

[Github URL](#)

[Github URL for Vercel](#)

[Vercel URL](#)

## W11-P1: Implement route /api/shop\_xx/:category in server

=> create tables category2\_xx and shop2\_xx using SQL and set foreign key of cat\_id referencing to cid in category2\_xx

The screenshot shows the pgAdmin 4 interface with the following details:

- Object Explorer:** Shows the database structure. A red box highlights the table `shop2_11` under the `category2_11` schema.
- Query Editor:** A red box highlights the connection bar at the top. The query is:

```
1 SELECT * FROM public.shop2_11
2 ORDER BY pid ASC
```
- Data Output:** Displays the results of the query, showing 35 rows of sneaker data. A red box highlights the first few rows.
- Create - Foreign key:** A modal window is open, showing the configuration for creating a foreign key. It has tabs for General, Definition, Columns, Action, and SQL. The **Columns** tab is selected. A red box highlights the "Referencing" section where the local column `cat_id` is mapped to the referenced column `cid` in the table `public.category2_11`.
- System Bar:** Shows the system tray with icons for weather (27°C), battery, and network.
- Taskbar:** Shows various application icons including File Explorer, Task Manager, and browser icons.

## => update and delete constraints of the foreign key

The screenshot shows the pgAdmin 4 interface for managing PostgreSQL objects. In the Object Explorer, under the 'Tables' section, the 'category2\_11' table is selected. The 'Constraints' section is expanded, showing a single constraint named 'shop2\_11\_pkey'. A modal window titled 'Create - Foreign key' is open, specifically the 'Definition' tab. The 'On update' field is set to 'CASCADE' and the 'On delete' field is set to 'SET NULL'. These settings are highlighted with a red box.

pid	pname	cat_id	price	img_url	remote_img_url
1	Brown Brim	1	25	/image/brown_brim.jpg	
2	Blue Beanie	1	18	/image/blue_beanie.jpg	
3	Brown Cowboy	1	35	/image/brown_cowboy.jpg	
4	Grey Brim	1	25	/image/grey_brim.jpg	
5	Green Beanie	1	18	/image/green_beanie.jpg	
6	Palm Tree Cap	1	14	/image/palm_tree_cap.jpg	
7	Red Beanie	1	18	/image/red_beanie.jpg	
8	Wolf Cap	1	14	/image/wolf_cap.jpg	
9	Blue Snapback	1	16	/image/blue_snapback.jpg	
10	Black Jean Shearling	2	125	/image/black_jean_shearling.jpg	
11	Blue Jean Jacket	2	90	/image/blue_jean_jacket.jpg	
12	Grey Jean Jacket	2	90	/image/grey_jean_jacket.jpg	
13	Brown Shearling	2	165	/image/brown_shearling.jpg	
14	Tan Trench	2	185	/image/tan_trench.jpg	
15	Adidas NMD	3	220	/image/adidas_nmd.jpg	
16	Adidas Yeezy	3	280	/image/adidas_yeezy.jpg	
17	Black Converse	3	110	/image/black_converse.jpg	
18	Nike White AirForce	3	160	/images/midterm/sneakers/white-nike-high-tops... https://libb.co/1RcFPk0/white-nike-hig...	
19	Nike Red High Tops	3	160	/images/midterm/sneakers/nikes-red.png https://libb.co/QcvzydB/nikes-red.png	
20	Nike Brown High Tops	3	160	/images/midterm/sneakers/nike-brown.png https://libb.co/fMTV342/nike-brown.pn	

=> Chrome, show /api/shop\_xx/womens with code (you need to use last two digits of your id)

The screenshot shows a Windows desktop environment. On the left, a browser window displays the URL `localhost:5000/api/shop_xx/womens`. The page content is a JSON object representing a product item, with a red box highlighting the entire response body. On the right, a code editor window titled `w11_next_fullstack_11` shows the `server.js` file. A red box highlights the section of code that handles the `/api/shop_xx/:category` route. The terminal below shows the server starting and listening on port 5000. The taskbar at the bottom includes icons for search, file explorer, and various applications.

```
server_11 > JS server.js > app.use('/api/shop_xx') callback
1  import express from 'express';
2  import cors from 'cors';
3
4  const app = express();
5
6  import db from './utils/database.js';
7
8  app.use(cors());
9
10 < app.use('/api/blog_xx', async (req, res, next) => {
11   const results = await db.query(`select * from blog_xx`);
12   console.log('results', JSON.stringify(results.rows));
13   res.json(results.rows);
14 });
15 < app.use('/api/shop_xx/:category', async (req, res, next) => {
16   console.log('category', req.params.category);
17   const results = await db.query(
18     `select * from category2_11, shop2_11 where cname = $1 and cid = cat_id`, [
19       // console.log('results', JSON.stringify(results.rows));
20       res.json(results.rows);
21     ]);
22 });
23 < app.use('/api/shop_xx', async (req, res, next) => {
24   const results = await db.query(`select * from shop2_11`);
25   console.log('results', JSON.stringify(results.rows));
26   res.json(results.rows);
27 });
28
29
30 < app.use('/', (req, res, next) => {
31   res.send('ChenYiHao, 213410011');
```

=> Chrome, show /api/shop\_xx/mens with code (you need to use last two digits of your id)

The screenshot shows a Windows desktop environment. On the left, a Microsoft Edge browser window is open, displaying a JSON response from a server at localhost:5000/api/shop\_xx/mens. The JSON data includes product details like cid, cname, size, image\_url, remote\_image\_url, link\_url, pid, pname, cat\_id, price, img\_url, and remote\_img\_url. A red box highlights the URL in the browser's address bar. On the right, a Visual Studio Code (VS Code) interface is visible, showing the file server.js. The code implements various routes including '/api/shop\_xx/:category' which queries a database for products belonging to a specific category. The VS Code terminal shows the server has started and is running on port 5000. The taskbar at the bottom includes icons for the Start button, search, and various system and application icons.

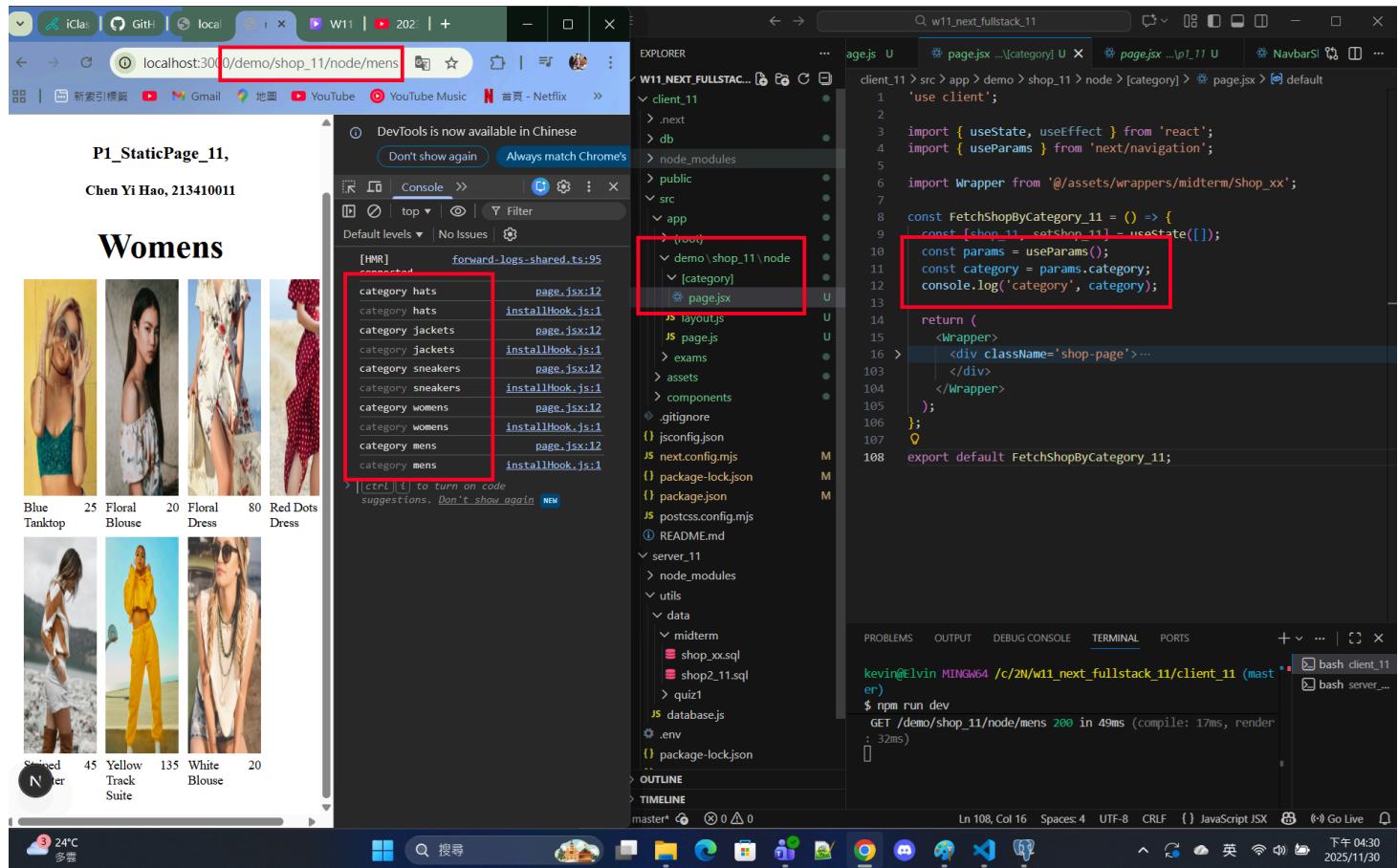
```
server_11 > JS server.js > app.use('/api/shop_xx') callback
1 import express from 'express';
2 import cors from 'cors';
3
4 const app = express();
5
6 import db from './utils/database.js';
7
8 app.use(cors());
9
10 app.use('/api/blog_xx', async (req, res, next) => {
11   const results = await db.query(`select * from blog_xx`);
12   console.log('results', JSON.stringify(results.rows));
13   res.json(results.rows);
14 });
15
16 app.use('/api/shop_xx/:category', async (req, res, next) => {
17   console.log('category', req.params.category);
18   const results = await db.query(
19     `select * from category2_11, shop2_11 where cname = $1 and cid = cat_id`, [req.params.category]
20   );
21   // console.log('results', JSON.stringify(results.rows));
22   res.json(results.rows);
23 });
24
25 app.use('/api/shop_xx', async (req, res, next) => {
26   const results = await db.query(`select * from shop2_11`);
27   console.log('results', JSON.stringify(results.rows));
28   res.json(results.rows);
29 });
30
31 app.use('/', (req, res, next) => {
32   res.send('ChenYiHao, 213410011');
```

1ae0bb0 Hao Yi Chen

Sun Nov 30 15:44:52 2025 +0800 W11-P1: Implement route /api/shop\_xx/:c

# W11-P2: Implement /demo/shop\_xx/node and /demo/shop/node/:category in client

=> show how to get category from params



## => show how to fetch category from the category main page

The screenshot shows a web application interface for a shopping site. On the left, there's a navigation bar with links like TKUdemo, Shop, Contact, Sign In, and a shopping cart icon. Below the navigation, the page title is "P1\_StaticPage\_11," and it displays the user information "Chen Yi Hao, 213410011". The main content area is titled "Womens" and shows four product cards:

- Blue Tanktop
- 25 Floral Blouse
- 20 Floral Dress
- 80 Red Dots Dress

Each card includes a small image of the item, its price, and a "Add to Card" button. The DevTools Console tab in the browser shows the state of `shop_11_data` with two categories: `womens` and `mens`. The code editor on the right contains the following code for `FetchShopByCategory_11`:

```
const FetchShopByCategory_11 = () => {
  const [shop_11, setShop_11] = useState([]);
  const params = useParams();
  const category = params.category;
  console.log('category', category);

  const fetchShopFromNode = async () => {
    try {
      const response = await fetch(`http://localhost:5000/api/shop_xx/${category}`);
      const data = await response.json();
      console.log('shop_11 data', data);
      if(data.length !== 0) {
        setShop_11(data);
      }
    } catch (err) {
      console.log(err);
    }
  };

  useEffect(() => {
    fetchShopFromNode();
  }, []);
}
```

The code uses `useEffect` to call `fetchShopFromNode` when the component mounts. The `fetchShopFromNode` function sends a GET request to the API endpoint `http://localhost:5000/api/shop_xx/${category}` and logs the received data to the console.

## => Chrome, show FetchShopByCategory\_xx.jsx by click Mens

Screenshot of a browser window showing a shopping website for men's clothing. The URL in the address bar is `localhost:3000/demo/shop_11/node/mens`. The page displays a user profile for "Chen Yi Hao, 213410011" and a section titled "Mens" with five product items:

- Camo Down Vest (Image)
- 325 Floral T-shirt (Image)
- 20 Black & White Longsleeve (Image)
- 25 Pink T-shirt (Image)
- Jean Long Sleeve (Image)
- 40 Burgundy T-shirt (Image)

The "Mens" section is highlighted with a red box. The DevTools console on the right shows the following data structures:

```
category womens           installHook.jsx:1
shop_11 data              page.jsx:19
  (7) [ { ... }, { ... }, { ... }, { ... }, { ... }, { ... }, { ... } ]
category womens           page.jsx:13
category womens           installHook.jsx:1
shop_11 data              page.jsx:19
  (7) [ { ... }, { ... }, { ... }, { ... }, { ... }, { ... }, { ... } ]
    0: { cid: 4, cname: 'womens', size: 'large' }
    1: { cid: 4, cname: 'womens', size: 'large' }
    2: { cid: 4, cname: 'womens', size: 'large' }
    3: { cid: 4, cname: 'womens', size: 'large' }
    4: { cid: 4, cname: 'womens', size: 'large' }
    5: { cid: 4, cname: 'womens', size: 'large' }
    6: { cid: 4, cname: 'womens', size: 'large' }
      length: 7
    [[Prototype]]: Array(0)

category womens           page.jsx:13
category womens           installHook.jsx:1
category mens             page.jsx:13
category mens             installHook.jsx:1
shop_11 data              page.jsx:19
  (6) [ { ... }, { ... }, { ... }, { ... }, { ... }, { ... } ]
    0: { cid: 5, cname: 'mens', size: 'large' }
    1: { cid: 5, cname: 'mens', size: 'large' }
    2: { cid: 5, cname: 'mens', size: 'large' }
    3: { cid: 5, cname: 'mens', size: 'large' }
    4: { cid: 5, cname: 'mens', size: 'large' }
    5: { cid: 5, cname: 'mens', size: 'large' }
      length: 6
    [[Prototype]]: Array(0)

category mens             page.jsx:13
category mens             installHook.jsx:1
shop_11 data              page.jsx:19
  (6) [ { ... }, { ... }, { ... }, { ... }, { ... }, { ... } ]
```

The DevTools sidebar shows the following system information:

- 24°C 多雲
- 搜尋
- 2025/11/30 下午 05:10

=> relevant code for FetchShopByCategory\_xx

```
client_11 > src > app > demo > shop_11 > node > [category] > page.jsx > FetchShopByCategory_11.js

1  'use client';
2
3  import { useState, useEffect } from 'react';
4  import { useParams } from 'next/navigation';
5
6  import Wrapper from '@/assets/wrappers/midterm/shop_xx';
7  import Product_11 from '@/components/midterm/Product_11';
8
9  const FetchShopByCategory_11 = () => {
10    const [shop_11, setShop_11] = useState([]);
11    const params = useParams();
12    const category = params.category;
13    console.log('category', category);
14
15    const fetchShopFromNode = async () => {
16      try {
17        const response = await fetch(`http://localhost:5000/api/shop_xx/${category}`);
18        const data = await response.json();
19        console.log('shop_11 data', data);
20        if(data.length !== 0) {
21          setShop_11(data);
22        }
23      } catch (err) {
24        console.log(err);
25      }
26    };
27
28    useEffect(() => {
29      fetchShopFromNode();
30    }, []);
31
32    return (
33      <Wrapper>
34        <div className='shop-page'>
35          <div className='section-title'>
36            <h4 className='text-center'> Chen Yi Hao, 213410011 </h4>
37          </div>
38          <div className='collection-page'>
39            <h1 className='title'>{category}</h1>
40            <div className='items'>
41              {shop_11.map((item)=>(
42                const { pid, pname, price, img_url } = item;
43                return (
44                  <Product_11
45                    key={pid}
46                    img_url={img_url}
47                    pname={pname}
48                    price={price}
49                  />
50                )
51              ))
52            </div>
53          </div>
54        </div>
55      </Wrapper>
56    );
57  };
58
59  export default FetchShopByCategory_11;
```

5d36fb3 Hao Yi Chen

Sun Nov 30 17:14:25 2025 +0800 W11-P2: Implement /demo/shop\_xx/node and