Hackathon Day - 5

Bandage Marketplace Template
Testing, Error Handling, and Backend Integration Refinement

Objective:

Prepare the **"Bandage Marketplace"** for real-world deployment by thoroughly testing, optimizing performance, and refining backend integrations. Ensure readiness to handle customer-facing traffic while delivering a seamless and error-free user experience.

Key Objectives:

1. **Comprehensive Testing**:
    - Validate all features and backend integrations.
    - Complete functional, non-functional, and user acceptance testing.
2. **Error Handling**:
    - Implement user-friendly error messages and fallback mechanisms.
    - Gracefully handle API errors with fallback UI elements.
3. **Performance Optimization**:
    - Enhance speed, responsiveness, and reliability.
    - Conduct performance testing to identify and fix bottlenecks.
4. **Cross-Browser and Device Compatibility**:
    - Test for seamless functionality across multiple browsers and devices.
5. **Security Testing**:
    - Identify vulnerabilities and secure sensitive data.
6. **Documentation**:
    - Create professional testing reports (CSV-based) summarizing results and resolutions.
    - Prepare deployment-ready documentation with best practices.

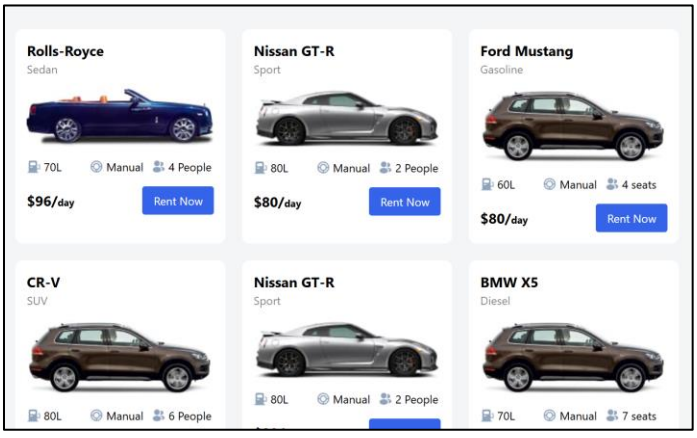
Key Learning Outcomes:

1. **Comprehensive Testing**:
    - Validate all features and backend integrations.
    - Ensure functional, non-functional, and user acceptance testing is completed.
2. **Error Handling**:
    - Implement user-friendly error messages and fallback mechanisms.
    - Gracefully handle API errors with fallback UI elements.
3. **Performance Optimization**:
    - Enhance speed, responsiveness, and reliability.
    - Conduct performance testing to identify and fix bottlenecks.
4. **Cross-Browser and Device Compatibility**:
    - Test for seamless functionality across multiple browsers and devices.
5. **Security Testing**:
    - Identify vulnerabilities and secure sensitive data.

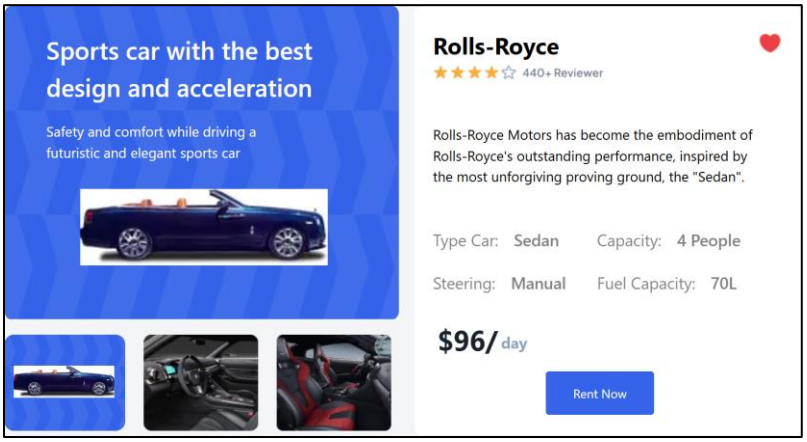
Functional Testing

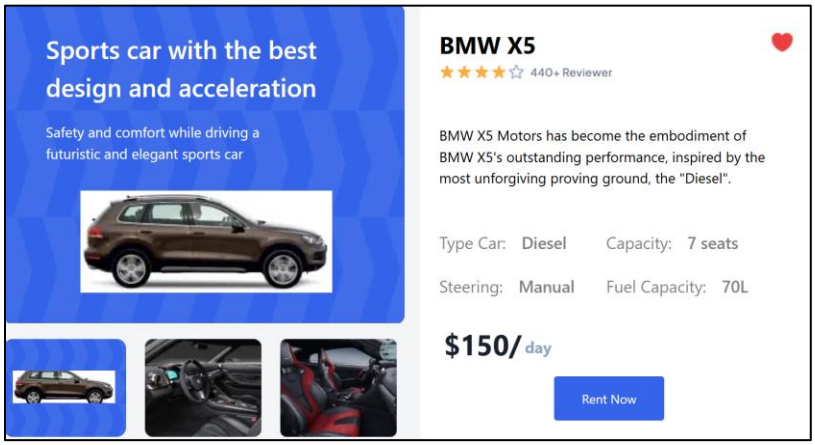
Key Features to Test:

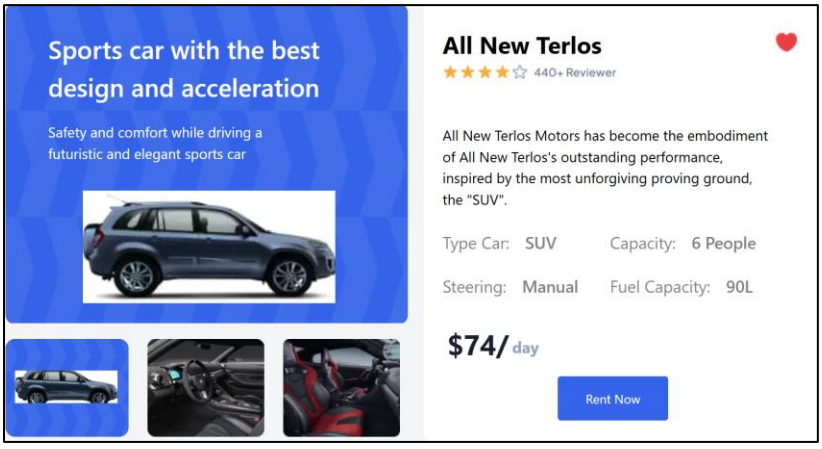
1. **Product Listing**: Verify products are displayed accurately with proper details.



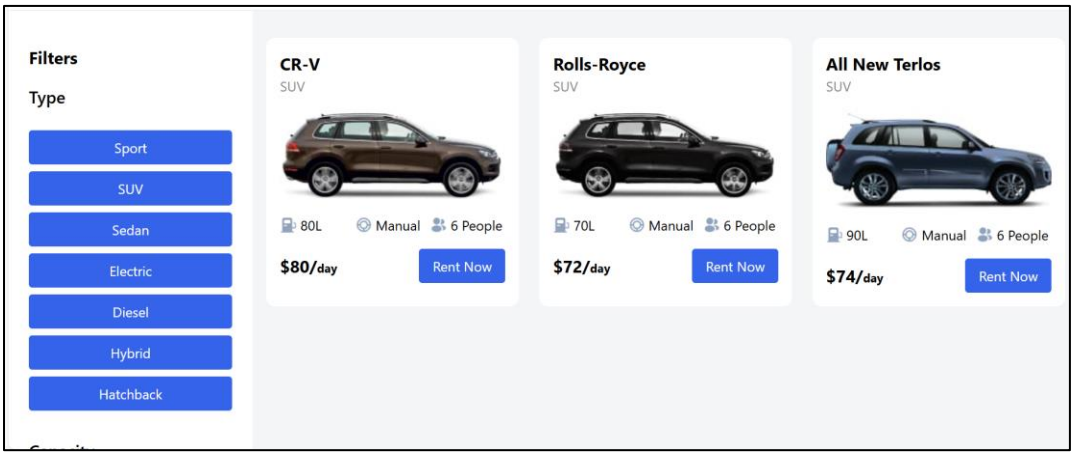
2. **Product Details**: Ensure product detail pages show correct information (e.g., price, description, images, and availability).

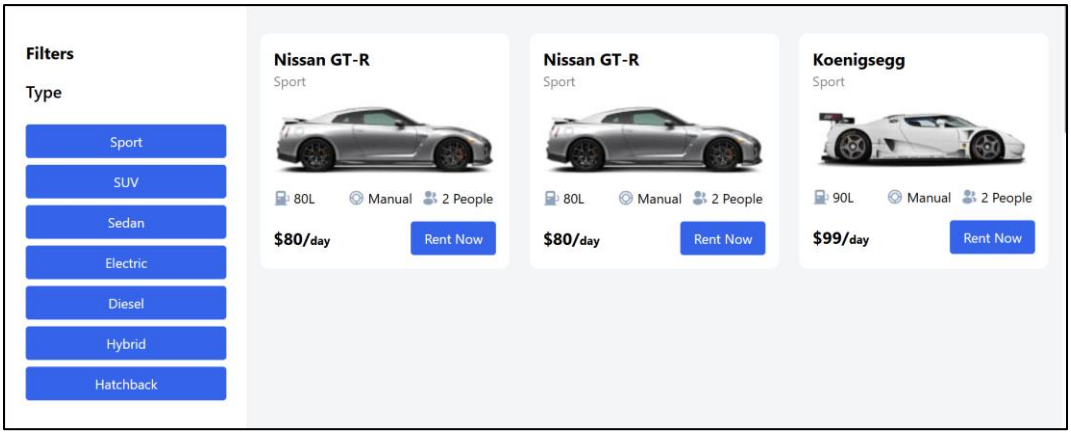


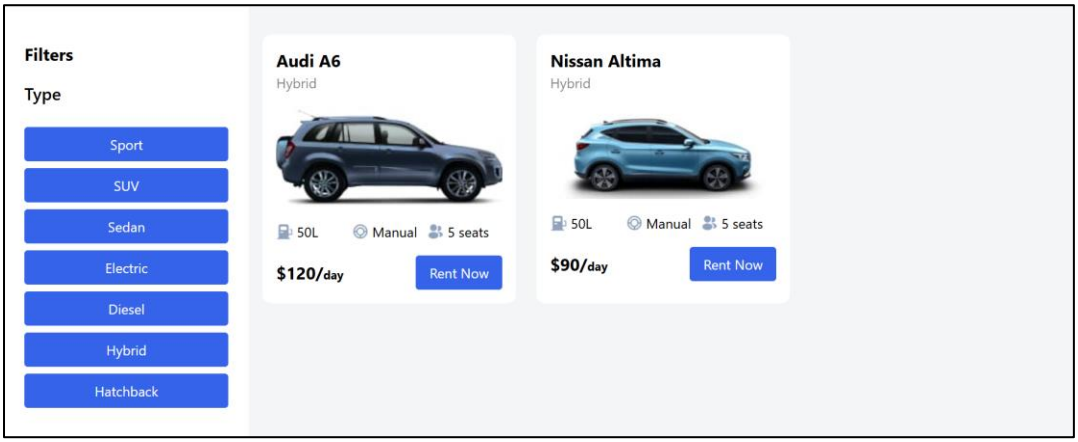




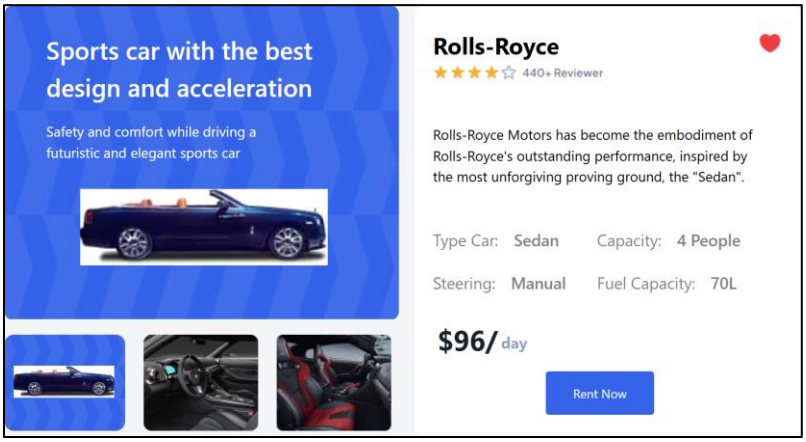
3. **Category Filters**: Test that filters return precise results based on user selection.

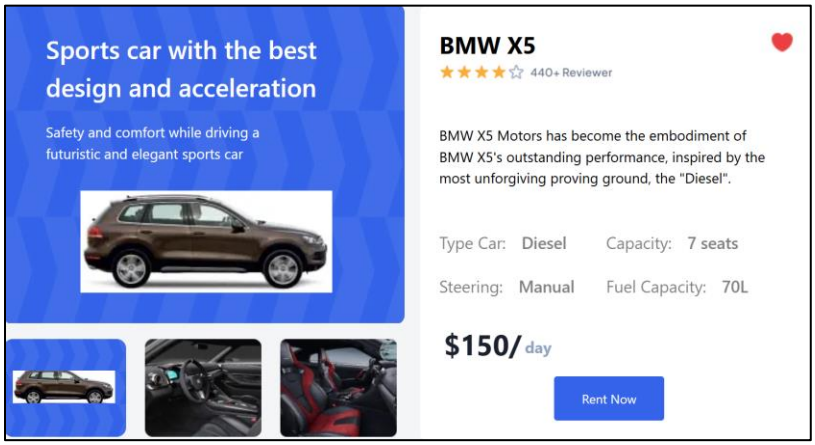


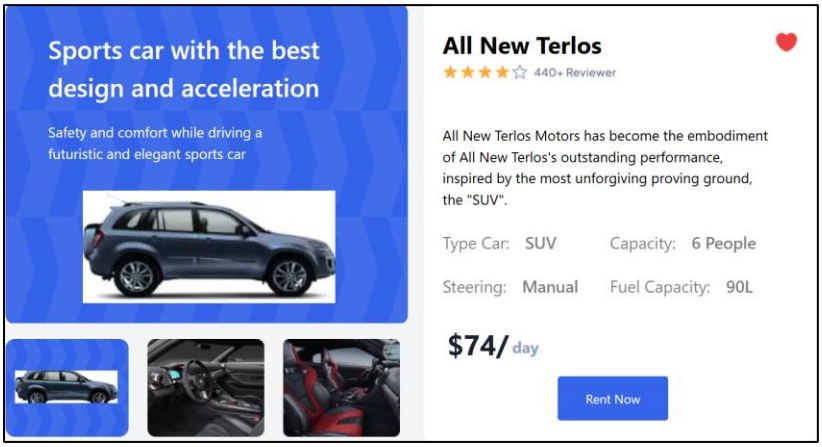




4. **Dynamic Routing**: Validate seamless navigation to individual product detail pages.

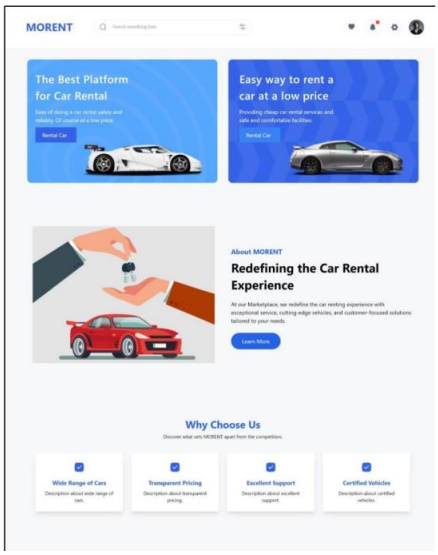






5. **Responsive Design**: Validate that all features and pages adapt seamlessly to various screen sizes (mobile, tablet, and desktop).

Desktop



Mobile

Error Handling

Implement robust error handling mechanisms to ensure a smooth user experience, even when issues arise, by displaying clear and user-friendly fallback messages.

Key Areas to Address in Error Handling:

1. **Network Failures**: Display a meaningful error message when there are connectivity issues.
    - Example: "Unable to connect to the server. Please check your internet connection."
2. **Invalid or Missing Data**: Handle cases where API responses return incomplete or invalid data.
    - Example: “Some information is missing. Please try again later”.
3. **Unexpected Server Errors**: Use generic fallback messages for unhandled server-side errors.
    - Example: "Something went wrong on our end. Please try again later."
4. **API Error Handling**: Use try-catch blocks to gracefully manage API errors.

5. **Fallback UI Elements**:
    - Provide alternative content when data is unavailable.
    - Example: Display a "No products available" message or a placeholder image when the product list is empty.
6. **Form Validation Errors**:
    - Validate user inputs on both the frontend and backend to avoid invalid data submissions.
    - Example: Show specific error messages like "Email is required" or "Invalid phone number format."

Some examples:

```
// Function to fetch car data by its ID
const fetchCarData = async (paymentid: number): Promise<Car | null> => {
  try {
    // Include the parameter directly in the query object
    const queryObject = {
      query: `*[_type == "car" && id == $id]`,
      params: { id: paymentid },
    };

    // Pass the object to sanityFetch
    const cars: Car[] = await sanityFetch(queryObject);

    console.log("Fetched cars:", cars);

    return cars[0] || null;
  } catch (error) {
    console.error("Error fetching car data:", error);
    return null;
  }
};
```

```
/ Dynamic CarPage component
onst CarPage = async ({ params }: { params: { payment: string } }) => {

 const paymentid = parseInt(params.payment, 10);

 if (isNaN(paymentid)) {
   return <div>Invalid car ID: {params.payment}</div>;
 }

 const car = await fetchCarData(paymentid);

 if (!car) {
   return <div>Car not found for ID: {paymentid}</div>;
 }
```

```
const imageUrl = car.image_url ? urlFor(car.image_url).url() : '/path/to/fallback-image.jpg';
```

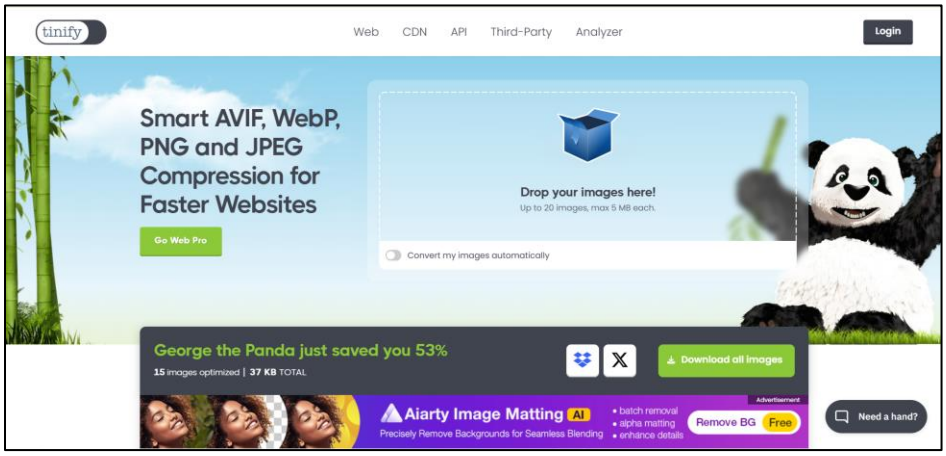
Performance Optimization

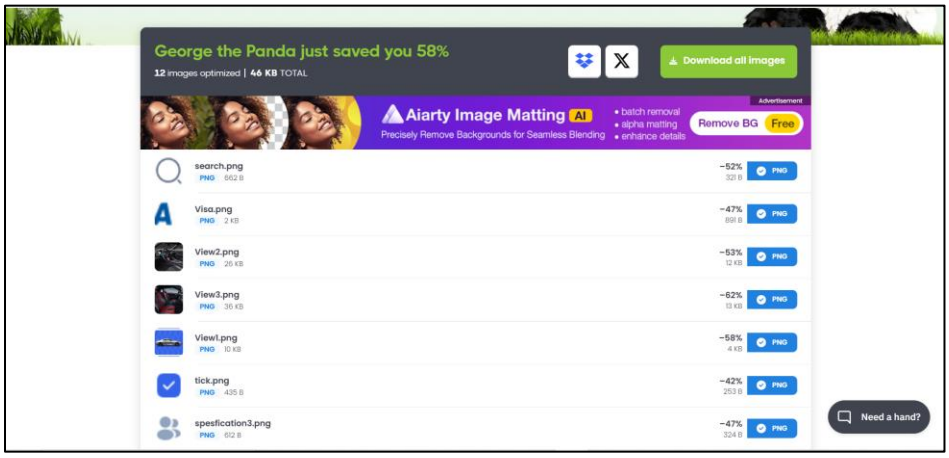
Optimize the marketplace to improve load times, responsiveness, and overall performance. This step involves identifying bottlenecks and applying strategies to enhance the user experience.

Key Areas to Address:

1. **Optimize Assets**:
    - **Image Optimization**: Compress images using tools like TinyPNG or ImageOptim to reduce size without losing quality.

      Example Of TinyPNG:





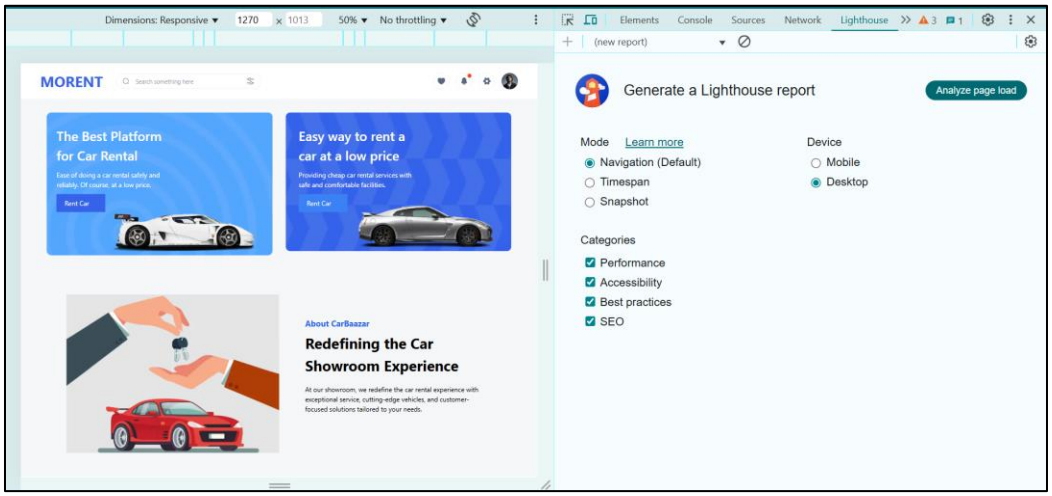
    - **Lazy Loading**: Implement lazy loading for images and assets to defer loading until needed, improving initial load times.

      Example: Using Next.js <Image> tag.

```
<Image src={car.image} alt={car.name} className="w-full h-auto mt-4" />
```

2. **Minimize JavaScript and CSS**:
    - **Minification**: Minify JavaScript (using Terser) and CSS (using CSSNano) to reduce file sizes for faster loading.
    - **Remove Unused Code**: Eliminate unused CSS and JavaScript to further reduce file sizes.
3. **Implement Caching Strategies**:
    - **Browser Caching**: Store static assets in the user's browser to avoid repeated network requests.
    - **Service Workers**: Cache resources and improve offline functionality using service workers.
4. **Analyze Performance**:
    - **Google Lighthouse**: Use Lighthouse to audit the website's performance and identify areas to improve.

      Analyzing Images: Available in Images folder.



    - **WebPageTest & GTmetrix**: Use these tools to identify performance bottlenecks and improve page load speeds.

Security Testing

Ensure the security of your marketplace by identifying vulnerabilities and implementing measures to protect sensitive data and prevent attacks.

Key Areas to Address:

1. **Input Validation**: Sanitize all user inputs to prevent SQL injection, cross-site scripting (XSS), and other injection attacks.
2. **Secure API Communication**: Ensure all API calls are made over HTTPS to encrypt data in transit.
3. Store Keys and Password in .env.local file.

```
$ .env.local
1  NEXT_PUBLIC_SANITY_PROJECT_ID="xyz"
2  NEXT_PUBLIC_SANITY_DATASET="production"
3  SANITY_API_TOKEN="abc"
```

4. **Authentication & Authorization**: Implement secure user authentication mechanisms like JWT (JSON Web Tokens) and OAuth.
5. **Cross-Site Request Forgery (CSRF) Protection**: Implement anti-CSRF tokens to prevent unauthorized actions.
6. **Security Headers**: Set HTTP security headers like Content-Security-Policy (CSP) to enhance security.
7. **Vulnerability Scanning**: Use tools like **OWASP ZAP** or **Burp Suite** to scan your marketplace for common security vulnerabilities.

User Acceptance Testing (UAT)

Objective:

Ensure that the marketplace meets user expectations and provides a seamless, intuitive experience before final deployment.

Key Areas to Address:

1. **Usability Testing**: Test user interface for intuitiveness and ease of navigation.
2. **Edge Cases**: Test edge cases like unusual search queries, and user interactions.
3. **Device and Browser Compatibility**: Confirm consistent user experience across devices (desktop, tablet, mobile) and browsers (Chrome, Firefox, Safari, Edge).

CSV Table Example:

Available in TestCase.pdf file.

Conclusion

Day 5 focused on preparing the marketplace for deployment by ensuring all components were thoroughly tested, optimized, and refined. Comprehensive functional, error handling, and performance tests were conducted, validating key features like product listing, cart operations, and dynamic routing. Robust error handling mechanisms and fallback UI elements were implemented to enhance reliability. Performance optimization efforts resulted in faster load times and improved responsiveness across devices and browsers. The team also documented all findings and resolutions in a professional format, ensuring readiness for real-world deployment.