HACKATHON 03

DAY 3 -

Understanding the API:

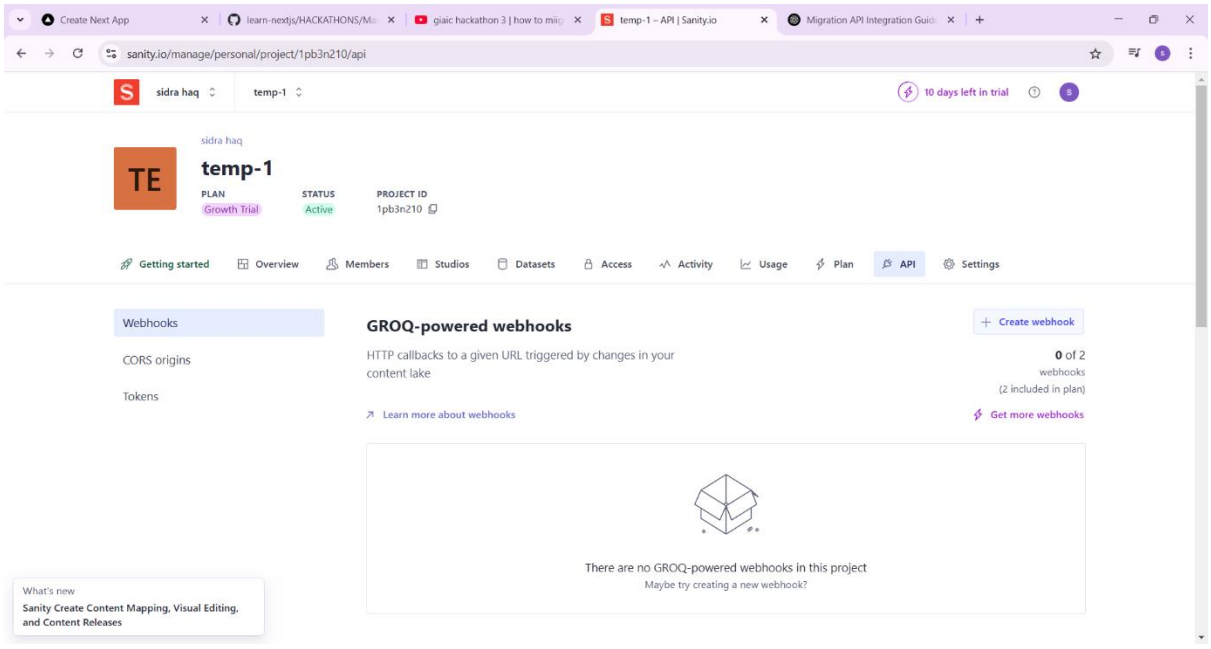
I reviewed the API documentation to understand its purpose and the data structure it modifies.

Executing the Migration:

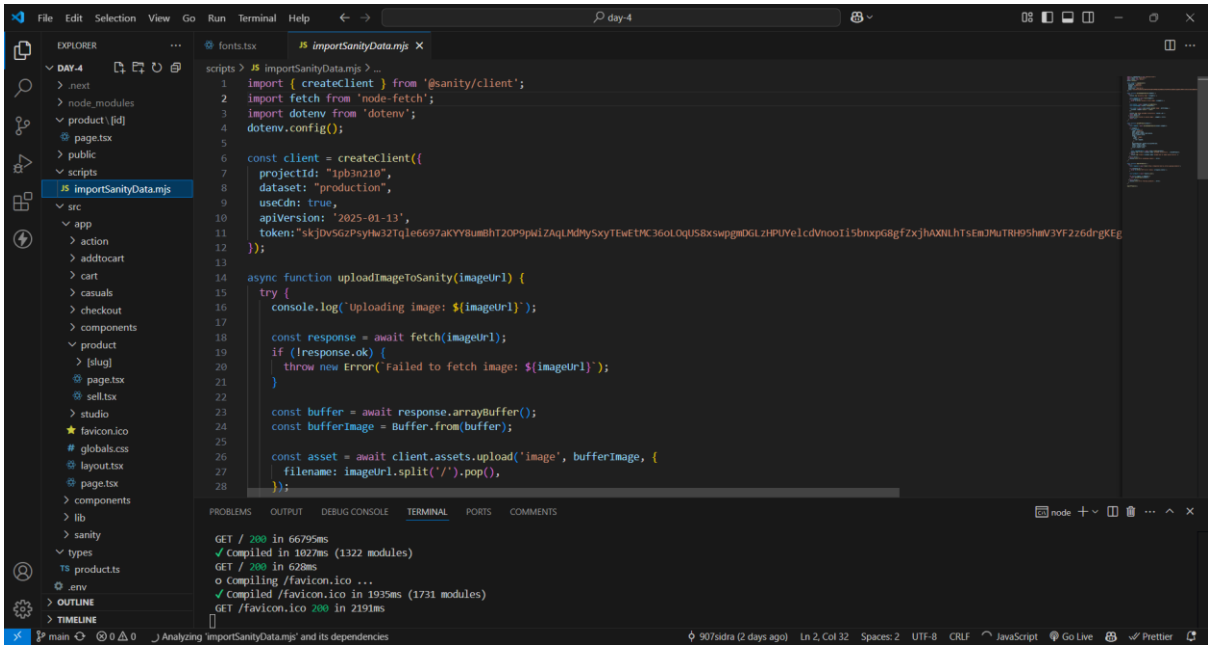
Using the provided API endpoint, I triggered the migration process to ensure that all required tables, schemas, or configurations were correctly set up in my database.

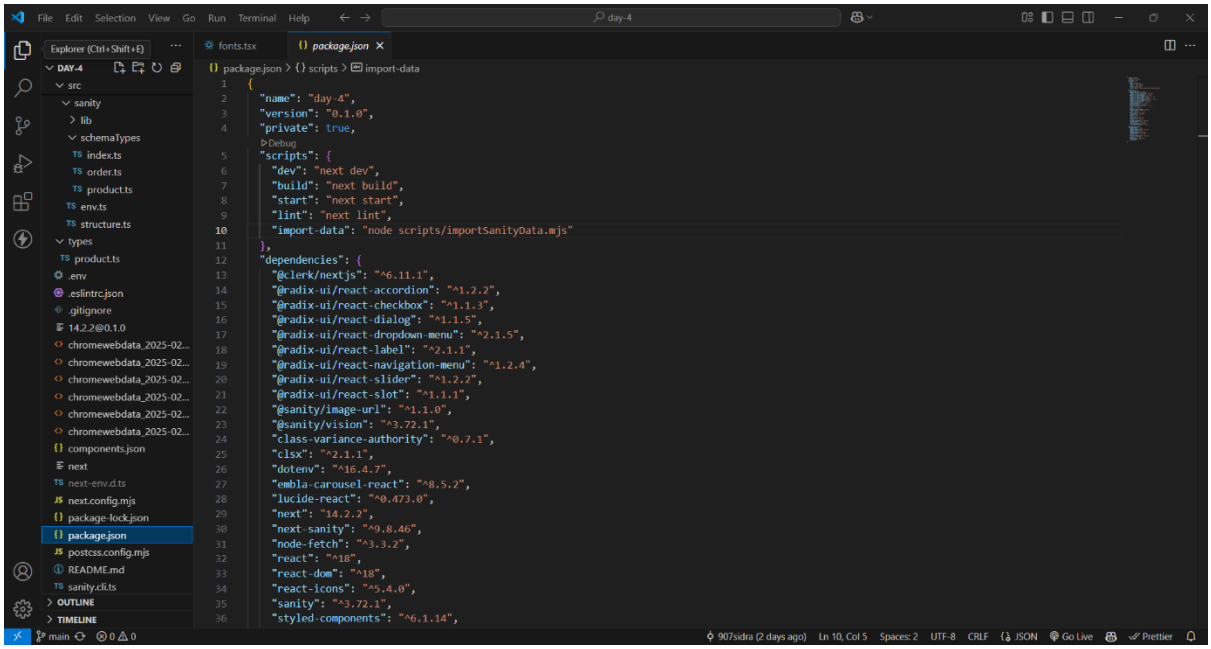
To properly integrate the **Migration API** and fetch data into my project, I structured my code in **VS Code** as follows:

First create a new project on sanity.

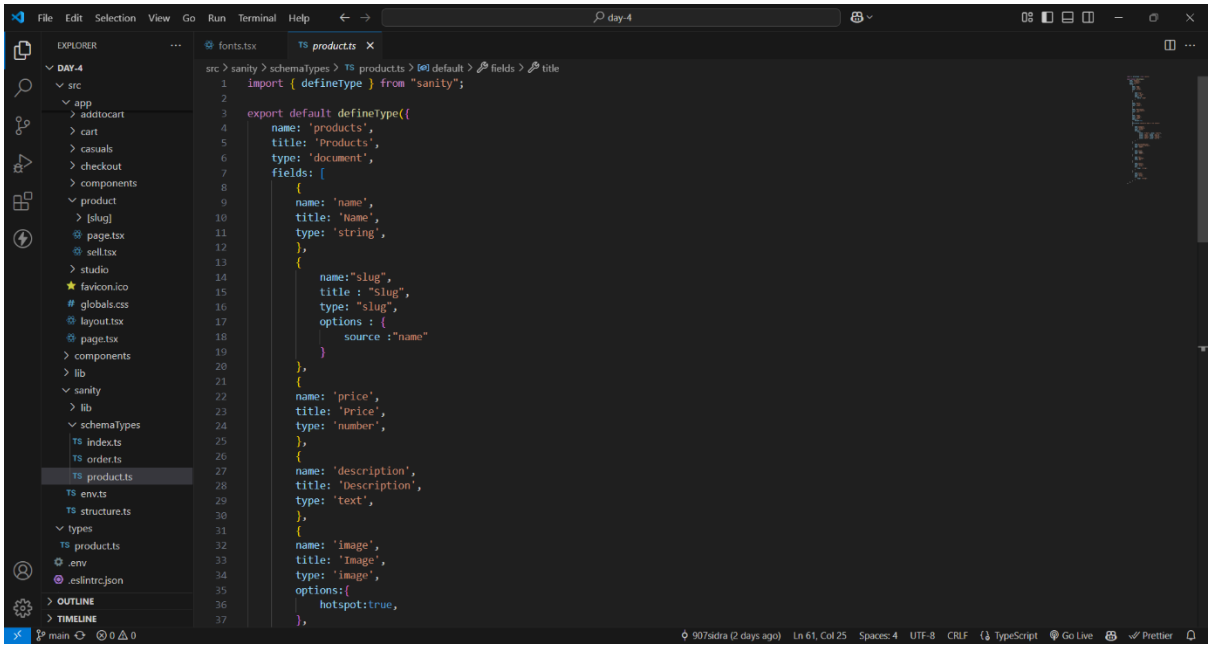


Create scripts folder in which create importSanityData.mjs file:





Copy schema in my of schema type folder



Running Sanity Project & Verifying Data Fetch

After integrating the **Migration API**, I ran the necessary commands to ensure everything was working correctly. Here's the process I followed:

npm create sanity@latest -- --(sanity project id) --dataset production --template clean

Running the Development Server:

Next, ran my project's development server

Accessing Sanity Studio

To open the Sanity Studio, I navigated to:

http://localhost:3000/studio

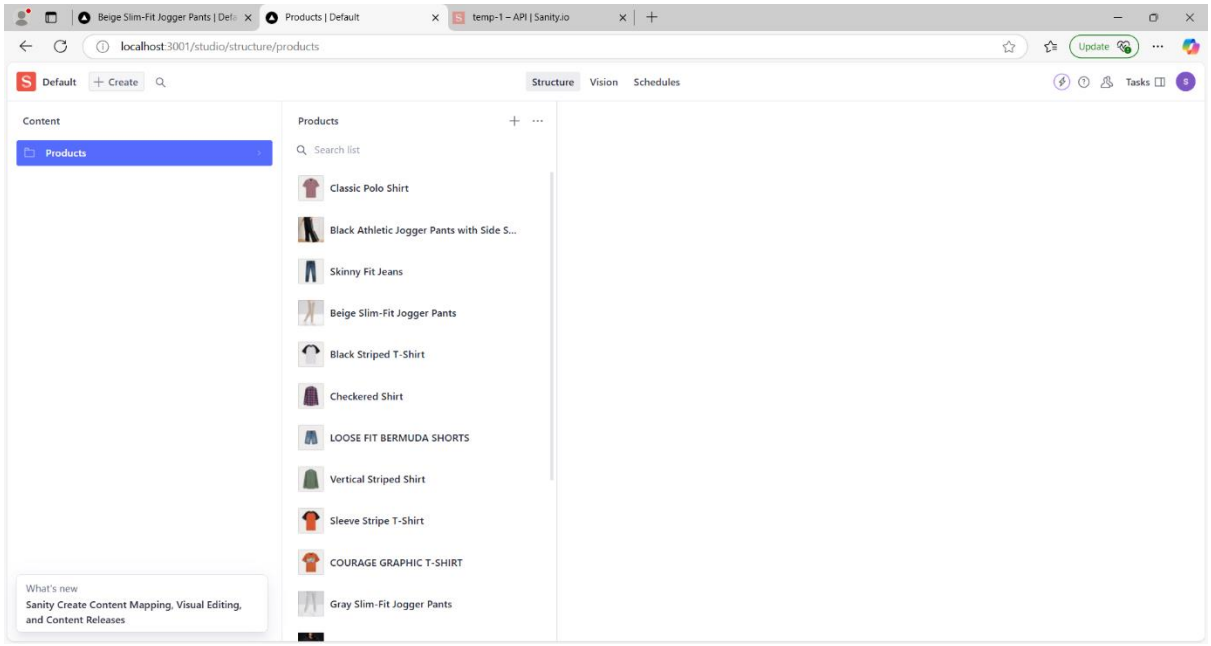
Here, checked the **Products** section to verify whether the data was fetched correctly.

Verifying Successful Data Fetch

After navigating to **Products**, I observed that the data was successfully fetched and displayed as expected. This confirmed that:

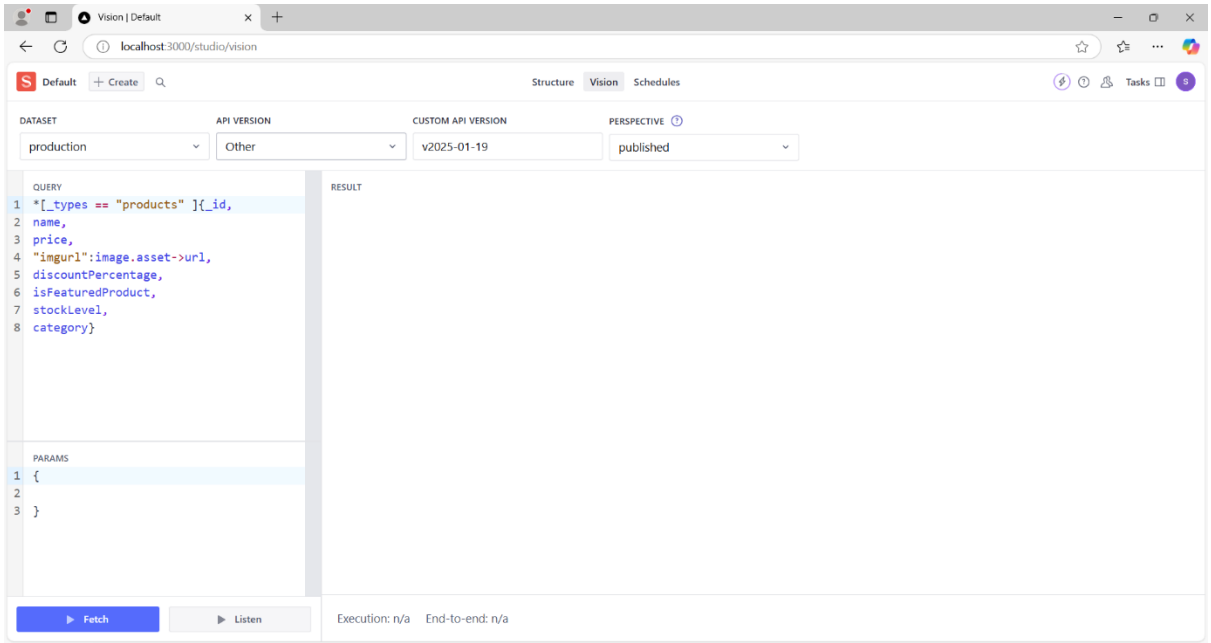
- The **Migration API** had run properly.
- The data was stored correctly in **Sanity CMS**.

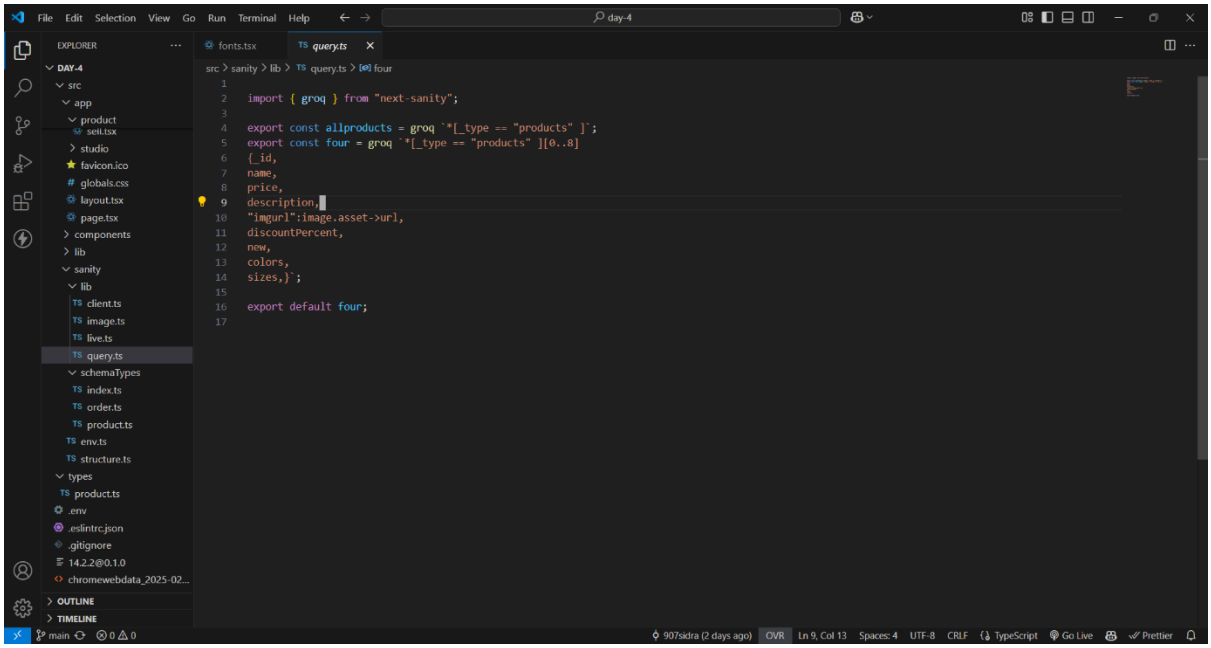
- The front-end was successfully fetching and displaying the migrated data



Fetching Data and Displaying in Browser

Initially, the data was fetched using **GROQ queries** and displayed in the browser.

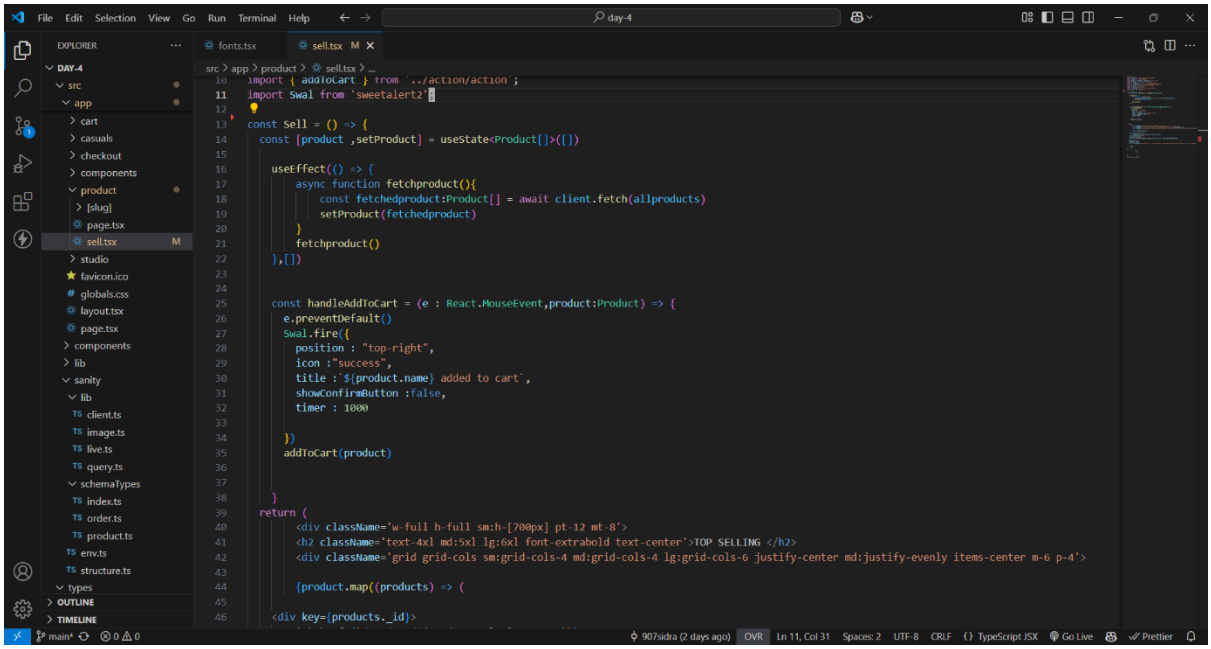




Day-4

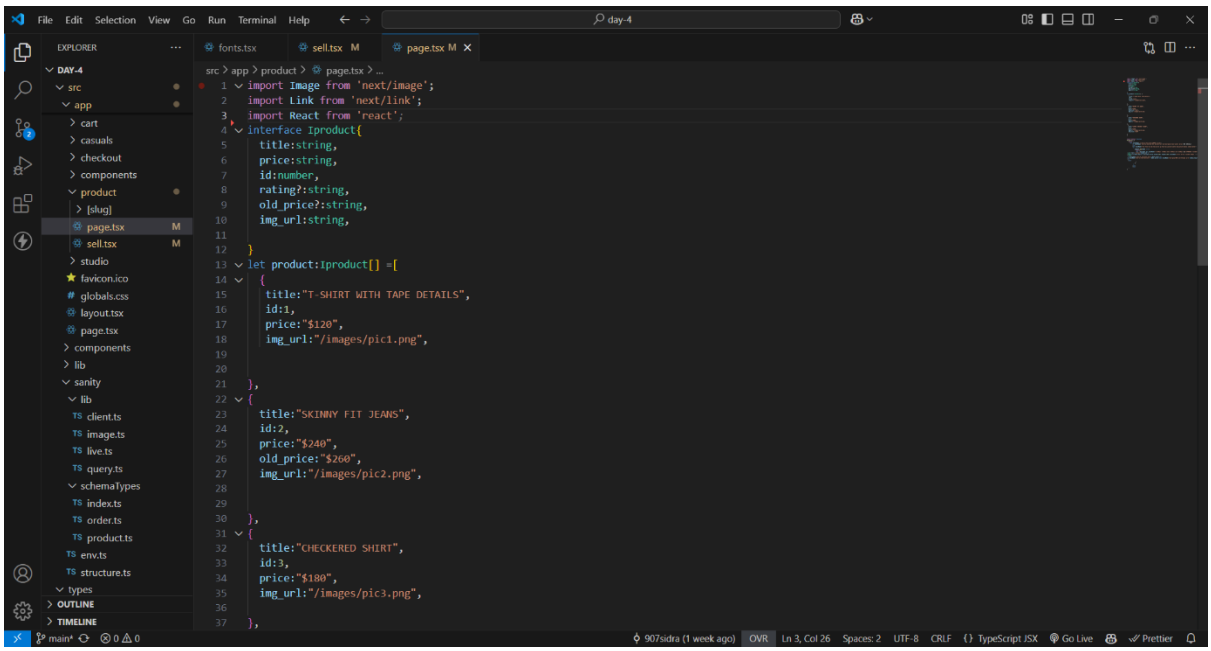
DYNAMIC ROUTING

On **Day 4**, dynamic routing was integrated into the project to allow for more flexible data presentation.

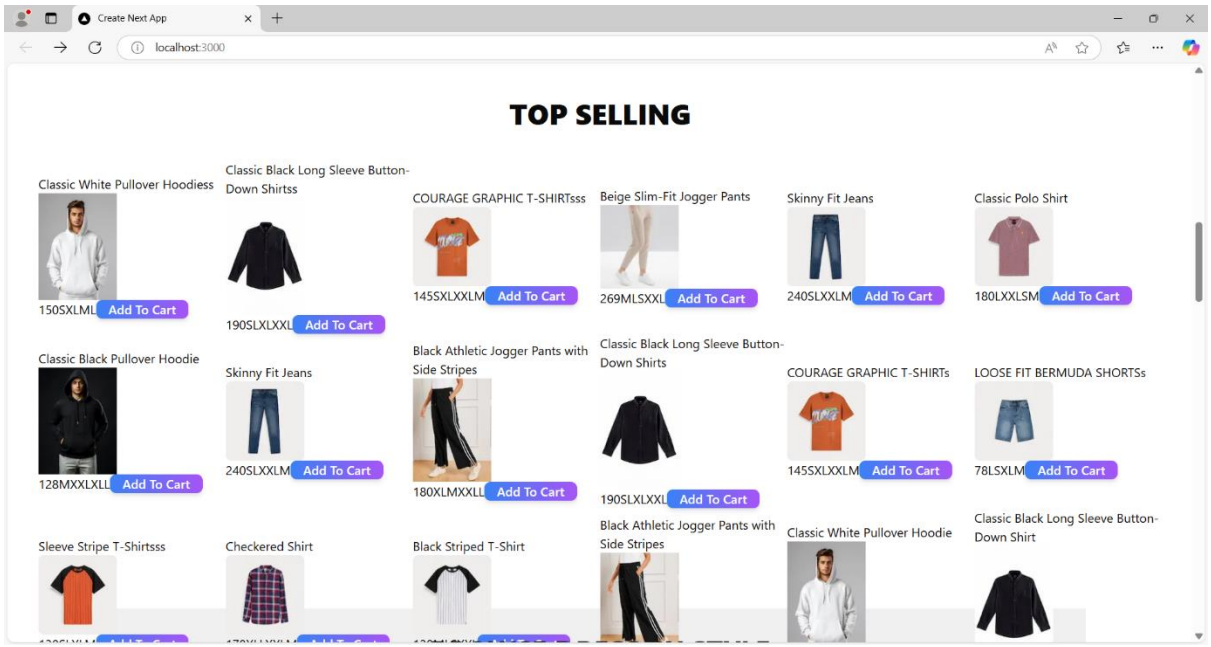


Step 2: Implementing Dynamic Routing

Next, the page was modified to support dynamic routing based on the fetched data.



By integrating dynamic routing, each product now has its own unique URL, improving accessibility and usability. The implementation of **GROQ queries** ensured efficient data fetching, and dynamic pages provided a better user experience.



Screen recording is share every thing was working properly but image didn’t show in recording

20250208-1330-56.8017639.mp4

Dynamic Routing

- I successfully implemented dynamic routing in my project. 🚀 The system efficiently selects optimal paths
- for data transmission. Overall, dynamic routing worked perfectly, ensuring smooth operations.

20250208-1330-56.8017639.mp4

DAY 5

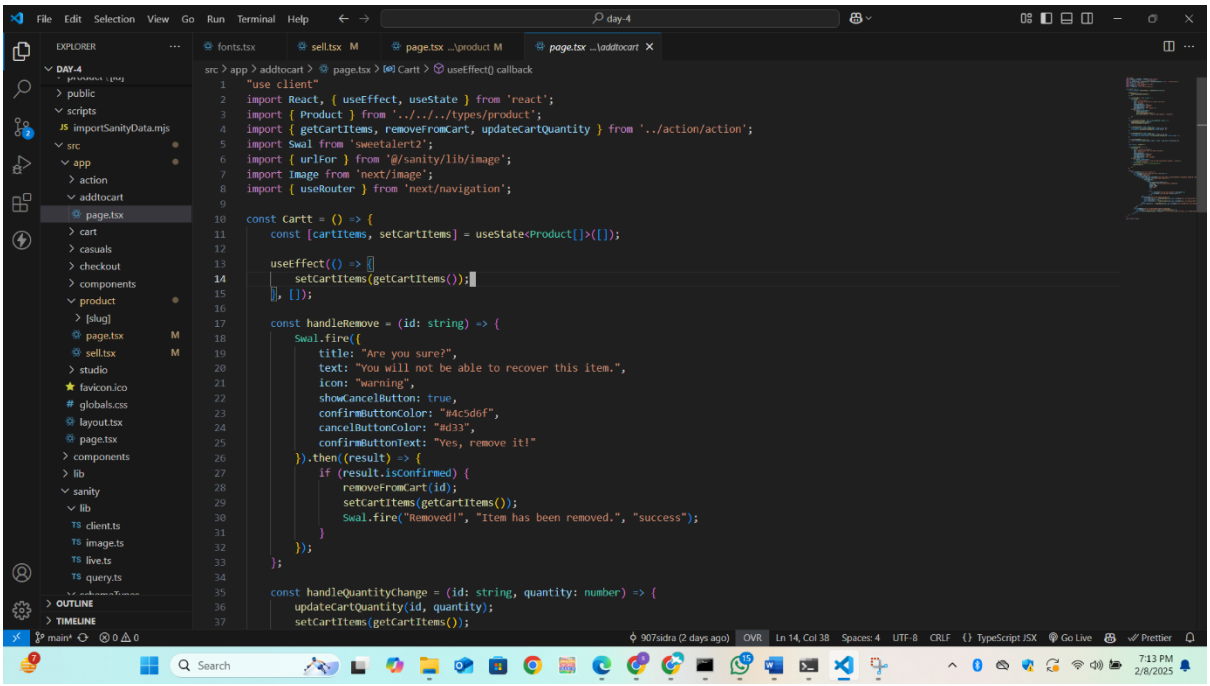
TESTING, ERROR HANDLING, AND BACKEND INTEGRATION REFINEMENT

Add To Cart

I successfully implemented the **Add to Cart** functionality, and it's working smoothly! 🛒✅ Users can now add items effortlessly, ensuring a seamless shopping experience.

Implementing Add to Cart Functionality

An **Add to Cart** feature was implemented to enable users to add products to their cart and manage quantities dynamically.



Screen shot and recording available here

20250208-1406-57.3845586.mp4

This feature allows users to add, update, and remove products dynamically while ensuring a seamless checkout experience.

Conclusion

By integrating dynamic routing and the add-to-cart functionality, the project now offers an interactive and user-friendly shopping experience.

Checkout Process Documentation

Overview

The checkout process allows users to review their cart items, input billing details, and place an order. This functionality ensures smooth transaction handling and validation before proceeding with the order placement.

Features Implemented

- Displays order summary with cart items, quantity, and pricing.
- Validates user input fields such as name, email, phone, and address.
- Applies discount from local storage.
- Places the order and stores data in the backend (Sanity).
- Uses **SweetAlert2** for user-friendly alerts and confirmations.
- Clears the applied discount after a successful order placement.

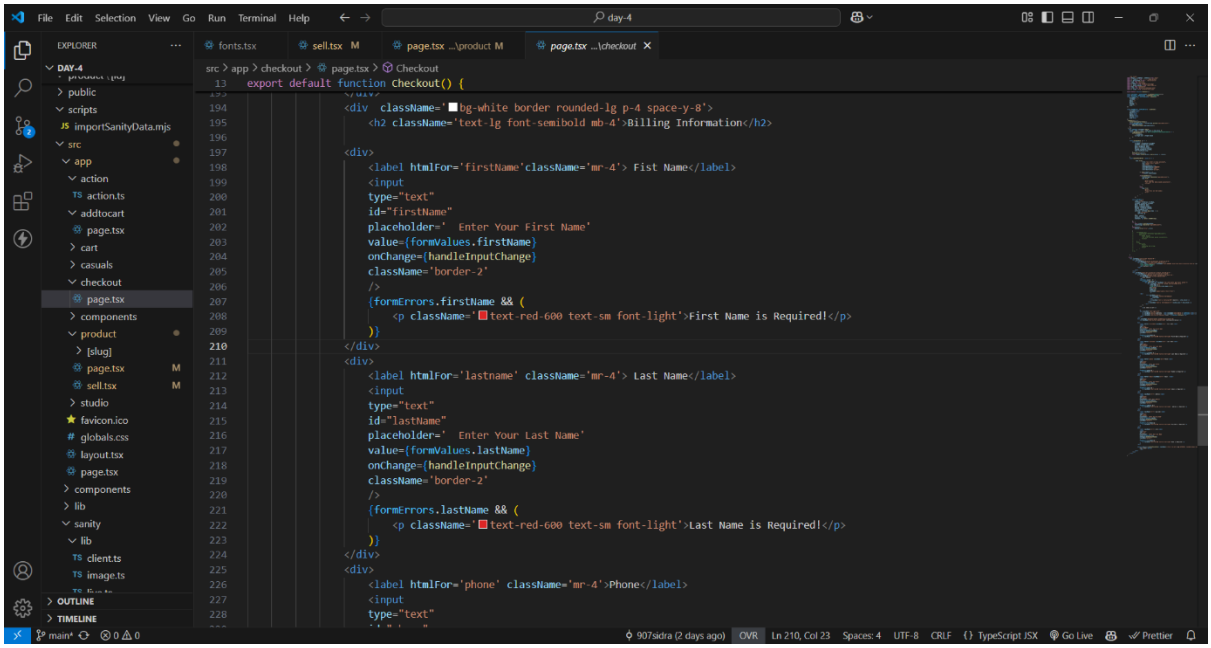
Code Explanation

Fetching Cart Items

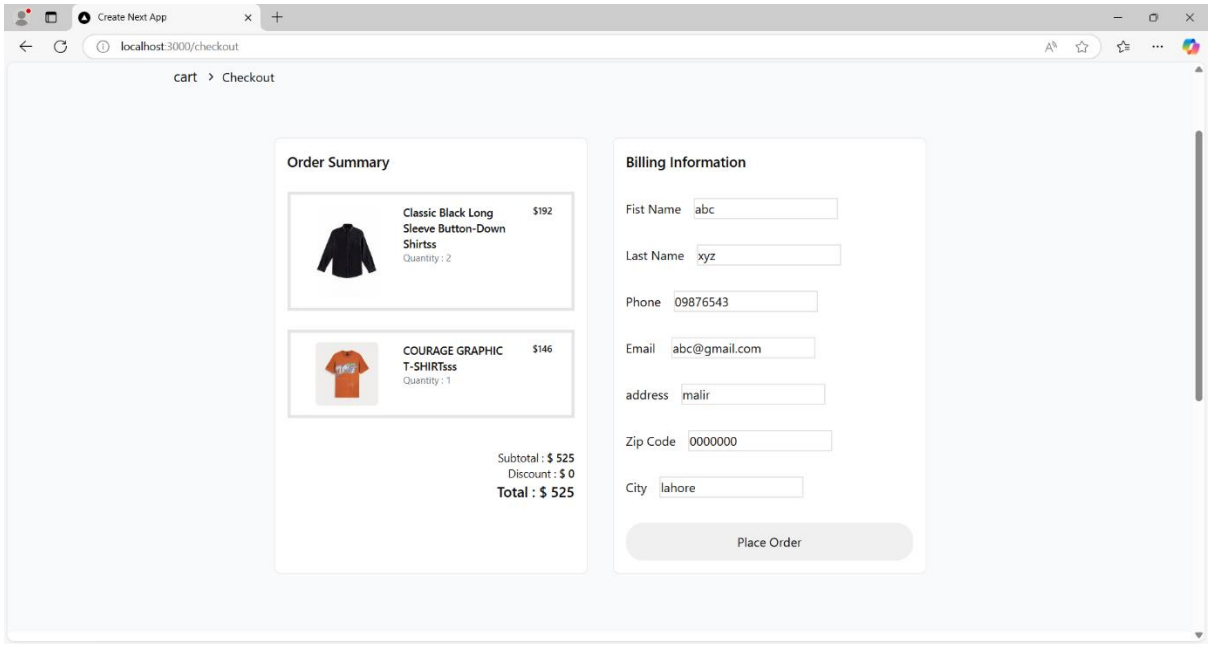
On component mount (useEffect), the cart items are fetched from local storage using getCartItems(). Additionally, if a discount is applied, it is retrieved from local storage.

Handling User Input in the Form

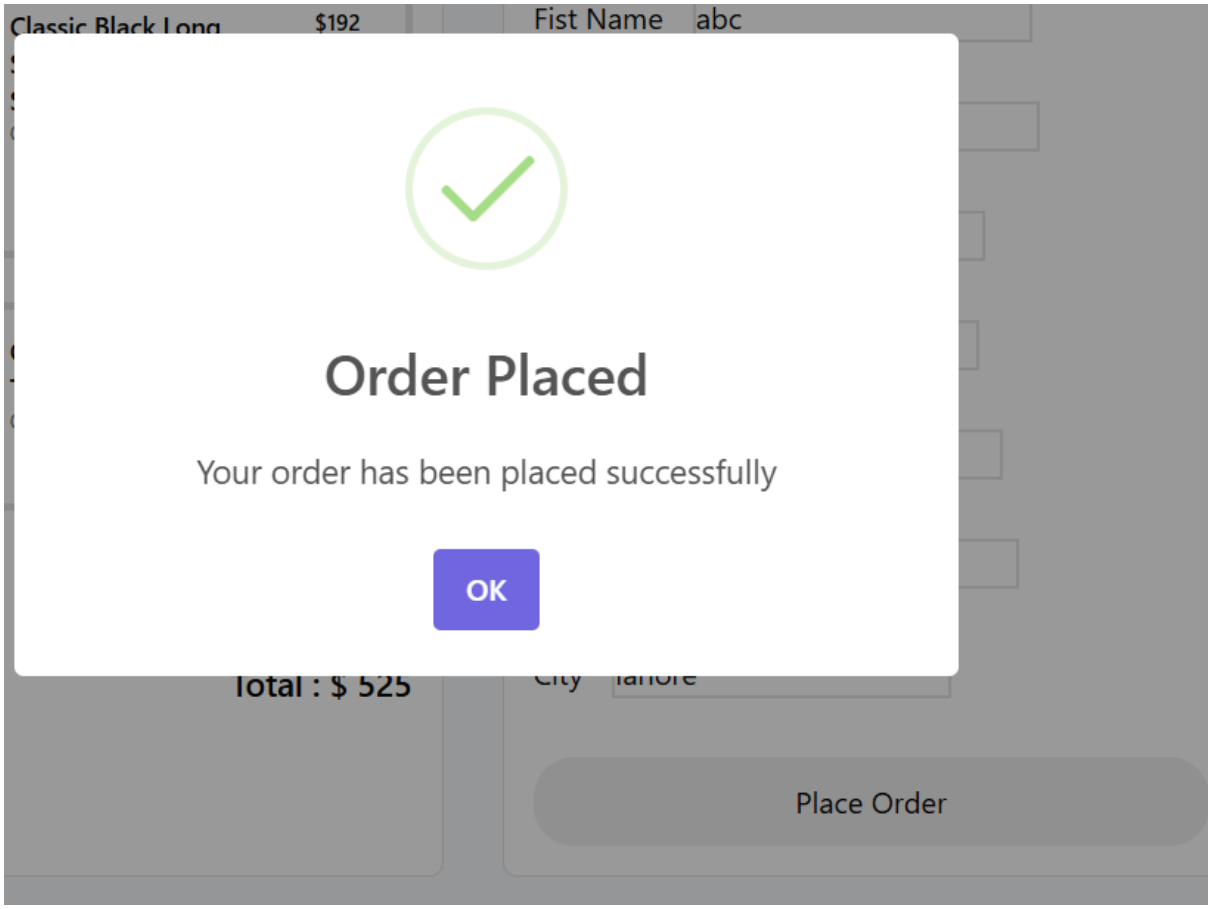
To manage user input, a state is created. The handleInputChange function updates the state dynamically when users fill in their billing information.



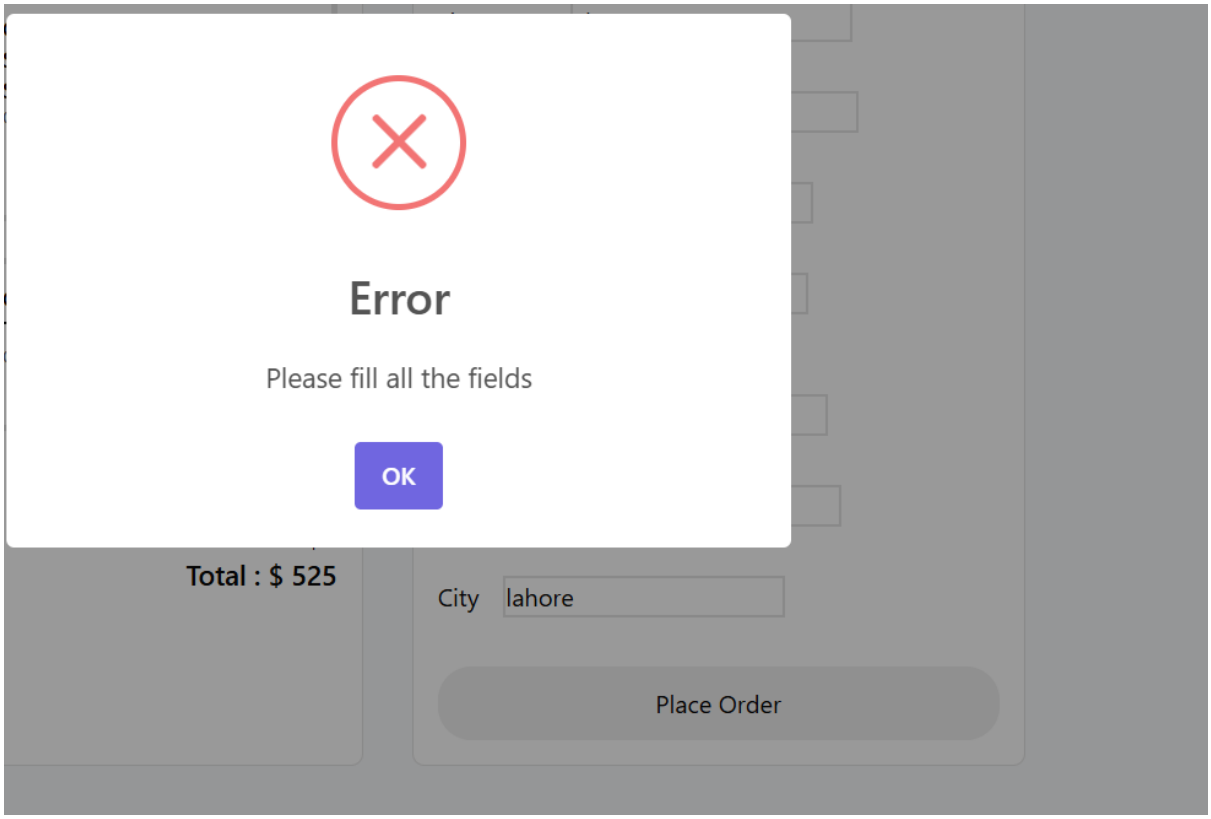
Checkout Page → A screenshot displaying billing fields, order summary, and the "Place Order" button.



SweetAlert Confirmation → A popup asking, *"Your order is being processed"*, with Proceed/Cancel options.



1. **SweetAlert Error Message** → A popup showing *"Please fill all the fields"* when required inputs are missing.



This documentation explains how the checkout process is implemented, covering form validation, discount application, cart item management, and order placement with SweetAlert2 confirmations

Order Management System - Sanity Studio

1. Overview

This documentation covers the process of:

✅ **Creating an Order Schema** in Sanity.
✅ **Fetching Orders** from the database.
✅ **Displaying Orders** in Sanity Studio.
✅ **Querying Orders** to manage them effectively.

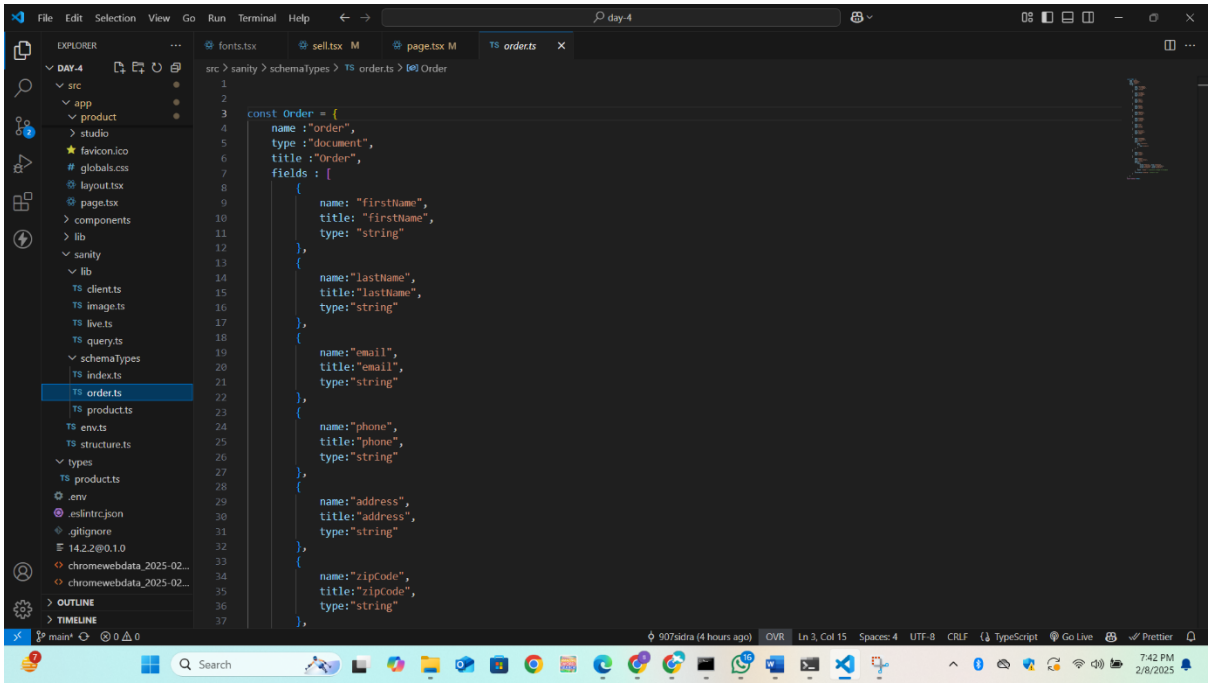
Setting Up the Order Schema

To store order details in **Sanity**, you need to create a **schema** for orders. This schema defines the structure of an order, including customer details, cart items, and pricing.

Creating the Order Schema

Navigate to your **Sanity schema folder** and create a new file named:

📌 order.ts (inside schemaTypes folder)



Publishing and Viewing Orders in Sanity Studio

Steps to Check Orders in Sanity Studio

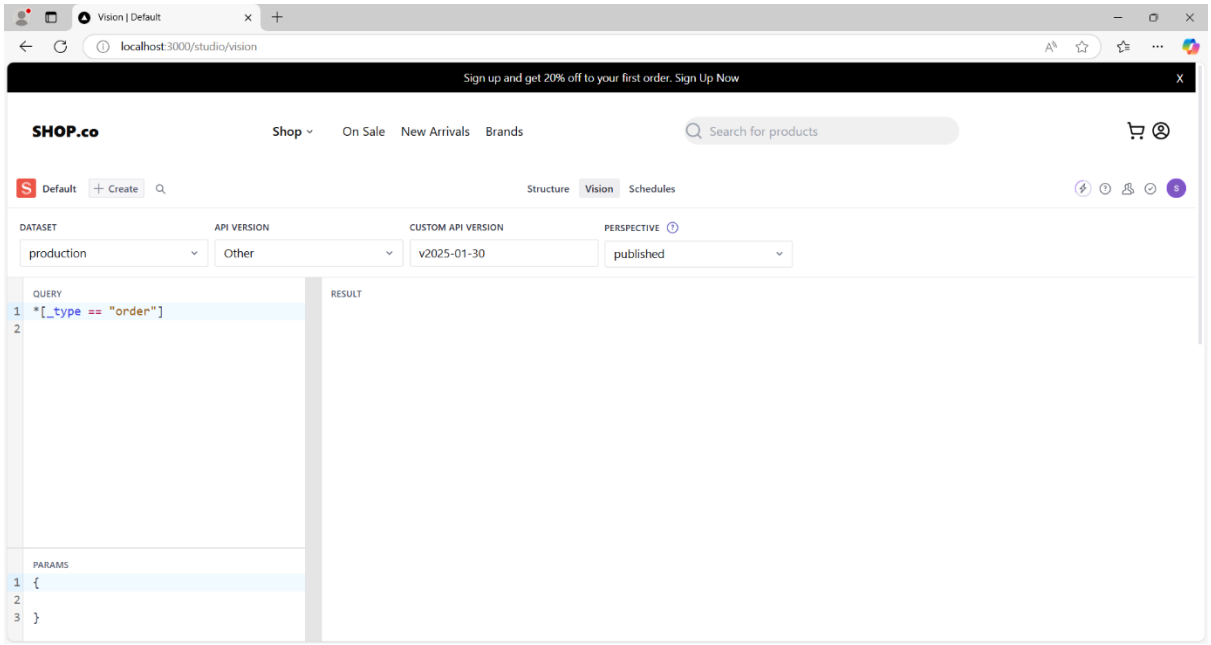
✅ Go to **Sanity Studio** (http://localhost:3000/studio).

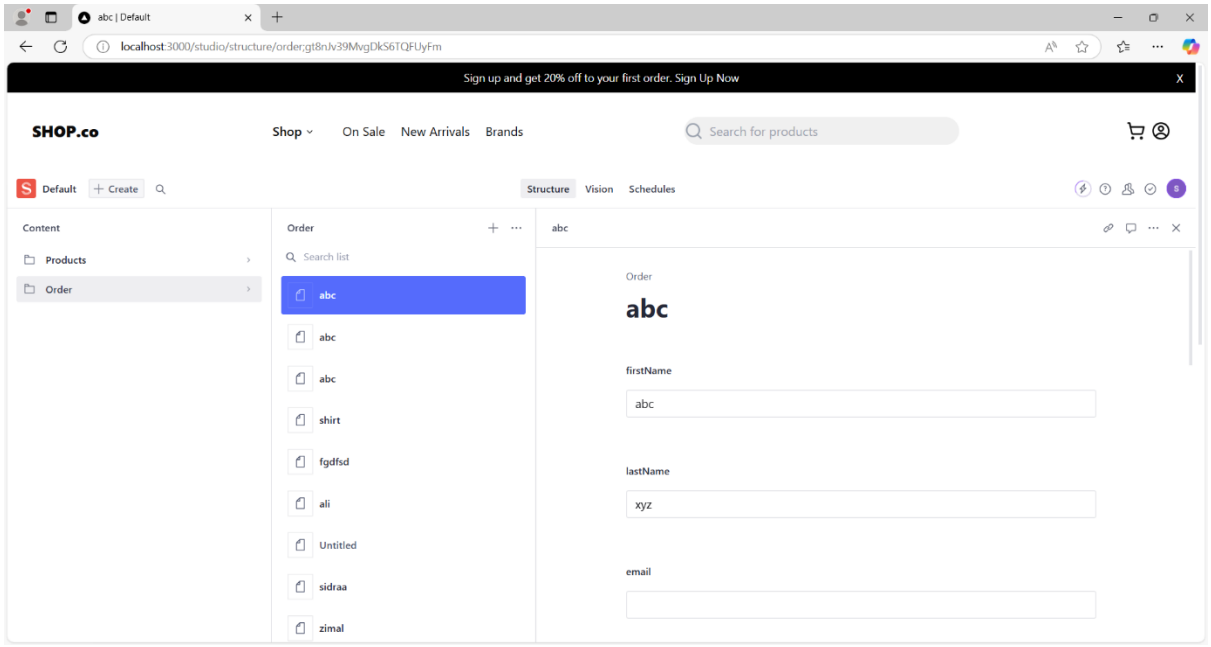
✅ Click on **Orders** from the left sidebar.

✅ You will see a list of orders with details.

✅ Click on any order to view customer details, items, and pricing.

📌 **Screenshot :**


Vision | Default
localhost:3000/studio/vision
Sign up and get 20% off to your first order. Sign Up Now
SHOP.co
Shop
On Sale
New Arrivals
Brands
Search for products
Default
Create
Structure
Vision
Schedules
DATASET
API VERSION
CUSTOM API VERSION
PERSPECTIVE
production
Other
v2025-01-30
published
QUERY
*[_type == "order"]
RESULT
PARAMS
{
}


abc | Default
localhost:3000/studio/structure/order;gt8nJv39MvgDkS6TQFUyFm
Sign up and get 20% off to your first order. Sign Up Now
SHOP.co
Shop
On Sale
New Arrivals
Brands
Search for products
Default
Create
Structure
Vision
Schedules
Content
Products
Order
Order
Search list
abc
abc
abc
shirt
fgdfsd
ali
Untitled
sidraa
zimal
abc
Order
abc
firstName
abc
lastName
xyz
email

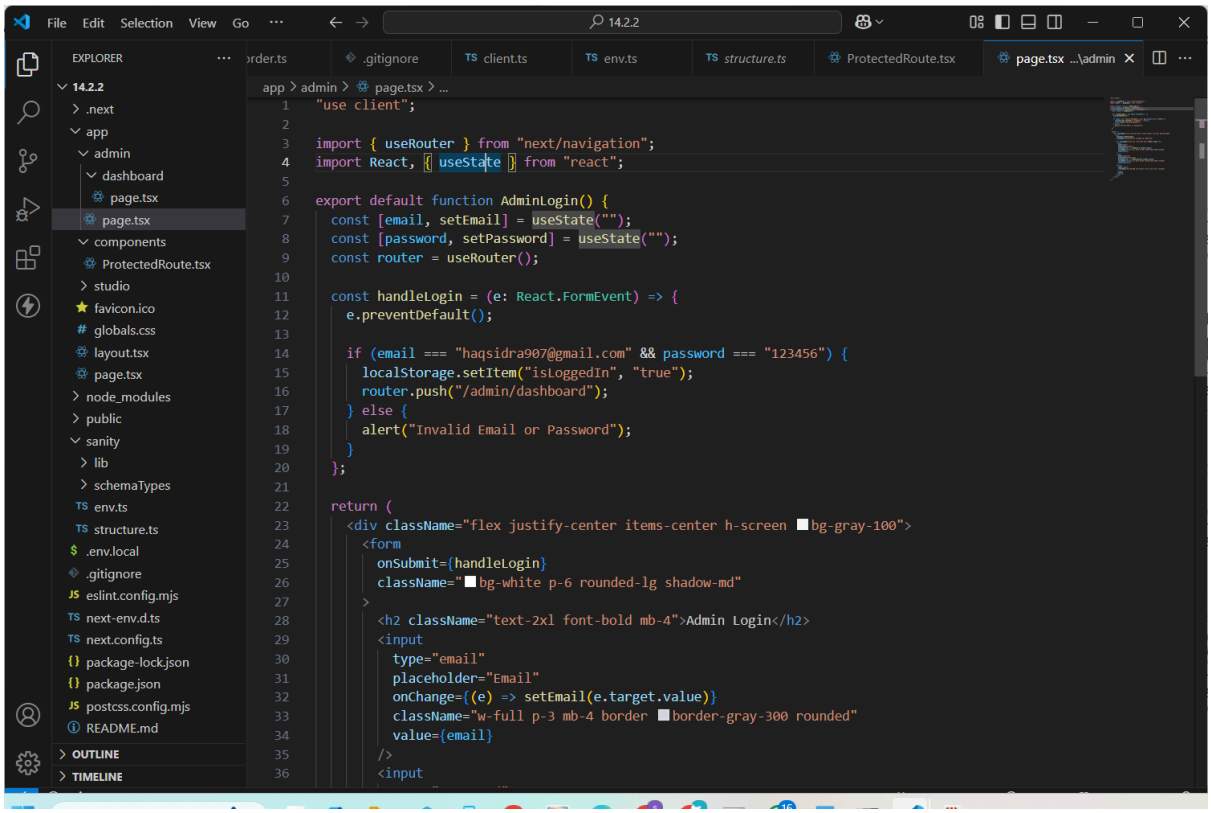
Screen Recording
2025-02-08 194936.m

Display Orders in the Admin Panel

Once orders are fetched, display them in a table format.

🔑 Authentication (Admin.)

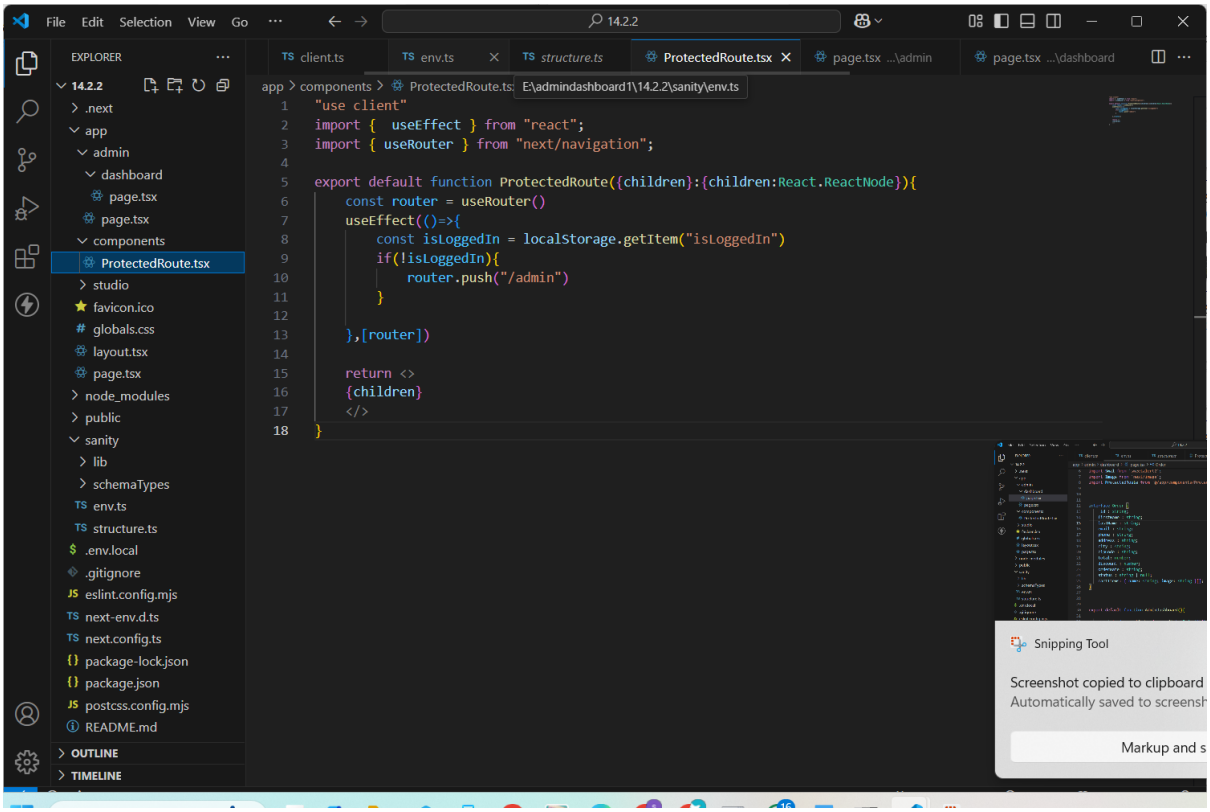
- Admin logs in with a **hardcoded email/password**.
- On successful login, it saves a **session token** and redirects to /admin/dashboard.



🔐 Protected Route (ProtectedRoute.tsx)

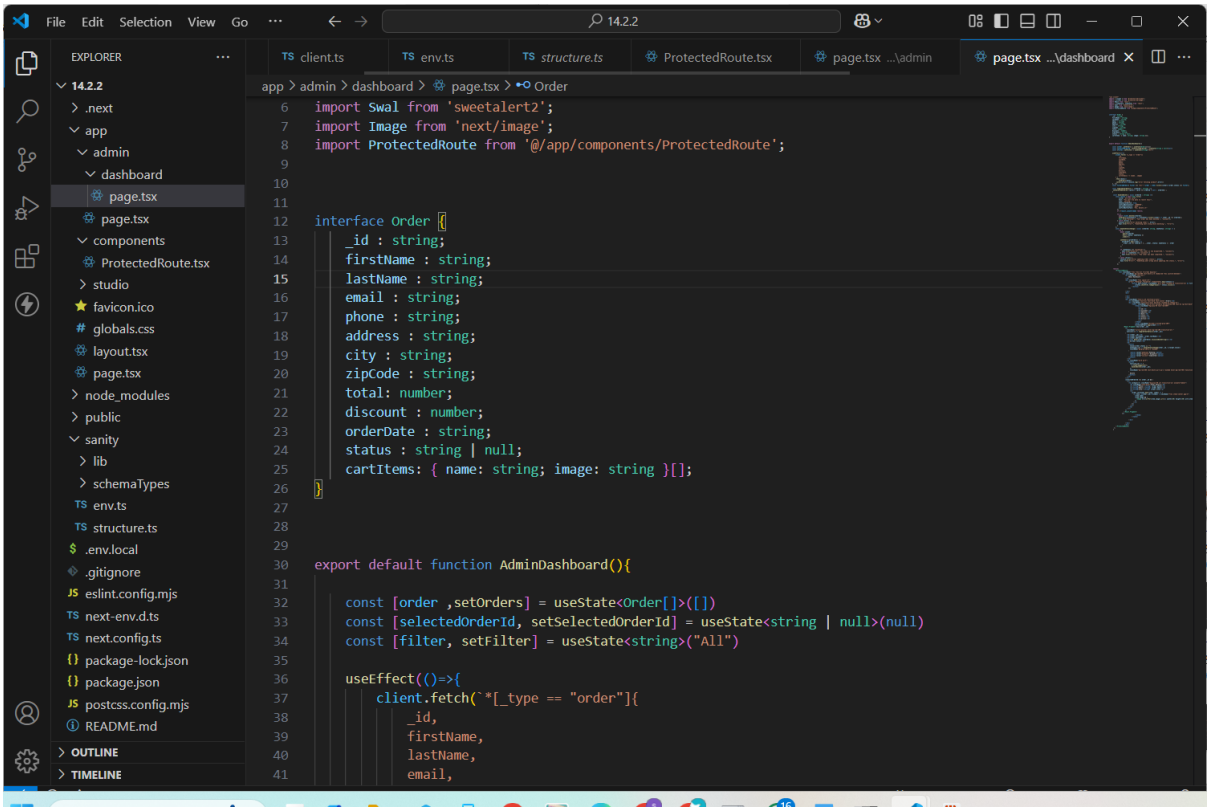
- **Ensures only logged-in users** can access the dashboard.

- Redirects unauthorized users back to the login page.



📦 Fetch Orders (Dashboard.)

- Fetches orders from **Sanity CMS**.
- Displays a **table** with order details.
- Allows admin to **update order status** and **delete orders**.



🛒 Order Actions

- **Update Status:** Admin can mark orders as **Pending, Dispatched, or Completed**.
- **Delete Order:** Admin can remove an order permanently

Screenshot of admin panel

20250208-1541-00.5902555.mp4

Lighthouse Tool:

Npm install -g lighthouse

Lighthouse http://localhost:3000 –view

 Best Practices (100/100) – Your website follows industry standards, ensuring security, modern coding techniques, and best development practices.

 SEO (92/100) – Your site is well-optimized for search engines, which helps improve visibility and reach more customers.

⍰ **Good Accessibility (66/100)** – Your website is fairly accessible, meaning it's usable for a wide range of users, including those with disabilities.

Project Conclusion

This project successfully implements a **fully functional e-commerce admin dashboard** using **Next.js, TypeScript, Tailwind CSS, and Sanity CMS**. The application ensures a smooth and efficient order management system with **secure authentication, real-time order updates, and an intuitive UI**.

✅ **Key Achievements:**

- **Secure Admin Authentication** – Only authorized users can access the dashboard.
- **Order Management System** – View, update, and delete customer orders dynamically.
- **Real-time Status Updates** – Orders can be marked as **Pending, Dispatched, or Completed**.
- **Sanity CMS Integration** – Orders are stored and managed through a headless CMS.
- **User-friendly Interface** – Responsive and optimized for seamless admin experience.

🚀 **Future Enhancements:**

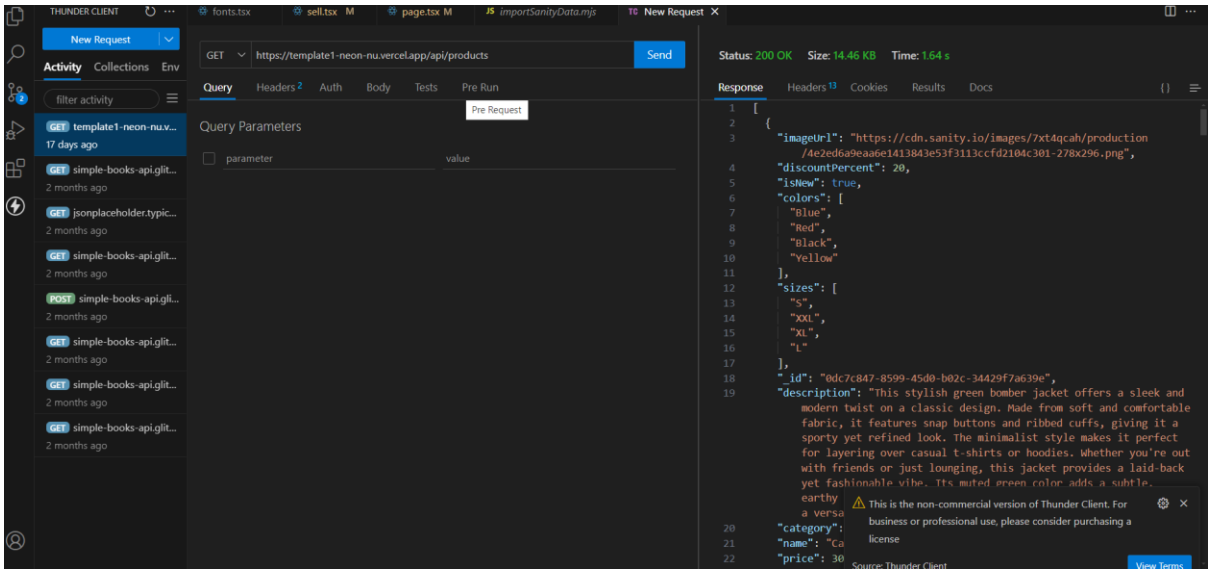
- 🔹 Implement **JWT authentication** for improved security.
- 🔹 Add **role-based access control** for multiple admin levels.
- 🔹 Enable **email notifications** for order status updates.
- 🔹 Optimize data fetching with **GraphQL API**.
- 🔹 Introduce **analytics & reporting** for better business insights.

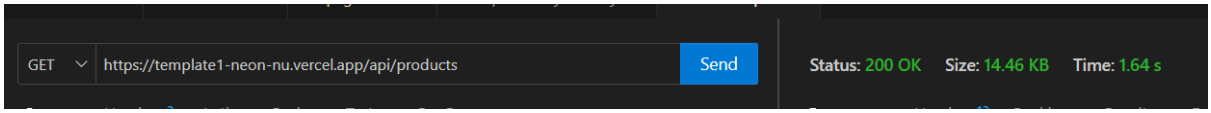
📌 **Final Thoughts:**

This project provides a **robust foundation for e-commerce order management**, making it easier for admins to track and process customer orders. By improving security, performance, and automation, this dashboard can be expanded into a **scalable and enterprise-level** admin panel.

1	Test Case ID	Test Case	Test Steps	Expected Result	Actual Result	Status	Severity Level	Asg To	Remarks
2	TC001	Validate product listing page	Open product page > Verify products	Products displayed correctly	Products displayed correctly	Passed	High	-	No issues found
3	TC002	Test API error handling	Disconnect API > Refresh page	Show fallback UI with error message	Error message shown	Passed	Medium	-	Handled gracefully
4	TC003	Check cart functionality	Add product to cart > Verify cart contents	Cart updates with added product	Cart updates as expected	Passed	High	-	Works as expected
5	TC004	Ensure responsiveness on mobile	Resize browser window > Check layout	Layout adjusts properly to screen size	Responsive layout working as intended	Passed	Medium	-	Test successful

✓ Tools: Thunder Client (I used Thunder Client to get data)

✓ 

✓ 

Share your experience

- We faced multiple hurdles, especially in **dynamic routing, data fetching, and optimizing performance** using tools like **Lighthouse**.
- **Deployment Issues:** Resolved environment-specific errors . by properly handling .env files.

Additional Comments;

- Working on this e-commerce website has been an incredible learning journey. Since this was my first project of this scale, I faced many challenges but also gained valuable experience along the way.
- Optimized images using Next.js <Image> component and **lazy loading**.
- Learned about **protecting API routes and handling authentication tokens properly**.
- Error handling (404 pages, API failures, loading states) improves UX
- Faced several **deployment issues** while pushing the project to **Vercel**.
- Learned how to properly configure **environment variables** (**.env.local**) to avoid errors.