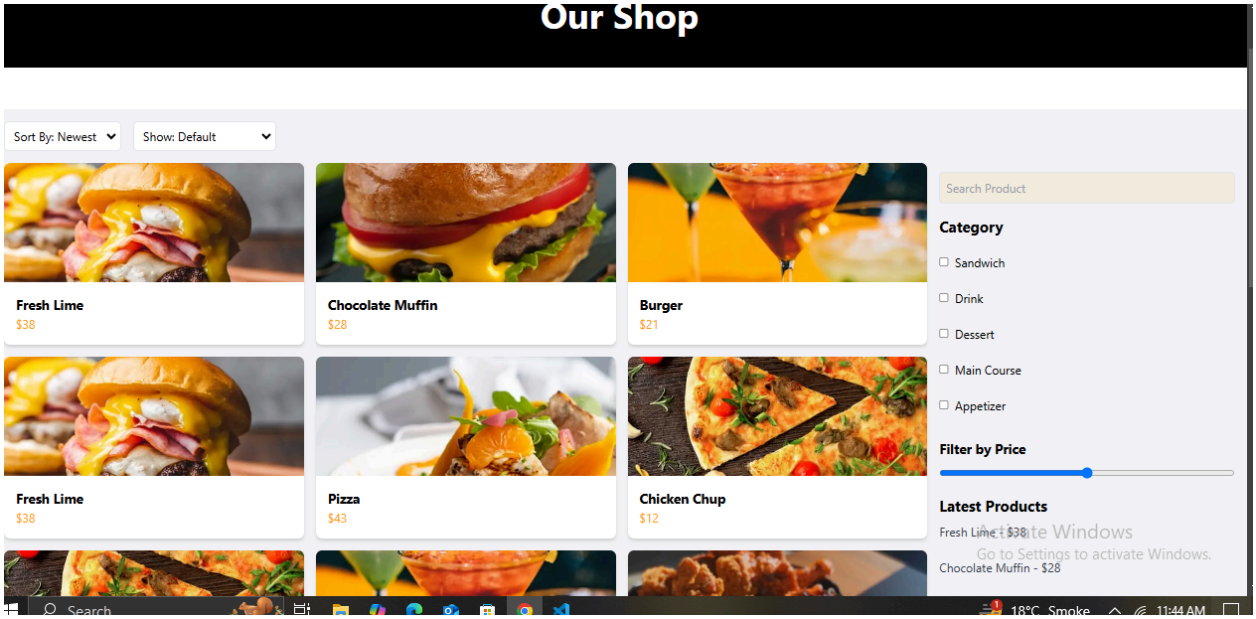
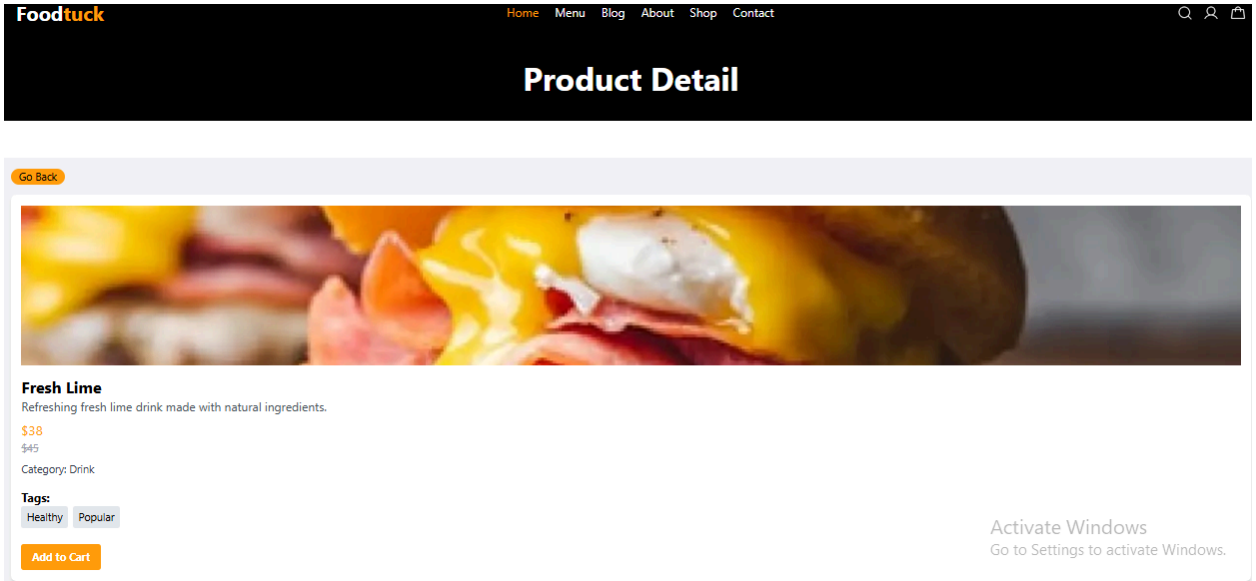
Tab 1

Functional Deliverables:

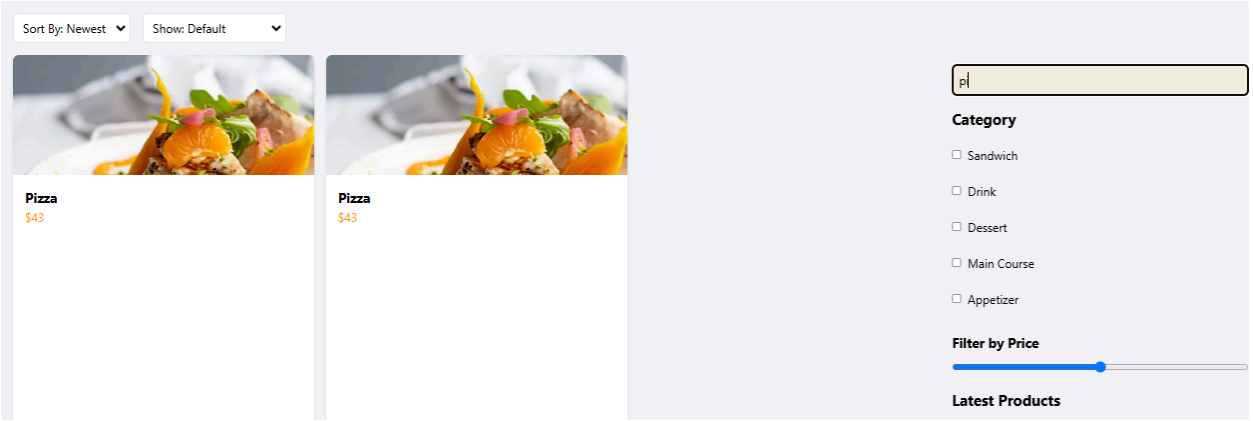
1)The product listing page with dynamic data:

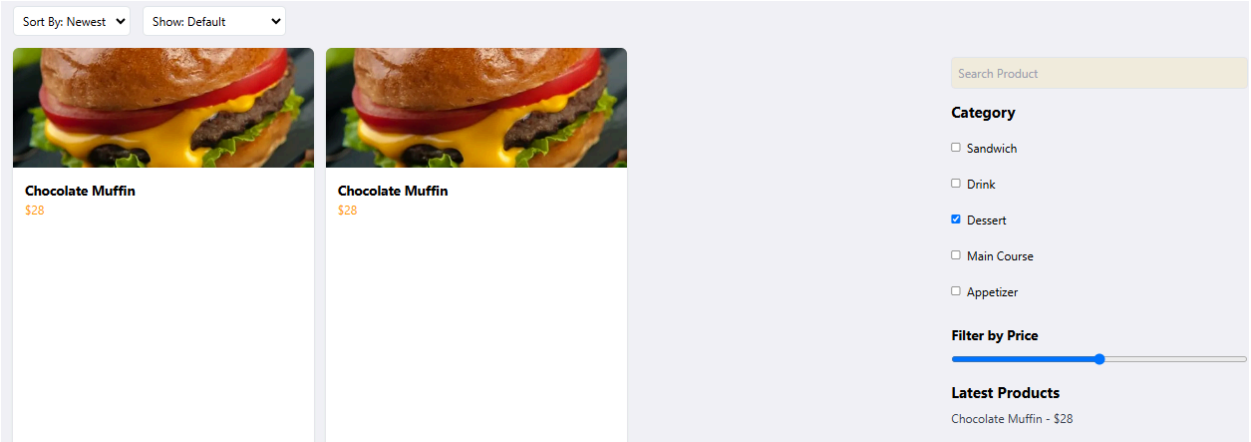


2)Individual product detail pages with accurate routing and data rendering:



3)Working category filters, search bar:





Tab 2

Code Deliverables:

Code snippets for key components (e.g., ProductCard, ProductList, SearchBar):

```
<div className="mb-4 mt-3">
  <input
    type="text"
    placeholder="Search Product"
    className="w-full p-2 border bg-[#FF9F0D1A] rounded-md"
    value={searchQuery}
    onChange={(e) => setSearchQuery(e.target.value)}
  />
</div>

{/* Category Filter */}
<div className="mb-4 flex flex-col gap-3">
  <h2 className="text-[20px] font-bold mb-2">Category</h2>
  <ul className="space-y-2 flex flex-col gap-4">
    <li>
      <label className="flex items-center">
        <input type="checkbox" value="Sandwich" onChange={handleCategoryChange} className="mr-2" /> Sandwich
      </label>
    </li>
    <li>
      <label className="flex items-center">
        <input type="checkbox" value="Drink" onChange={handleCategoryChange} className="mr-2" /> Drink
      </label>
    </li>
    <li>
      <label className="flex items-center">
        <input type="checkbox" value="Dessert" onChange={handleCategoryChange} className="mr-2" /> Dessert
      </label>
    </li>
    <li>
      <label className="flex items-center">
        <input type="checkbox" value="Main Course" onChange={handleCategoryChange} className="mr-2" /> Main Course
      </label>
    </li>
    <li>
      <label className="flex items-center">
```

```
const handleCategoryChange = (e: React.ChangeEvent<HTMLInputElement>) => {
  const { value, checked } = e.target;
  setSelectedCategories((prevCategories) => {
    if (checked) {
      return [...prevCategories, value];
    } else {
      return prevCategories.filter((category) => category !== value);
    }
  });
};

const filteredFoods = foods.filter((food) => {
```

```
<div class="flex flex-wrap">
  {/* Products Section */}
  <div className="w-full lg:w-3/4 grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-4">
    {filteredFoods.map((food) => (
      <div
        key={food._id}
        className="bg-white rounded-lg shadow-md overflow-hidden cursor-pointer"
        onClick={() => handleCardClick(food._id)} // Navigate on card click
      >
        <Image
          src={urlFor(food.image).toString()}
          alt={food.name}
          className="w-full h-40 object-cover"
          width={500}
          height={300}
        />
        <div className="p-4">
          <h3 className="text-lg font-bold">{food.name}</h3>
          <p className="text-[#FF9F0D]">${food.price}</p>
        </div>
      </div>
    ))}
  </div>

  {/* Sidebar */}
  <aside className="w-full lg:w-1/4 lg:pl-4 mt-4 lg:mt-0">
    {/* Search */}
    <div className="mb-4 mt-3">
      <input
        type="text"
        placeholder="Search Product"
        className="w-full p-2 border bg-[#FF9F0D1A] rounded-md"
        value={searchQuery}
        onChange={(e) => setSearchQuery(e.target.value)}
      />
    </div>
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
      }
    });
  };

  const filteredFoods = foods.filter((food) => {
    // Filter based on search term
    const matchesSearch = food.name.toLowerCase().includes(searchQuery.toLowerCase());

    // Filter based on selected categories
    const matchesCategory = selectedCategories.length === 0 || selectedCategories.includes(food.categor

    return matchesSearch && matchesCategory;
  });

  return (
```

Scripts or logic for API integration and dynamic routing:

```
<div className="w-full lg:w-3/4 grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3 gap-4">
  {filteredFoods.map((food) => (
    <div
      key={food._id}
      className="bg-white rounded-lg shadow-md overflow-hidden cursor-pointer"
      onClick={() => handleCardClick(food._id)} // Navigate on card click
    >
```

```
  const handleCardClick = (id: any) => {
    router.push(`/Shop/${id}`); // Navigate to the product detail page
  };
```

Tab 3

Technical Report for Today's Work

1. Introduction:

This report outlines the steps taken to implement and update the **Shopping Cart** functionality, along with additional features such as **individual product detail pages**, **advanced category filters**, and a **search bar** for filtering products.

---

2. Steps Taken to Build and Integrate Components:

a. State Management:

- **useState** and **useEffect** hooks were used to manage the cart state and fetch data from localStorage on component mount.
    - useState manages the cart items' state.
    - useEffect ensures that cart data persists across page reloads by fetching from localStorage and updating the component state.

b. Initialization of Cart Items:

- On fetching cart data, I ensured that each item had a **default quantity** of 1 if the quantity was not provided.
    - This was done by using map() on the fetched cart items and adding a check to assign a default value for quantity.

c. Rendering Cart Items:

- Cart items are displayed dynamically with:
    - **Product Image**: Fetched from each item's image property, with a fallback to a placeholder image.
    - **Product Name, Price, and Total**: These details are displayed alongside each cart item.
    - **Quantity**: An **input field** allows for the modification of the quantity of each item, with real-time updates reflected on the total cost.
    - **Remove Option**: Each item has a button to remove it from the cart, and the state is updated accordingly.

d. Dynamic Routing:

- **Individual Product Detail Pages:**
  - Product pages were implemented using dynamic routing (/products/[id]). Each product has a dedicated page displaying its detailed information such as name, image, description, price, and available quantity.
  - This allows users to click on a product in the cart or on a product listing page to view more detailed information.

- **Advanced Category Filters**:
  - **Category Filtering** was implemented to help users refine their product views based on categories.
  - Filters were created dynamically from product categories, allowing users to narrow down products by selecting relevant categories.
  - This was achieved by mapping through the available categories and dynamically filtering products based on user selection
    .
- **Search Bar**:
  - A **search bar** was added to allow users to filter products by their **name** or **tags**.
  - The search functionality updates in real-time as the user types in the search query, dynamically filtering the displayed products.

e. LocalStorage Updates:

- When removing or updating quantities, the cart's state was synchronized with localStorage to ensure persistence even after page reloads. This was done by using localStorage.setItem('cart', JSON.stringify(updatedCart)) after every change.

f. Handling Responsive Design:

- **Mobile-first approach**: The design was created to be mobile-first with Tailwind CSS:
  - On smaller screens, the cart layout switches to a vertical, stacked form.
  - On larger screens, the cart items are displayed in a table with **scrolling enabled** for better visibility and accessibility.
- **Tailwind CSS utilities** were used for responsiveness, including:
  - Flexbox for layout adjustments.
  - sm:, md:, and lg: breakpoints for responsive behavior.
  - Table elements such as overflow-x-auto for scrolling tables on small screens.

---

3. Challenges Faced and Solutions Implemented:

a. Challenge 1: Ensuring Proper Quantity Initialization:

- **Problem**: Cart items were sometimes not initialized with a quantity, showing an empty value in the quantity field.
- **Solution**: I added logic inside the `useEffect` to ensure that each item had a default quantity of `1` if it was missing.

b. Challenge 2: Handling Different Screen Sizes:

- **Problem**: The cart layout wasn't displaying correctly on smaller screens.
- **Solution**: I used Tailwind CSS to make the layout responsive by adding utility classes for various screen sizes (like `sm:` for small screens). The cart's table became scrollable on smaller screens for better usability.

c. Challenge 3: Real-Time Quantity Update:

- **Problem**: Updating quantities didn't immediately reflect on the total price.
- **Solution**: The `onChange` event was added to the input field for quantity, which triggered a state update, recalculated the total price for the item, and updated the localStorage.

d. Challenge 4: Dynamic Routing Integration:

- **Problem**: Integrating dynamic routing to display individual product details and cart items dynamically was initially complex.
- **Solution**: I used Next.js's dynamic routing to generate URLs for individual products, ensuring the cart was updated dynamically, and users could navigate to product details pages seamlessly. This was done with the `[id].js` file in the `pages` folder for product-specific routes.

e. Challenge 5: Category Filters Implementation:

- **Problem**: Dynamically creating filters based on available product categories was challenging.
- **Solution**: I dynamically created filter options from the product categories and implemented filtering logic that would update the displayed products based on the selected category.

f. Challenge 6: Search Bar Implementation:

- **Problem**: Filtering products by name and tags with a search bar required real-time updates.
- **Solution**: The search bar was integrated with a real-time filter function that checked the product name and tags against the user's query, updating the displayed products instantly.

---

4. Best Practices Followed:

a. Component Reusability:

- I ensured that reusable components like the **Navbar**, **Footer**, and product listing components were created to maintain consistency across the site.

b. Efficient State Management:

- The state was managed using React's built-in `useState` and `useEffect` hooks to ensure smooth data handling without redundant re-renders.
- **localStorage** was leveraged for persistent cart data across sessions, which reduces unnecessary data fetching.

c. Dynamic Routing:

- Next.js's **file-based dynamic routing** was used to create clean, SEO-friendly URLs for individual product pages and to handle dynamic cart data effectively.

d. User Experience (UX):

- **Advanced Filters** and **Search Functionality** were implemented to improve user navigation and product discovery, allowing users to quickly find products based on categories, names, or tags.

This report summarizes the progress made today, highlighting key features like dynamic product pages, advanced category filters, and a search bar, while also detailing the challenges faced and solutions implemented.