Non-functional Testing

Non-Functional Testing During Development

Arthur Clarkson

4th September 2024

Error handling was outside of the functional specification, and is brought down to the developer to implement and test - meaning that it had to be non-functionally tested. I tried multiple UI components to show the exception returned from the server, such as an banners at the top of the holiday entry form, popups, and toast notifications.

From a usability standpoint, toast notifications align well with several key heuristics. They maintain visibility of system status by succinctly informing the user about errors or updates in real-time, without requiring a dedicated action (such as closing a popup). This non-intrusive design respects the user's workflow, reducing unnecessary disruption and cognitive load. Additionally, toast notifications adhere to established consistency and standards by appearing in a predictable corner of the screen and typically dismissing themselves after a short duration. As a result, they strike a balance between ensuring the message is noticed and allowing the user to continue working without being forced to acknowledge the notification.
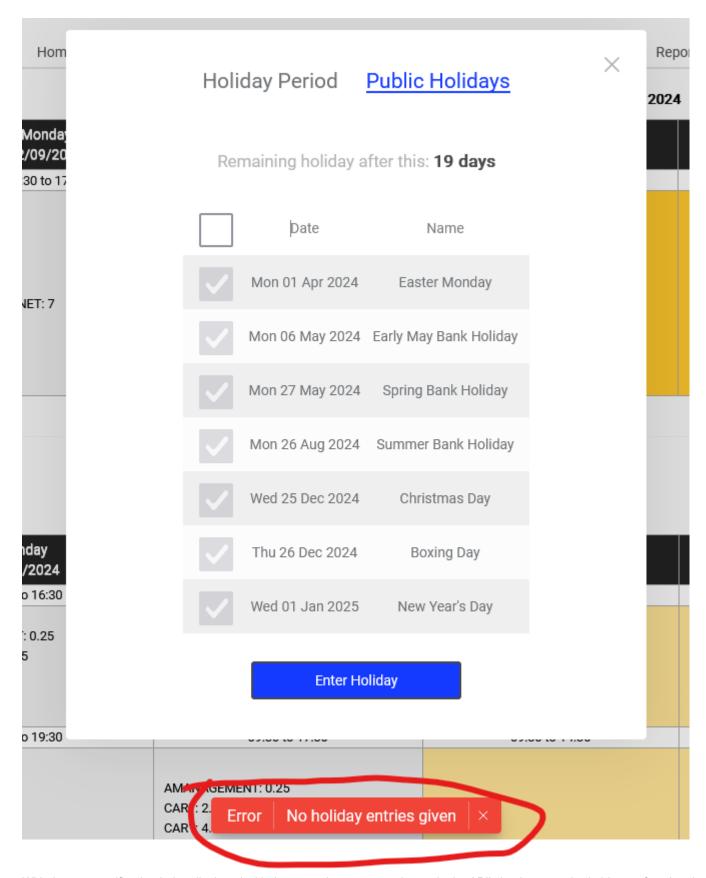
Toast notifications were the best pick.

```csharp
public void CreateHolidayEntriesForUser(List<HolidayEntryItem> holidayEntries, int userId)
{
        // Start of CreateHolidayEntriesForUser...

        bool noHolidayEntires = holidayEntries.Count == 0;
        if (noHolidayEntires)
        {
                throw new Exception("No holiday entries given");
        }

        // Rest of CreateHolidayEntriesForUser...
}
```

To test that the toast notifications are working, I pressed the "Enter Holiday" without any holidays selected:

With the toast notification being displayed with the exception message thrown in the API's business method, this non-functional test is considered a success.