

## My Github Repo URL

# W11-P1: Supabase Database setting

The screenshot shows the Supabase Settings interface. On the left, there's a sidebar with various project settings like General, Database, API, Auth, Storage, Billing, Subscription, Usage, and Invoices. The Database section is selected and highlighted with a red box. The main area is titled "Database Settings" and contains fields for "Connection info": Host (db.jiewhttktusvivcyqnki.supabase.co), Database name (postgres), Port (5432), User (postgres), and Password ([The password you provided when you created this project]). Below this, there's a "Database password" section with a "Reset Database Password" button. At the bottom, there's a "Connection string" section with tabs for PSQL, URI, Golang, JDBC, .NET, Nodejs, PHP, and Python.

# W11-P2: Connecting Supabase

The screenshot shows a Visual Studio Code interface with several files open in the Explorer sidebar: database.js, app.js, .env, w02-p6.png, w03, w06, w11, w01, w02, w06, w11, node\_modules, public, routes, utils, data, database.js, test-db.js, test.js, test.json, views, .env, .gitignore, category\_28.tar, crown2\_28.tar, notes.txt, package-lock.json, package.json, and README.md. The app.js file is the active tab, showing code for creating an Express app, setting up middleware, and defining routers for index, users, and crowns. A red box highlights the line `const dotenv = require('dotenv'); dotenv.config();`. The database.js file shows code for connecting to a PostgreSQL database using the pg pool library. A red box highlights the conditional logic for connecting to Supabase or local PostgreSQL based on the DATABASE environment variable. The terminal at the bottom shows the command `npm run dev` being run, and the output indicates that the application is connecting to a PostgreSQL database running on port 3000.

# W11-P3: Put md img file into Supabase

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a project structure with files like app.js, .env, .gitignore, and w11\_28.md.
- Terminal:** Displays the command \$ git log --pretty=format:"%h%x09%an%x09%ad%x09%" --after="2023-4-25".
- Code Editor:** The file w11\_28.md contains several code snippets and URLs. Two specific URLs are highlighted with red boxes:
  - Line 5: [\[w11-p1.png\]\(https://jiewhttktusivvcygnki.supabase.co/storage/v1/object/public/demo\\_28/28\\_img/w11-p1.png\)](https://jiewhttktusivvcygnki.supabase.co/storage/v1/object/public/demo_28/28_img/w11-p1.png)
  - Line 9: [\[w11-p2.png\]\(https://jiewhttktusivvcygnki.supabase.co/storage/v1/object/public/demo\\_28/28\\_img/w11-p2.png?t=2023-04-26T06%3A34%3A51.263Z\)](https://jiewhttktusivvcygnki.supabase.co/storage/v1/object/public/demo_28/28_img/w11-p2.png?t=2023-04-26T06%3A34%3A51.263Z)
- Supabase Database Settings:** A separate window titled "W11-P1: Supabase Database setting" shows the database configuration. It includes fields for Host (db.supabasehq.com), Database name (postgres), Port (5432), User (postgres), and Password (the password you provided when you created this project). A "Database password" field is also present.
- Terminal:** Shows the output of the command \$ node app.js, which includes logs about connecting to PostgreSQL and creating tables.

# W11-P4: Connect Supabase from local pgAdmin, restore midprep\_xx.tar

The screenshot shows the pgAdmin interface. The left sidebar displays the database structure under the 'public' schema, with the 'product\_28' table highlighted by a red box. The main area shows a query editor with the following SQL:

```
1 SELECT * FROM public.product_28
2 ORDER BY pid ASC
```

Below the query editor is a data grid titled 'Data Output' showing the contents of the 'product\_28' table:

pid	pname	cat_id	price	img_url
[PK] integer	character varying (255)	integer	real	character varying (255)
1	Blue Tanktop	4	25	/img/womens/blue-tank.png
2	Floral Blouse	4	20	/img/womens/floral-blouse.png
3	Floral Dress	4	80	/img/womens/floral-skirt.png
4	Red Dots Dress	4	80	/img/womens/red-polka-dot-dress.png
5	Brown Brim	1	25	/img/hats/brown-brim.png
6	Blue Beanie	1	18	/img/hats/blue-beanie.png
7	Brown Cowboy	1	35	/img/hats/brown-cowboy.png
8	Grey Brim	1	25	/img/hats/grey-brim.png

The screenshot shows the Supabase Table editor. The left sidebar lists the 'Tables (3)' section, with the 'product\_28' table highlighted by a red box. The main area shows the table data:

pid	pname	cat_id	price	img_url
1	Blue Tanktop	4	25	/img/womens/blue-tank.png
2	Floral Blouse	4	20	/img/womens/floral-blouse.png
3	Floral Dress	4	80	/img/womens/floral-skirt.png
4	Red Dots Dress	4	80	/img/womens/red-polka-dot-dress.png
5	Brown Brim	1	25	/img/hats/brown-brim.png
6	Blue Beanie	1	18	/img/hats/blue-beanie.png
7	Brown Cowboy	1	35	/img/hats/brown-cowboy.png
8	Grey Brim	1	25	/img/hats/grey-brim.png

## W12-P1: Create foreign key from cat\_id (shop2\_xx ) to id (category2\_xx)

pgAdmin 4

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays the database structure. A red box highlights the 'Constraints' section under the shop2\_28 table, which contains two entries: fkey1 and shop\_xx\_pkey. The main window shows the 'shop2\_28' table configuration with the 'Foreign Key' tab selected. A red box highlights the first row of the 'Foreign Key' table, which defines the constraint 'fkey1' with columns '(cat\_id) -> (id)' and a referenced table 'public.category2\_28'. The bottom right corner of the main window shows the status bar with 'Adidas NMD'.

## W12-P2: give SQL to get products based on a category

pgAdmin 4

The screenshot shows the pgAdmin 4 interface. The object browser on the left highlights the 'category2\_28' and 'shop2\_28' tables, both enclosed in red boxes. The main window shows a query editor with the following SQL code:

```

1 select C.name as category, S.id, S.name, price, S.local_url
2 from category2_28 as C, shop2_28 as S
3 where S.cat_id = C.id and C.name = 'sneakers'
4

```

The results of the query are displayed in a table below, also enclosed in a red box. The table has columns: category, id, name, price, and local\_url. The data shows 8 rows of sneaker products.

category	id	name	price	local_url
sneakers	20	Adidas NMD	220	/img/sneakers/adidas-nmd.png
sneakers	21	Adidas Yeezy	280	/img/sneakers/yeezy.png
sneakers	22	Black Converse	110	/img/sneakers/black-converse.png
sneakers	23	Nike White AirForce	160	/img/sneakers/white-nike-high-tops.png
sneakers	24	Nike Red High Tops	160	/img/sneakers/nikes-red.png
sneakers	25	Nike Brown High Tops	160	/img/sneakers/nike-brown.png
sneakers	26	Air Jordan Limited	190	/img/sneakers/nike-funky.png
sneakers	27	Timberlands	200	/img/sneakers/timberlands.png

## W12-P3: implement route /crown2\_xx/shop2\_xx/:category, and show json data regarding some category in Browser

The screenshot shows the Visual Studio Code interface with several files open. The current file is `crown2_28.js`, which contains a route handler for `/shop2_28/:category`. The code uses an asynchronous function to query a database for products in a specific category and then renders a template with the results. The browser window on the right shows the JSON response for the 'sneakers' category.

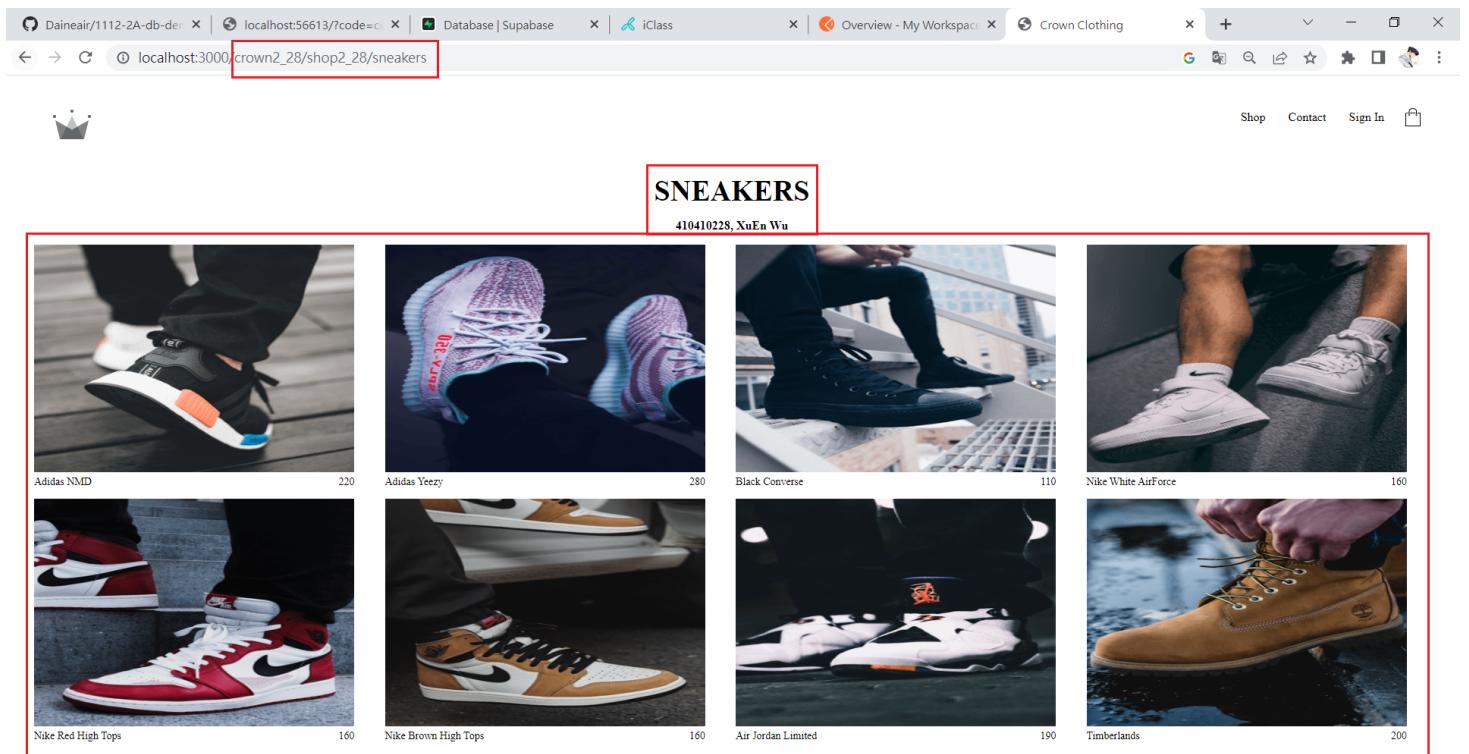
```
routes > JS crown2_28.js > router.get('/shop2_28/:category') callback
18
19
20
21
22 router.get('/shop2_28/:category', async function (req, res, next){
23     console.log('category', req.params.category);
24     try {
25         let results = await db.query(`
26             select C.name as category, S.id, S.name, price, S.local_url
27             from category2_28 as C, shop2_28 as S
28             where S.cat_id = C.id and C.name = $1
29         `, [req.params.category]);
30         console.log(results, JSON.stringify(results.rows));
31         res.json(results.rows);
32     } catch(error) {
33         console.log(error);
34     }
35 });
36
37 */
38
39
40
41 });

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Web server running on port 3000
category sneakers
results [{"category": "sneakers", "id": 20, "name": "Adidas NMD", "price": 220, "local_url": "/img/sneakers/adidas-nmd.png"}, {"category": "sneakers", "id": 21, "name": "Adidas Yeezy", "price": 280, "local_url": "/img/sneakers/yeezy.png"}, {"category": "sneakers", "id": 22, "name": "Black Converse", "price": 110, "local_url": "/img/sneakers/black-convers-e.png"}, {"category": "sneakers", "id": 23, "name": "Nike White AirForce", "price": 160, "local_url": "/img/sneakers/nike-white-airforce.png"}, {"category": "sneakers", "id": 24, "name": "Nike Red High Tops", "price": 160, "local_url": "/img/sneakers/nikes-red.png"}, {"category": "sneakers", "id": 25, "name": "Nike Brown High Tops", "price": 190, "local_url": "/img/sneakers/nike-brown.png"}, {"category": "sneakers", "id": 26, "name": "Air Jordan Limited", "price": 190, "local_url": "/img/sneakers/nike-funkyy.png"}, {"category": "sneakers", "id": 27, "name": "Timberlands", "price": 200, "local_url": "/img/sneakers/timberlands.png"}]
GET /crown2_28/shop2_28/sneakers 304 1026.658 ms - -
Spaces: 4 UTF-8 CRLF ↵ Go Live iCode ready ✓ Spell ✓ Prettier ⚡
```

The browser window displays the JSON data for the 'sneakers' category. It lists various sneaker models with their IDs, names, prices, and local URLs. A red box highlights the URL in the address bar: `localhost:3000/crown2_28/shop2_28/sneakers`.

```
// 20230503145503
// http://localhost:3000/crown2_28/shop2_28/sneakers
[
    {
        "category": "sneakers",
        "id": 20,
        "name": "Adidas NMD",
        "price": 220,
        "local_url": "/img/sneakers/adidas-nmd.png"
    },
    {
        "category": "sneakers",
        "id": 21,
        "name": "Adidas Yeezy",
        "price": 280,
        "local_url": "/img/sneakers/yeezy.png"
    },
    {
        "category": "sneakers",
        "id": 22,
        "name": "Black Converse",
        "price": 110,
        "local_url": "/img/sneakers/black-convers-e.png"
    },
    {
        "category": "sneakers",
        "id": 23,
        "name": "Nike White AirForce",
        "price": 160,
        "local_url": "/img/sneakers/nike-white-airforce.png"
    },
    {
        "category": "sneakers",
        "id": 24,
        "name": "Nike Red High Tops",
        "price": 160,
        "local_url": "/img/sneakers/nikes-red.png"
    },
    {
        "category": "sneakers",
        "id": 25,
        "name": "Nike Brown High Tops",
        "price": 190,
        "local_url": "/img/sneakers/nike-brown.png"
    },
    {
        "category": "sneakers",
        "id": 26,
        "name": "Air Jordan Limited",
        "price": 190,
        "local_url": "/img/sneakers/nike-funkyy.png"
    },
    {
        "category": "sneakers",
        "id": 27,
        "name": "Timberlands",
        "price": 200,
        "local_url": "/img/sneakers/timberlands.png"
    }
]
```

## W12-P4: implement route /crown2\_xx/shop2\_xx/:category, and show in shop2\_xx.ejs



Screenshot of Visual Studio Code showing the implementation of route /crown2\_xx/shop2\_xx.

The code in `crown2_28.js` (left) contains logic to handle the request for `/shop2_28/:category`. It uses `db.query` to select products from `shop2_28` by category and renders the results to the client.

The code in `shop2.ejs` (right) defines the view for displaying the products. It includes a header with title and category information, a main section with product cards (each showing an image, name, and price), and a footer with a "Add to Cart" button.

```
JS crown2_28.js M views shop2.ejs U
routes > JS crown2_28.js > router.get('/shop2_28/:category') callback > title
17     }catch(error){
18         console.log(error);
19     }
20 }
21 }
22 }
23 });
24
25 router.get('/shop2_28/:category', async function (req, res, next){
26     console.log('category', req.params.category);
27     try {
28         let results = await db.query(`SELECT C.name AS category, S.id, S.name, price, S.local_url
29             FROM category_28 AS C, shop2_28 AS S
30             WHERE S.cat_id = C.id AND C.name = '$1'
31             `, [req.params.category]);
32         console.log('results', JSON.stringify(results.rows));
33         //res.json(results.rows);
34         res.render('crown2_28/shop2_28',{
35             data:results.rows,
36             category: req.params.category,
37             title: 'XuEn Wu',
38             ID:'410410228',
39         });
40     }catch(error){
41         console.log(error);
42     }
43 }
44 //UPDATE
45
46 //DELETE
47
48 
```

```
<div class="shop-page">
<div class="collection-page">
<h1 class="title"><%= category.toUpperCase() %></h1>
<h2 style="text-align: center; padding-bottom: 1rem"><%= ID %>, <%= title %></h2>
<div class="items">
<% for (let i=0; i<%= data[i].name %></span>
<span class="price"><%= data[i].price %></span>
</div>
<button class="custom-button">Add to Cart</button>
</div>
<% } %>
</div>
</div>
<script>
function cartToggle(){
    p = document.querySelector('.cart-dropdown');
    p.classList.toggle("show");
}
</script>
```

## W12-P5: implement route /crown2\_xx/shop2\_xx to show all products in shop2\_xx.ejs

Screenshot of Visual Studio Code showing the implementation of route /crown2\_xx/shop2\_xx.

The code in `crown2_28.js` (left) contains logic to handle the request for `/shop2_28`. It uses `db.query` to select all products from `shop2_28` and renders the results to the client.

The browser screenshot (right) shows the resulting page titled "ALL PRODUCTS". It displays a grid of product cards, each showing an image, name, and ID. The products include various hats, jackets, and shirts.

```
JS crown2_28.js M w12_28.md M
routes > JS crown2_28.js > router.get('/shop2_28', async function (req, res, next){
21     try {
22         let results = await db.query('SELECT * FROM shop2_28');
23         console.log('results', JSON.stringify(results.rows));
24         //res.json(results.rows);
25         res.render('crown2_28/shop2_28',{
26             data:results.rows,
27             category: "All Products",
28             title: 'XuEn Wu',
29             ID:'410410228',
30         });
31     }catch(error){
32         console.log(error);
33     }
34 });
35
36 router.get('/shop2_28/:category', async function (req, res, next){
37     try {
38         let results = await db.query(`SELECT C.name AS category, S.id, S.name, price, S.local_url
39             FROM category_28 AS C, shop2_28 AS S
40             WHERE S.cat_id = C.id AND C.name = '$1'
41             `, [req.params.category]);
42         console.log('results', JSON.stringify(results.rows));
43         //res.json(results.rows);
44         res.render('crown2_28/shop2_28',{
45             data:results.rows,
46             category: req.params.category,
47             title: 'XuEn Wu',
48             ID:'410410228',
49         });
50     }catch(error){
51         console.log(error);
52     }
53 });
54
55 
```

Browser Screenshot:

ALL PRODUCTS  
410410228, XuEn Wu

The browser window displays a grid of 16 product cards. Each card includes an image, a name, and a small number. The products are categorized under "All Products". Some visible items include "Brown Beanie", "Blue Beanie", "Brown Cowboy", "Grey Beanie", "Green Beanie", "Pink Beanie", "Red Beanie", "Yellow Beanie", "Blue Snapback", "Black Jean Shading", "Blue Jean Jacket", "Grey Jean Jacket", "Brown Jacket", "Blue Jacket", and "Pink Sneakers".

# W12-P6: make each category link works in /crown2\_xx page

The screenshot shows a dual-pane interface. On the left is Visual Studio Code with an open file named index.ejs. The code contains a section for a homepage menu:

```
<div class="homepage">
  <h1>Crown2_28 from DB: <%= title %>, <%= ID %></h1>
  <div class="directory-menu">
    <% for(let i=0;i<data.length; i++){ %>
      <div class="<%= data[i].size %> menu-item">
        
        <a href="<%= data[i].link_url %>">
          <span class="content">
            <h1 class="title"><%= data[i].name.toUpperCase() %</h1>
            <span class="subtitle">SHOP NOW</span>
          </span>
        </a>
      </div>
    <% } %>
  </div>
</div>
```

On the right is a browser window showing a Supabase Table editor for a 'category2\_xx' table. The table has columns 'id' (int4) and 'link\_url' (varchar). The data is as follows:

	id	link_url
1		/crown2_28/shop2_28/hats
2		/crown2_28/shop2_28/jackets
3		/crown2_28/shop2_28/sneakers
4		/crown2_28/shop2_28/womens
5		/crown2_28/shop2_28/mens

## Github logs of Week 12

The terminal window shows the following git log output:

```
User@E201-42 MINGW64 /d/2A/1112-2A-db-demo-410410228 (main)
$ git log --pretty=format:"%h%x09%an%x09%ad%x09%s" --after="2023-4-25"
e3d87933      Daineair       Thu May 4 17:10:06 2023 +0800   w12 0504 supabase
d598c31d      Daineair       Wed May 3 15:01:46 2023 +0800   w12 0503
8f1e78d3      Daineair1~    Wed Apr 26 15:03:31 2023 +0800   w11 0426
```

```
$ git log --pretty=format:"%h%x09%an%x09%ad%x09%s" --after="2023-4-25"
e3d87933      Daineair       Thu May 4 17:10:06 2023 +0800   w12 0504 supabase
d598c31d      Daineair       Wed May 3 15:01:46 2023 +0800   w12 0503
8f1e78d3      Daineair1~    Wed Apr 26 15:03:31 2023 +0800   w11 0426
```