Self-Validation Checklist for Day 1 and Day 2

Day 1: Business Focus Outcome Checklist

1. Business Goals: ✓
2. Market Research: ✓
3. Data Schema Draft: ✓
4. Submission from Day 1: ✓

Day 2: Technical Planning Outcome Checklist:

1. Technical Plan: ✓
2. Workflows: ✓
3. API Requirements: ✓
4. Sanity Schema: ✓
5. Collaboration Notes: ✓
6. Submission from Day 2: ✓

Steps Followed in My Project

1. **Sanity Installation**
   - Installed Sanity in my project.
   - Verified the `projectId` and `token` were configured correctly.
2. **Environment Variable Setup**
   - Ensured that `NEXT_PUBLIC_SANITY_PROJECT_ID`, `NEXT_PUBLIC_SANITY_DATASET`, and `SANITY_API_TOKEN` were written accurately in all necessary files, including the Sanity client configuration.
3. **Data Import**
   - Added an `importdata.mjs` file to the project for importing data into Sanity.
   - Faced some errors during the data import process but resolved them with assistance from ChatGPT.
   - Successfully imported data into the Sanity dataset.

4. **Data Import Challenges**

- Faced errors while importing data due to:
  - Incorrect `.env` file path in the project setup.
  - Mismatch in the token name, where the Sanity API token was correctly referenced as `SANITY_API_TOKEN` in the `.env` file and Sanity client, but mistakenly written as `NEXT_PUBLIC_SANITY_TOKEN` in the import file.
- Adjusted the errors, fixed the path and naming issues, and successfully imported the data into Sanity

5. **Data Fetching for Components**
   - Started fetching data for the "Popular" and "Recommended" components.
     - Used the `popular` tag to fetch data for the "Popular" section.
     - Used the `recommended` tag to fetch data for the "Recommended" section.
   - Utilized the Sanity client for fetching data and successfully displayed the data in the respective components.
6. **Schema Adjustment**
   - Initially created a single schema to handle all vehicles.
   - Realized this approach was inefficient for managing various vehicle types.
   - Revised the schema design by creating separate schemas for each vehicle type, ensuring better organization and scalability.
7. **Final Outcome**
   - Successfully completed the configuration, data fetching, and schema adjustments for a streamlined and functional application.\

1. Schemas

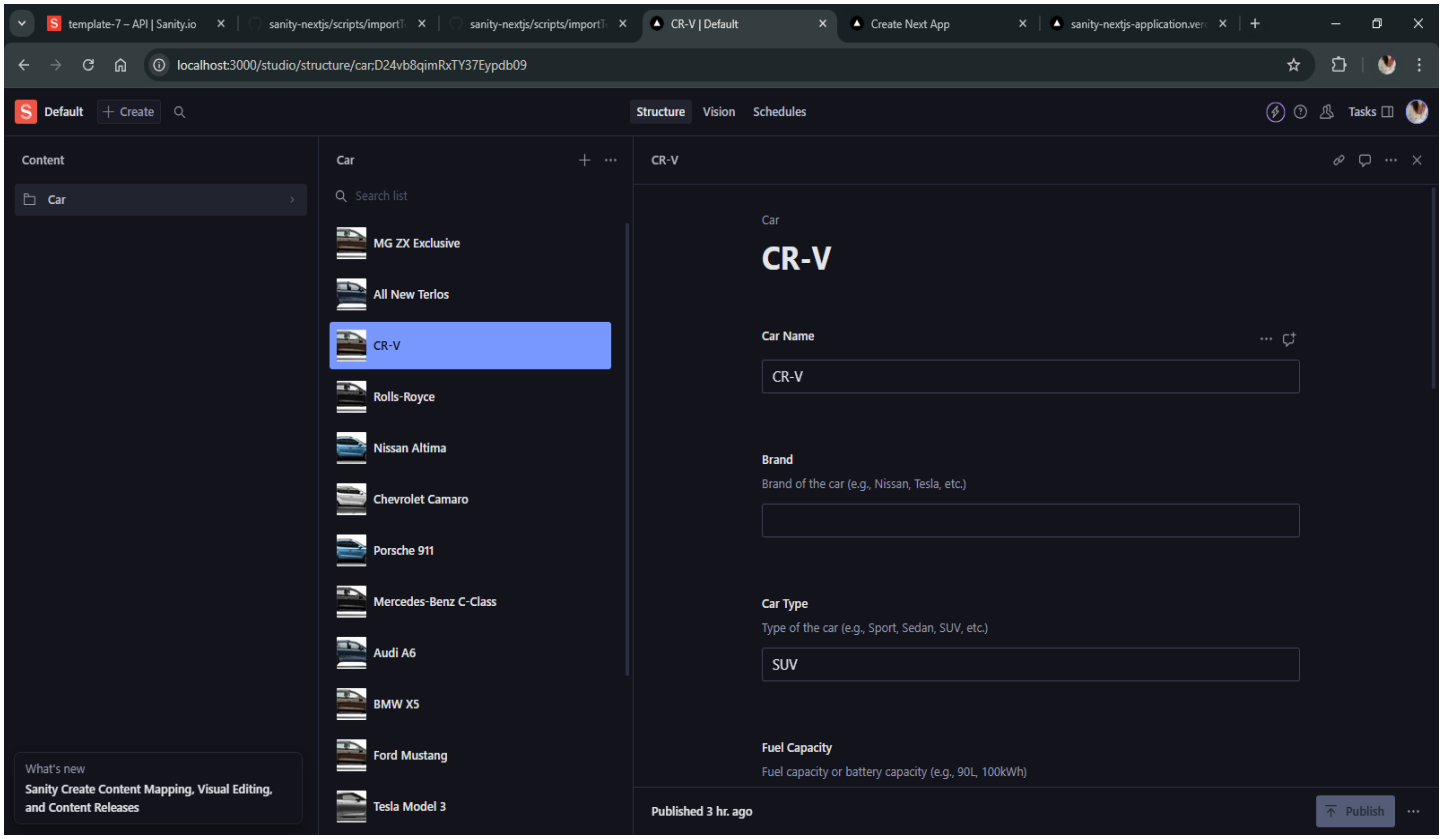
```
File Edit Selection View Go Run Terminal Help
cars.tsx U | importData.mjs U | page.tsx U | index.ts U | .env.local | PopularCar.ts
src > sanity > schemaTypes > cars.tsx > default > fields
```

```
export default {
  name: 'car',
  type: 'document',
  title: 'Car',
  fields: [
    {
      name: 'name',
      type: 'string',
      title: 'Car Name',
    },
    {
      name: 'brand',
      type: 'string',
      title: 'Brand',
      description: 'Brand of the car (e.g., Nissan, Tesla, etc.)',
    },
    {
      name: 'type',
      type: 'string',
      title: 'Car Type',
      description: 'Type of the car (e.g., Sport, Sedan, SUV, etc.)',
    },
    {
      name: 'fuelCapacity',
      type: 'string',
      title: 'Fuel Capacity',
      description: 'Fuel capacity or battery capacity (e.g., 90L, 100kWh)',
    },
    {
      name: 'transmission',
      type: 'string',
      title: 'Transmission',
      description: 'Type of transmission (e.g., Manual, Automatic)',
    },
    {
      name: 'seatingCapacity',
      type: 'string',
      title: 'Seating Capacity',
      description: 'Number of seats (e.g., 2 People, 4 seats)',
```

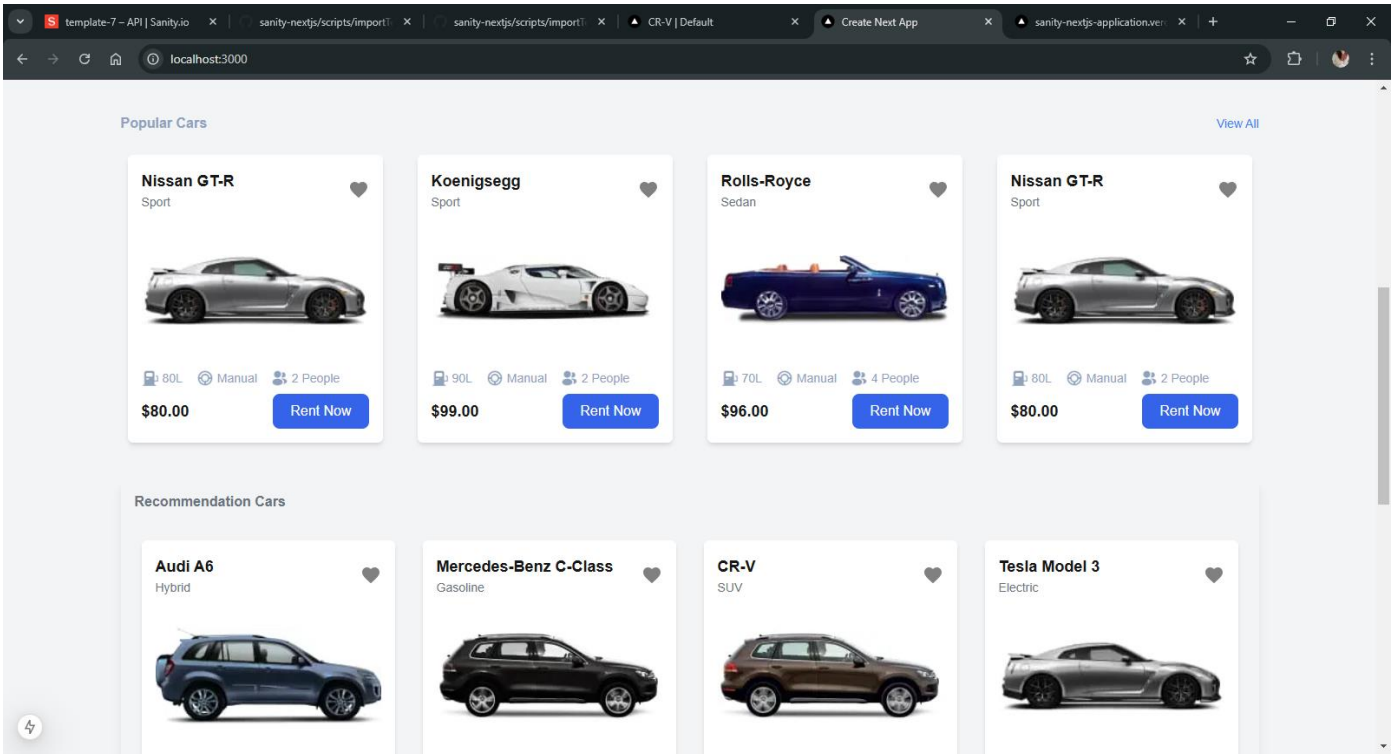
```
Selection View Go Run Terminal Help
tsx U | importData.mjs U | page.tsx U | index.ts U | .env.local | Popu
nity > schemaTypes > cars.tsx > default > fields
```

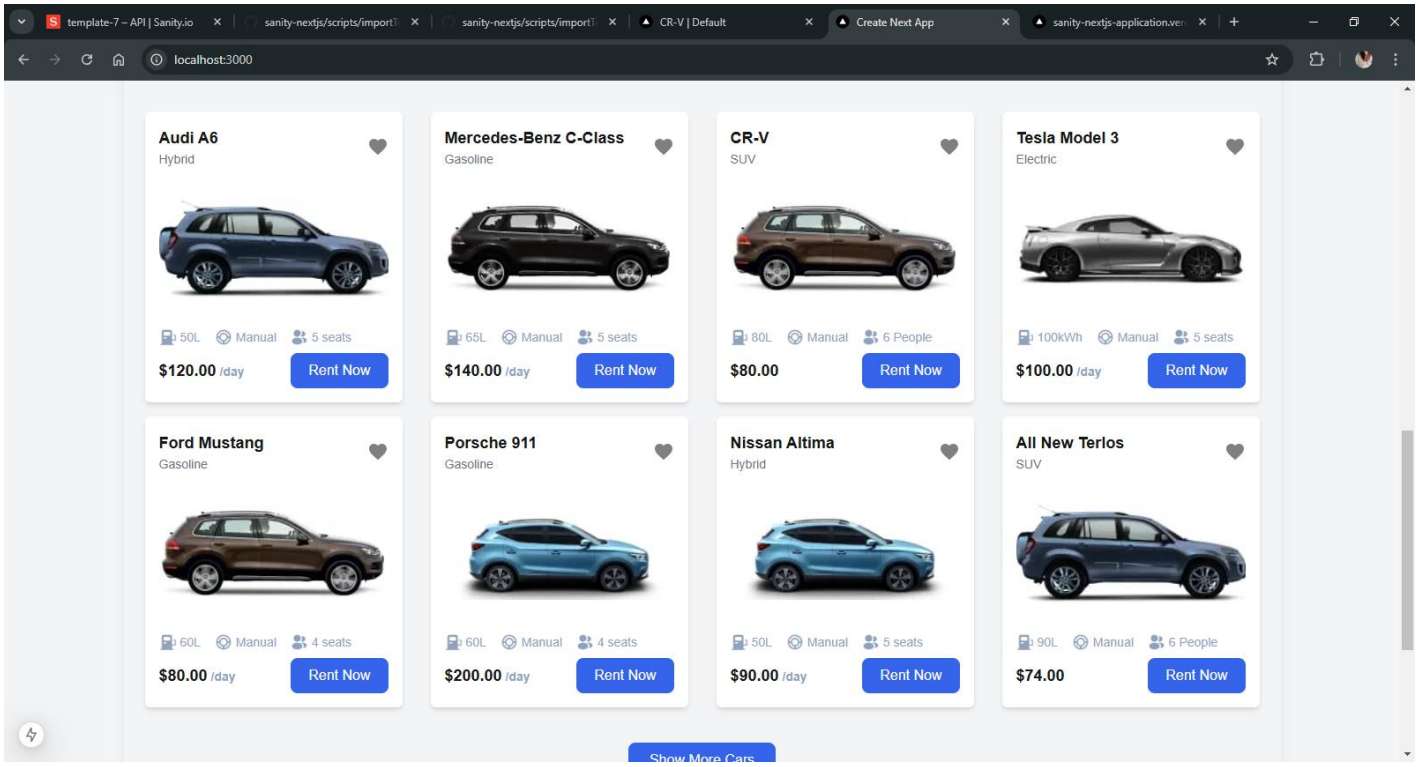
```
export default {
  fields: [
      name: 'seatingCapacity',
      type: 'string',
      title: 'Seating Capacity',
      description: 'Number of seats (e.g., 2 People, 4 seats)',
    },
    {
      name: 'pricePerDay',
      type: 'string',
      title: 'Price Per Day',
      description: 'Rental price per day',
    },
    {
      name: 'originalPrice',
      type: 'string',
      title: 'Original Price',
      description: 'Original price before discount (if applicable)',
    },
    {
      name: 'tags',
      type: 'array',
      title: 'Tags',
      of: [{ type: 'string' }],
      options: {
        layout: 'tags',
      },
      description: 'Tags for categorization (e.g., popular, recommended)',
    },
    {
      name: 'image',
      type: 'image',
      title: 'Car Image',
      options: {
        hotspot: true
      }
    }
  ],
};
```

2. Sanity Data



3. Frontend Fetched Data





4. API calls

```
"use client";
import { useState, useEffect } from "react";
import { client } from "../../sanity.cli"; // Import the Sanity client
import CarCard from "./CarCard"; // Importing CarCard component
import Link from "next/link";
import { Swiper, SwiperSlide } from "swiper/react"; // Import Swiper components
import "swiper/css"; // Import Swiper styles

interface Car {
  _id: number;
  name: string;
  type: string;
  petrol: string;
  transmission: string;
  seats: number;
  price: string;
  image: string;
}

const PopularCars = () => {
  const [carData, setCarData] = useState<Car[]>([]); // Store fetched car data
  const [favorites, setFavorites] = useState<number[]>([]); // Tracks favorite car IDs

  // Fetch data from Sanity
  useEffect(() => {
    const fetchCars = async () => {
      try {
        const data: Car[] = await client.fetch(
          `*[_type == "car" && "popular" in tags]{
            _id,
            name,
            type,
            "petrol": fuelCapacity,
            transmission,
            "seats": seatingCapacity,
            "price": pricePerDay,
            "image": image.asset->url
          }`
        );
```

```
const PopularCars = () => {
  useEffect(() => {
    const fetchCars = async () => {
      setCarData(data);
    } catch (error) {
      console.error("Failed to fetch popular cars:", error);
    }
  };

  fetchCars();
  }, []);

  // Toggle the favorite status of a car
  const toggleFavorite = (id: number) => {
    setFavorites((prev) =>
      prev.includes(id) ? prev.filter((carId) => carId !== id) : [...prev, id]
    );
  };

  return (
    <section className="popular-cars-section max-w-[1312px] mx-auto my-8">
      <div className="flex justify-between items-center mb-4">
        <h2 className="text-[16px] font-bold text-[#90A3BF]">Popular Cars</h2>
        <Link
          href="/category"
          className="text-blue-500 hover:text-blue-700 text-sm font-medium"
        >
          View All
        </Link>
      </div>

      {/* For Large Screens - Using Grid Layout */}
      <div class="hidden md:grid grid-cols-2 lg:grid-cols-4 gap-6 transition-transform duration-300">
        {carData.length > 0 ? (
          carData.slice(0, 4).map((car) => (
            <CarCard
              key={car._id}
              id={car._id}
              name={car.name}
```

```
const PopularCars = () => {
          carData.slice(0, 4).map((car) => (
                name={car.name}
                type={car.type}
                petrol={car.petrol}
                transmission={car.transmission}
                seats={car.seats}
                price={car.price}
                image={car.image}
                isFavorite={favorites.includes(car._id)} // Pass favorite status
                toggleFavorite={() => toggleFavorite(car._id)} // Pass toggle function
              />
            ))
          ) : (
            <p>No cars available</p>
          )}
        </div>

        {/* For Smaller Screens - Using Swiper */}
        <div className="md:hidden">
          <Swiper
            spaceBetween={10}
            slidesPerView="auto" // Adjust number of slides visible at once
            loop={true}
            pagination={{
              clickable: true,
            }}
            grabCursor={true}
          >
            {carData.length > 0 ? (
              carData.slice(0, 4).map((car) => (
                <SwiperSlide key={car._id}>
                  <CarCard
                    id={car._id}
                    name={car.name}
                    type={car.type}
                    petrol={car.petrol}
                    transmission={car.transmission}
                    seats={car.seats}
```

```
const PopularCars = () => {
            {carData.length > 0 ? (
              carData.slice(0, 4).map((car) => (
                <SwiperSlide key={car._id}>
                  <CarCard
                    id={car._id}
                    name={car.name}
                    type={car.type}
                    petrol={car.petrol}
                    transmission={car.transmission}
                    seats={car.seats}
                    price={car.price}
                    image={car.image}
                    isFavorite={favorites.includes(car._id)} // Pass favorite status
                    toggleFavorite={() => toggleFavorite(car._id)} // Pass toggle function
                  />
                </SwiperSlide>
              ))
            ) : (
              <p>No cars available</p>
            )}
          </Swiper>
        </div>
      </section>
    );
  };

  export default PopularCars;
```

Day 3: Self-Validation Checklist:

1. API Understanding: ✓
2. Schema Validation: ✓
3. Data Migration: ✓
4. API integration in Next.js: ✓
5. Submission Preparation: ✓