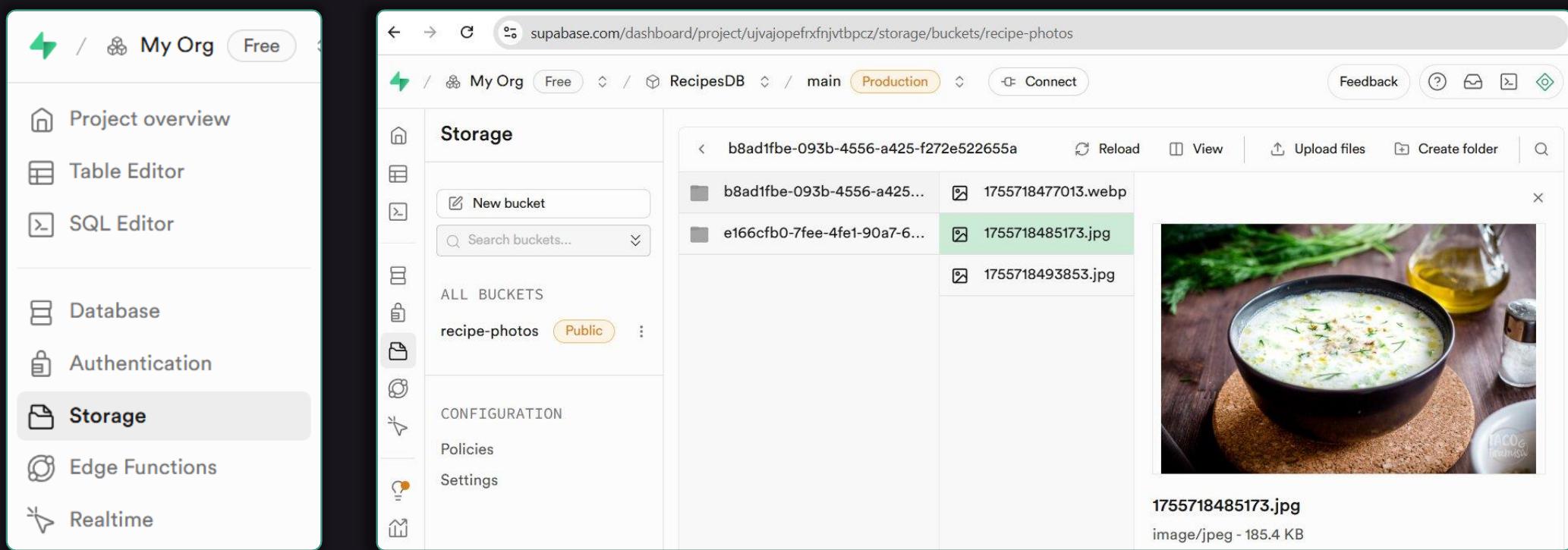
Supabase Storage

Upload and Download Images and Files



Photos and File Uploads

- Apps typically store **photos, images, documents, files** in the **back-end** or in a **cloud storage**
- In Supabase images are stored in **Supabase Storage**



Supabase Storage

- Supabase Storage implements server-side file uploads, storage and downloads
 - Store images, photos, documents, videos, etc.
 - Storage structure:
 - Buckets (root folders, which hold files)
 - Files (with file paths, imitating folders)
 - Functionality: file upload, download, share
 - Access control: tables with RLS in the `storage` schema

Implementing Blog Images



- Implement **blog images** with **Supabase Storage**
 - We already have the **blog system**, which allows users to view and publish blog article
 - Now we shall **implement cover image** for the articles

Implement **cover images** for the blog articles

- Implement **image upload** in Add / Edit articles
- Keep image uploads in the **Supabase Storage**
- Implement proper **RLS policies**: everyone can view images, but only image owners can edit / delete them

Blog with Images in Action

Blog

Published Articles



Tarator - the full recipe

Admin

February 4, 2026

[tarator](#) [recipe](#) [food](#)

On days when the thermometers are showing nearly 40 degrees, everyone is looking for a way to escape the heat. Of course, the best solution to the pro...

[Read More](#)

Admin Logout Welcome, Svetlin Nakov

MyOrg FREE / BlogDB main PRODUCTION Connect Feedback

Storage

Files > Buckets > article-images PUBLIC

MANAGE Files Analytics Vectors

311ab519-1500-4023-b902-d22b19c19299
800b6c01-cddd-48b9-847...
9c8bc ecc-eca1-46a9-8c94...
1770216403718_tarator4.jpeg

Reload View

Moussaka recipe

Steve Nak February 4, 2026

[food](#) [moussaka](#) [recipe](#) [bulgarian](#)

Ingredients Original recipe (1X) yields 12 servings ¾ pound ground beef (85% lean) ¾ pound ground pork ½ cup olive oil, divided 1 large carrot, f...

[Read More](#)

```
async function publishArticle(e) {
  if (imageFile) {
    const timestamp = Date.now();
    const filename = `${articleData.id}/${currentUser.id}/${
      timestamp
    }.jpeg`;
    // Upload file to storage
    const { error: uploadError } = await supabaseClient
      .storage
      .from('article-images')
      .upload(filename, imageFile);

    if (uploadError) throw uploadError;

    // Get public URL
    const { data: urlData } = supabaseClient
      .storage
      .from('article-images')
      .getPublicUrl(filename);
  }
}
```

Lesson Summary

- **Supabase Auth** has built-in authentication system: users, login / logout, JWT tokens, RLS security, OAuth
- **Database migrations**: SQL scripts for DB schema changes, ordered by time, tracked in both DB and local files
- **Row-Level Security (RLS)**: defining policies for row-level access control (for SELECT, INSERT, UPDATE, DELETE)
- Implementing app **users** and **roles** (visitors, users, admins) in Supabase with **user_roles** table + **RLS joins**
- **Supabase Storage**: implementing file uploads / downloads

Questions?

Postbank – Exclusive Partner for SoftUni AI



- One of the leading **banking institutions** in Bulgaria
- Member of the Eurobank Group with € 103 billion assets
- Innovative trendsetter with next generation **beyond banking**, transforming today, empowering tomorrow
- Certified **Top Employer 2025** by the international Top Employers Institute
- Proven people care and **wellbeing initiatives**
- Benefits and unlimited access to professional, **personal and leadership trainings and programs**
- www.postbank.bg / careers.postbank.bg



Diamond Partners of Software University



Diamond Partners of SoftUni Digital



**SUPER
HOSTING
.BG**



Diamond Partners of SoftUni Digital



HUMAN

NETPEAK
DIGITAL GROWTH PARTNER



Marmalab | Е-комерс агенция

ETIEN YANEV
Break Your *Limits*. Live Your *Brand*.

1FORFIT



Diamond Partners of SoftUni Creative



Organization Partners of SoftUni Creative

