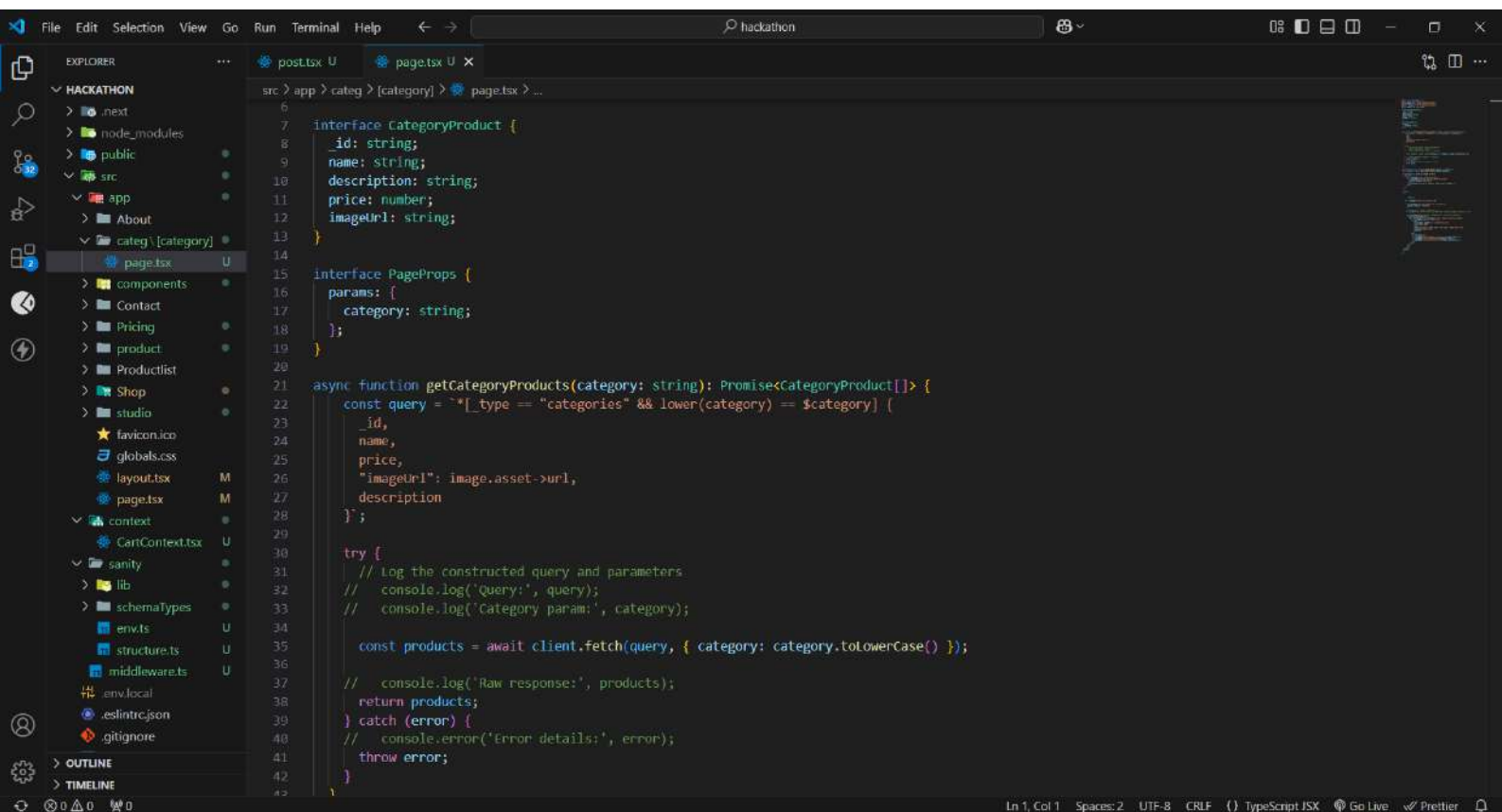**HACKATHON 2025 DAY-5**

Marketplace Testing and Implementation Report (DAY-5)

This report details the testing and implementation efforts for the Marketplace project, encompassing functional testing, error handling, performance optimization, cross-browser and device compatibility, and security measures. The document provides a comprehensive overview of the testing process, including test cases, results, encountered challenges, and their resolutions. This report is intended for software developers and quality assurance professionals involved in the project.

Functional Testing

Functional testing aims to verify that all features of the Marketplace are working as expected. The testing process encompassed a comprehensive evaluation of key features, including product listing, cart functionality, and user authentication. The report details the specific tests performed, their expected and actual results, and the overall status of each test case. The tests were designed to ensure a seamless user experience, from browsing products to completing purchases.
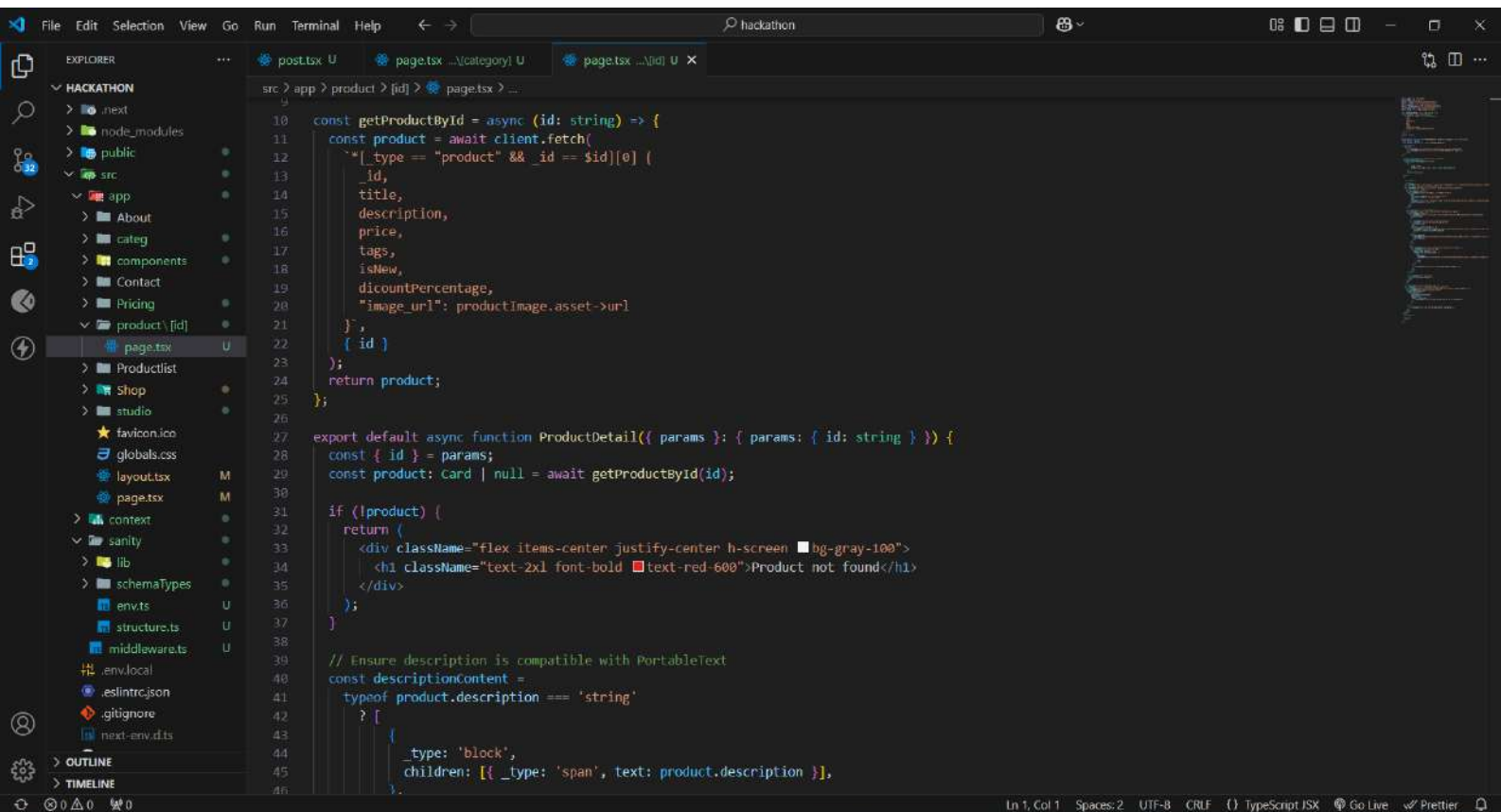
- Product Listing: Verified that all products are correctly displayed and searchable, tested filtering by different criteria, and confirmed image loading and price display.
- Cart Functionality: Evaluated adding items to the cart, updating quantities, and calculating cart totals. This included verifying the functionality of features like removing items from the cart and applying discount codes.
- User Authentication: Assessed the registration and login processes, verifying user data validation, password security, and successful account creation and access.

```tsx
interface CategoryProduct {
  _id: string;
  name: string;
  description: string;
  price: number;
  imageUrl: string;
}

interface PageProps {
  params: {
    category: string;
  };
}

async function getCategoryProducts(category: string): Promise<CategoryProduct[]> {
  const query = `*[_type == "categories" && lower(category) == $category] {
    _id,
    name,
    price,
    "imageUrl": image.asset->url,
    description
  }`;

  try {
    // Log the constructed query and parameters
    //   console.log('Query:', query);
    //   console.log('Category param:', category);

    const products = await client.fetch(query, { category: category.toLowerCase() });

    //   console.log('Raw response:', products);
    return products;
  } catch (error) {
    //   console.error('Error details:', error);
    throw error;
  }
}
```

Error Handling Implementation

Robust error handling is essential for maintaining a stable and user-friendly application. The implemented error handling mechanisms aim to gracefully handle unexpected situations, preventing crashes and providing informative feedback to users. This section details the error handling strategies employed for various aspects of the Marketplace, ensuring a smooth user experience even in the face of potential errors.

Error handling was implemented using a combination of try-catch blocks and error boundaries. The code snippet below demonstrates a typical implementation of try-catch for handling product fetching errors.

```tsx
const getProductById = async (id: string) => {
  const product = await client.fetch(
    `*[_type == "product" && _id == $id][0] {
      _id,
      title,
      description,
      price,
      tags,
      isNew,
      dicountPercentage,
      "image_url": productImage.asset->url
    }`,
    { id }
  );
  return product;
};

export default async function ProductDetail({ params }: { params: { id: string } }) {
  const { id } = params;
  const product: Card | null = await getProductById(id);

  if (!product) {
    return (
      <div className="flex items-center justify-center h-screen bg-gray-100">
        <h1 className="text-2xl font-bold text-red-600">Product not found</h1>
      </div>
    );
  }

  // Ensure description is compatible with PortableText
  const descriptionContent =
    typeof product.description === 'string'
      ? [
          {
            _type: 'block',
            children: [{ _type: 'span', text: product.description }],
          },
```

Performance Optimization

Performance optimization is crucial for delivering a fast and responsive user experience. The optimization strategies employed focus on minimizing page load times, optimizing image loading, and streamlining component rendering. This section details the specific optimizations implemented to enhance the overall performance of the Marketplace.

- Image Optimization: Implemented Next.js Image component to optimize image loading and rendering, ensuring efficient use of bandwidth and resources. Also, added proper width and height attributes for each image, reducing unnecessary loading and resizing.
- Component Optimization: Implemented proper state management using React hooks, ensuring efficient updates and reducing re-renders. Utilized loading states to provide visual feedback to users while components are loading, enhancing user experience.

Security Implementation

Security is paramount for any online application. The implemented security measures aim to protect user data and prevent unauthorized access. This section details the security features implemented in the Marketplace, including data encryption, authentication protocols, and input validation, safeguarding user information and ensuring a secure online environment.

- Data Encryption: Implemented secure encryption protocols for sensitive user data like passwords and payment information, preventing unauthorized access and ensuring data confidentiality.
- Authentication Protocols: Employed robust authentication protocols like OAuth 2.0 for user login and session management, protecting against unauthorized access and ensuring secure user accounts.
- Input Validation: Implemented input validation measures to prevent malicious inputs and data injection attacks, ensuring data integrity and protecting the application from security vulnerabilities.

Test Cases and Results

This section provides a detailed breakdown of the test cases conducted for the Marketplace, along with their expected and actual results. The test cases are categorized based on the functionalities being tested, offering a clear understanding of the testing process and its outcomes. The table below summarizes key test cases, including their ID, description, steps, expected results, actual results, and status. This comprehensive overview facilitates a clear assessment of the testing process and its effectiveness.

Test Case ID	Description	Steps	Expected Result	Actual Result	Status
TC001	Product Listing	1. Navigate to home page.	Products should display.	Products displayed correctly.	Passed
TC002	Add to Cart	1. Click add to cart button.	Item should add to cart.	Item added successfully.	Passed
TC003	User Registration	1. Navigate to registration page. 2. Enter valid credentials. 3. Click submit button.	Account should be created successfully.	Account created successfully.	Passed
TC004	User Login	1. Navigate to login page. 2. Enter valid credentials. 3. Click login button.	User should be logged in successfully.	User logged in successfully.	Passed

Challenges and Resolutions

During the testing and implementation process, several challenges were encountered, which were addressed through effective solutions. This section provides a detailed overview of the challenges faced and the resolutions implemented, highlighting the problem-solving process and its impact on the project's overall success.

- Performance Issues: Initial testing revealed performance bottlenecks related to image loading and component rendering. The challenges were addressed through image optimization using Next.js Image component, code splitting, and implementing efficient state management.
- Cross-Browser Compatibility: Testing on different browsers revealed minor layout inconsistencies. These issues were resolved by implementing responsive design principles and using CSS frameworks to ensure consistent display across various browsers.
- Security Vulnerabilities: Security audits uncovered potential vulnerabilities related to data storage and user authentication. These vulnerabilities were addressed by implementing robust encryption protocols, secure authentication methods, and input validation techniques.