Context: On a product listing page, users can view multiple items. Each product has an "Add to Cart" button. During testing, you notice that clicking "Add to Cart" doesn’t update the cart counter.

Questions:
1. Design at least 5 test scenarios for verifying the "Add to Cart" functionality.

2. Write a bug report for the issue where the cart is not updating after clicking "Add to Cart".

3. How would you test if the cart counter is syncing correctly across different pages?

4. What API or network validations would you do in this scenario?

5. If the cart works in staging but fails in production, how would you approach debugging?

1. Design at least 5 test scenarios for verifying the "Add to Cart" functionality.

Add a Single Product to Cart
A product should be added to the cart and the cart count should rise by one when the user clicks the "Add to Cart" button. When the cart is opened, the item needs to be visible as well.

Add Multiple Different Products
The user adds multiple items to their cart. Every time, the cart counter should update accurately, and every new item should show up in the cart.

Add the Same Product More Than Once
When the user hits "Add to Cart" many times for the same product, the cart must accurately display the proper quantity, and the cart counter should increment correspondingly.

Add to Cart Without Refreshing the Page
The cart counter should update instantly after selecting "Add to Cart," guaranteeing a seamless user experience without requiring a site refresh.

Add to Cart as a Guest User
The application's behavior for guest users should be followed when a user who is not signed in adds a product to the cart; the item should still be added appropriately, and the cart counter should update as anticipated.

2. Write a bug report for the issue where the cart is not updating after clicking "Add to Cart".

Bug Title:
Cart counter does not update after clicking “Add to Cart”

Description:
The product does not appear in the cart counter when a user clicks the Add to Cart button on the product listing page. The cart count doesn't change even after the button is clicked, giving the impression that the item wasn't added. Users may become confused by this, click the button more than once, or abandon the website.

Steps to Reproduce:

1) Open the product listing page.

2) Click the Add to Cart button on any product.

3) Check the cart counter at the top of the page.

Expected Result:
The added item should be displayed and the cart counter should instantly increase by one.

Actual Result:
After selecting Add to Cart, the cart counter remains unchanged.

3. How would you test if the cart counter is syncing correctly across different pages?

I would first add an item to the cart from the product listing page and make sure the cart counter changes instantly without requiring a page reload in order to test whether the cart counter is correctly syncing across multiple pages. I would verify that the application gets the right response and that the request to add the item to the cart is successful using browser developer tools.

To make sure the cart counter is reading from the same data source and shows a constant value everywhere, I would then move between other pages, such as the product details page, homepage, and cart page. To make sure the cart state is maintained and not just reliant on in-memory data, I would reload every page.

I would then add additional items from different pages and monitor the network requests to confirm that the correct cart API calls are triggered and return accurate cart data. I would also verify that the UI updates match the backend response.

To make sure the cart counter stays synchronized, I would also launch the program in a different browser tab during the same session. To ensure that the cart data is appropriately stored and fetched from the backend, I would log out and back in for users who are currently logged in. I would make sure that the cart remains active for the duration of the session for visitors.

Finally, I would test edge cases such as rapid multiple clicks on “Add to Cart” and navigation during ongoing API calls to ensure the cart counter remains accurate and does not become out of sync.

4. What API or network validations would you do in this scenario?

1) Verify whether the "Add to Cart" API is being called.
I would start by confirming whether or not the API request is being triggered by clicking the Add to Cart button. The cart counter will obviously not update if the request is not being sent. Additionally, I would make sure the API provides a success status, such as 200 or 201.

2) Validate request payload sent to the API
I would inspect the request payload and make sure it contains the correct productId, quantity, and user/session details. Sometimes the API is called, but wrong or missing data causes the backend to ignore the request.

3) Verify cart data from Get Cart API
After adding an item, I would see if the updated cart data is returned in the add-to-cart response or if the Get Cart API is called. The accurate cart item count should be included in the answer, and the extra product should be reflected in that count.

4) Match UI cart counter with API response
I'd compare the cart counter value shown on the UI with the cartItemCount coming from the API response. If the API has the correct count but the UI doesn't update, then the issue is clearly on the frontend side.

5) Validate API call sequence and timing
Before the UI updates, I would make sure the Add to Cart API is finished correctly. The cart counter may display outdated data if the user interface updates too soon or if the Get Cart API is called before the add request is completed.

6) Check error handling and failed API scenarios
I'd test what happens when the API fails. The cart counter should not update incorrectly, and the user should see a proper error message instead of a failure.

5. If the cart works in staging but fails in production, how would you approach debugging?

Debugging Approach When Cart Works in Staging but Fails in Production

1. **Review Production Logs and Errors**
   Analyze server and application logs to identify failures, exceptions, or timeout issues related to cart functionality.

2. **Inspect Network Requests and API Responses**
   Check API calls in the browser to confirm correct request payloads, response status codes, and data returned from the backend.

3. **Validate Production Configuration and Environment Variables**
   Ensure API endpoints, database connections, and third-party service configurations are correctly set in production.

4. **Confirm Deployment Consistency**
   Verify that the same application build and version running in staging is properly deployed in production.

5. **Check Caching and Client-Side Storage**
   Investigate browser cache, cookies, and local/session storage for stale or corrupted cart data.