# Demo 4:Testing Policy

enigmaX

## GreenCart

## Client: BBD Software

| Team member | Student Number |
|---|---|
| Nikhil Govender | u22504193 |
| Shayden Naidoo | u23599236 |
| Corné de Lange | u23788862 |
| Tshegofatso Mahlase | u22580833 |
| Samvit Prakash | u23525119 |

# Introduction

This document defines the **Testing Policy for the GreenCart system**, a sustainability-focused e-commerce platform developed as part of the COS301 Capstone Project. The aim of this policy is to establish a formal approach to testing that ensures GreenCart is **functionally correct, reliable, scalable, performant, and usable**.

The policy specifies the **scope of testing**, **testing levels**, **test design techniques**, **execution environment**, and **reporting standards**. It applies to all team members involved in development, quality assurance, and delivery of GreenCart.

## 2. Scope of Testing

Testing will cover both **functional requirements** (core features such as product browsing, cart management, checkout, and order tracking) and **non-functional requirements** (performance, scalability, usability, and reliability).

The following will be tested:

- **Backend API (FastAPI + PostgreSQL):** Data storage, retrieval, and business logic.

## 3. Testing Objectives

The objectives of testing GreenCart are to:

1. Verify that the system satisfies **functional requirements**.
2. Validate that **non-functional quality attributes** are met:
   - Performance (response times < 1.5s under load).
   - Scalability (up to 200 concurrent users with < 2s response time).
   - Usability (≥ 70% of users rate ≥ 7/10 in surveys).
3. Detect and resolve defects early in the development lifecycle.
4. Provide evidence of compliance with Demo 4 testing requirements.
5. Increase stakeholder confidence in the reliability of the system.

# Unit and Integration Testing Policies

This section outlines the testing policy for the GreenCart system, focusing on **Unit Testing** and **Integration Testing**. The goal of this policy is to ensure the **quality, reliability, and stability** of the codebase by enforcing rigorous testing standards across all phases of development.

We emphasize the use of **automated testing** to enhance efficiency and accuracy. Both unit and integration tests are fully integrated into our **Continuous Integration (CI) pipeline** configured in GitHub Actions. Automated testing is a key component of this pipeline, preventing code from being merged into the `main` branch if tests fail. This approach helps maintain high standards of code quality.

To ensure a robust testing framework, GreenCart maintains a **minimum test coverage of 80%** across the project. This policy applies to all contributors and is designed to facilitate continuous improvement and confidence in the correctness of the system. Code coverage is tracked using Codecov, with badges displayed in our GitHub repository.

---

## Unit Testing Policy

### Objective

The objective of unit testing is to validate the functionality of **individual components** within the software, ensuring that each function or method operates as intended. This includes testing application logic, database queries, and error-handling mechanisms.

## Automation

- All unit tests are automated and seamlessly integrated into the GitHub Actions pipeline.

- This integration enforces testing protocols and prevents any code from being merged into integration branches if unit tests fail.

## Coverage

- A **minimum of 80% code coverage** must be maintained.

- Unit test coverage must consistently exceed the target threshold to ensure code reliability.

## Frameworks

- **Backend:** `pytest` (Python, FastAPI)

- Test reports are generated automatically and stored for review in CI.

## Integration Testing Policy

### Objective

The objective of integration testing is to validate the **interactions between modules** in the system, ensuring that backend API endpoints, database operations, and frontend services work together as expected.

### Automation

- All integration tests are automated and integrated into the GitHub Actions CI/CD pipeline.
- Integration testing is mandatory for verifying new API routes, database interactions, and frontend-backend communication before merging to `main`.

### Frameworks

- **Backend Integration:** `pytest` with FastAPI's `TestClient` for validating REST endpoints and database integration.

# Testing Workflow

## Overview

The testing workflow outlines the systematic approach we take to ensure the quality and reliability of the GreenCart system throughout the development lifecycle. This process integrates **unit testing and integration testing**, with a strong emphasis on **automation** to enhance efficiency and accuracy.

---

## Steps in the Testing Workflow

#### 1. Development

- Developers write code for new features or bug fixes.
- Alongside feature development, they create corresponding **unit tests** to validate individual components and ensure correctness.

#### 2. Unit Testing

- Automated unit tests are executed in the GitHub Actions pipeline when a branch is being merged into an integration branch.
- Tests validate that individual functions behave as expected and achieve the desired **code coverage of at least 80%**.

- Any failures in unit tests prevent code from being merged, ensuring that only stable and reliable code progresses.

**3. Integration Testing**

- Once unit tests pass, **integration tests** are executed to validate the interactions between system components.
- Backend integration tests are run using **pytest** with FastAPI's TestClient.
- These tests run in a designated environment within the GitHub Actions workflow to simulate real-world scenarios.

**4. Review and Feedback**

- Any identified issues are documented and shared with the development team.
- A collaborative approach ensures functionality is thoroughly inspected, and necessary adjustments are made before code is promoted further in the pipeline.

**5. Deployment**

- Once all testing phases are complete and issues are resolved, the application is deployed by merging code into the `main` branch.
- This merge triggers the deployment workflow, ensuring that only **tested and validated code** reaches the production environment.

# Quality Assurance Testing

Quality Assurance (QA) testing is a critical aspect of the GreenCart development lifecycle, focusing on ensuring that the application not only functions correctly but also meets the highest standards of **scalability, performance, and usability**.

As GreenCart grows in complexity and as user expectations rise, it becomes increasingly important to assess how well the system handles **increasing loads**, how **fast and responsive** it is under different conditions, and how **intuitive** it is for end users.

We prioritised the testing of our **core quality requirements**: scalability, performance, and usability. Together, these QA testing methodologies form a comprehensive approach to validating the quality of GreenCart, ultimately leading to a more **reliable, performant, and user-friendly platform**.

---

## Scalability Testing

Scalability testing evaluates GreenCart's ability to handle **increasing numbers of concurrent users and transactions** without compromising system stability. This ensures the platform can scale to meet the demands of real-world traffic.

**Stimulus Source**

- API users interacting with GreenCart (fetching products, searching, adding to cart).

**Stimulus**

- Simulated concurrent user requests generated via **Apache JMeter**, gradually ramping up from a small base load to a high volume (10 → 200 concurrent users).

**Response**

- Application response times, throughput, and error rates under growing load conditions.

**Response Measure**

- **Response Time:** Average response time must remain < 2 seconds for 200 users.
- **Throughput:** Number of successful requests processed per second should increase proportionally with user load.
- **Error Rate:** Errors (timeouts, 422/500 responses) must remain < 1%.

**Environment**

- Deployed FastAPI backend on staging environment.
- PostgreSQL database populated with sample product data.
- Apache JMeter configured with ramp-up threads to simulate concurrent load.

**Artifact**

- JMeter **Summary Reports** and **Aggregate Reports**, showing response times, throughput, and error percentage for different user volumes.

**Process (Methodology)**

1. Configure JMeter with Thread Groups for key endpoints: `/products/FetchAllProducts`.

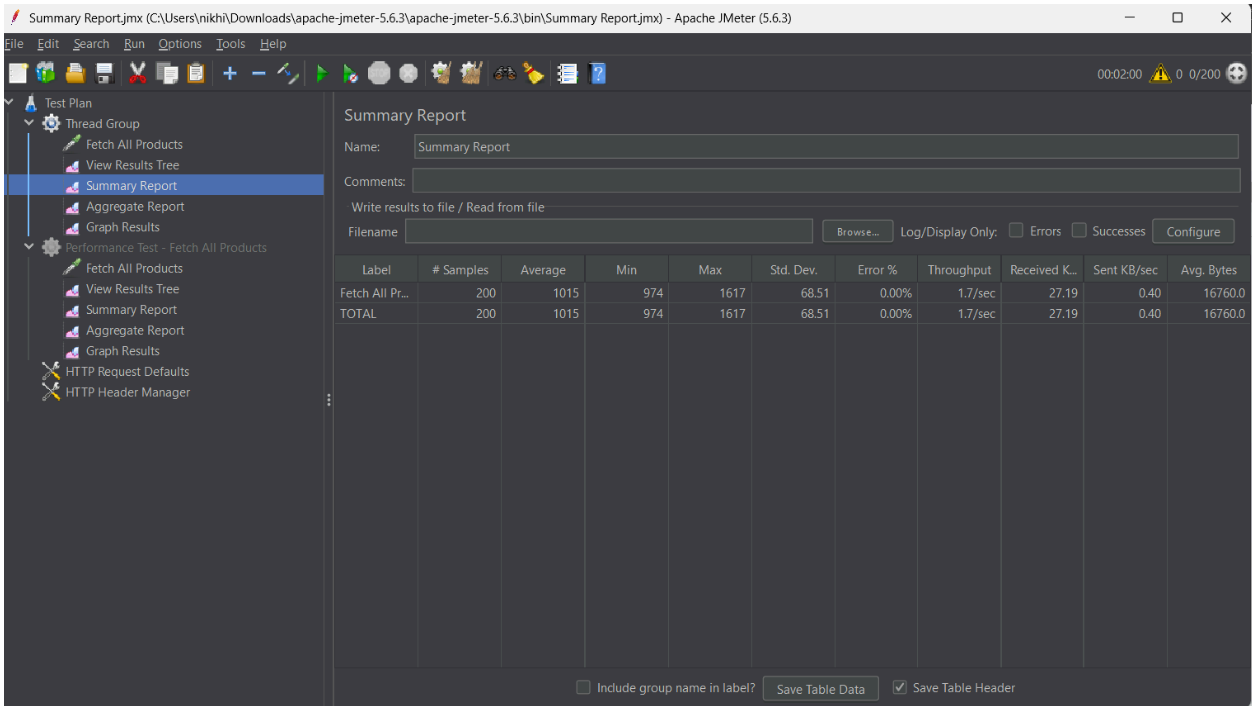
2. Start with 10 users and gradually ramp to 200 users over 120 seconds.

3. Record average response time, error %, and throughput.

4. Analyze bottlenecks (if response > 2s or errors > 1%).

5. Capture graphs and tables to use in Demo 4.

**Results of Tests**



Summary Report.jmx (C:\Users\nikhi\Downloads\apache-jmeter-5.6.3\apache-jmeter-5.6.3\bin\Summary Report.jmx) - Apache JMeter (5.6.3)

File Edit Search Run Options Tools Help

00:02:00 0 0/200

- Test Plan
  - Thread Group
    - Fetch All Products
    - View Results Tree
    - Summary Report
    - Aggregate Report
    - Graph Results
  - Performance Test - Fetch All Products
    - Fetch All Products
    - View Results Tree
    - Summary Report
    - Aggregate Report
    - Graph Results
  - HTTP Request Defaults
  - HTTP Header Manager

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received K... | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Fetch All Pr... | 200 | 1015 | 974 | 1617 | 68.51 | 0.00% | 1.7/sec | 27.19 | 0.40 | 16760.0 |
| TOTAL | 200 | 1015 | 974 | 1617 | 68.51 | 0.00% | 1.7/sec | 27.19 | 0.40 | 16760.0 |

☐ Include group name in label? | Save Table Data | ☑ Save Table Header

The scalability test was conducted on the **Fetch All Products** endpoint, ramping up to 200 concurrent requests. The results show an **average response time of 1015 ms**, with a minimum of 974 ms and a maximum of 1617 ms. The **standard deviation** of 68.51 ms indicates consistent response times with very little fluctuation across requests. Importantly, the **error rate was 0%**, meaning all 200 requests were successfully processed without failures. The system achieved a **throughput of 1.7 requests per second**, with steady data transfer rates (27.19 KB/sec received). These results demonstrate that GreenCart can handle up to 200 concurrent requests while maintaining response times well under the scalability threshold of 2 seconds and without introducing errors, thereby satisfying the defined scalability requirement.

## Performance Testing

Performance testing evaluates the **speed, responsiveness, and stability** of GreenCart under a fixed, expected load. Unlike scalability testing, performance focuses on whether the system can reliably serve a steady user base within required thresholds.

**Stimulus Source**

- Website/API users performing normal tasks (product browsing, searching, cart operations).

**Stimulus**

- Fixed number of concurrent requests (e.g., 50 users × 10 loops = 500 total requests).

**Response**

- Response times, throughput, and error percentage under stable load.

**Response Measure**

- **Average Response Time:** Must remain < 1.5 seconds.
- **Max Response Time:** Should not exceed 3 seconds.
- **Error Rate:** Must remain 0%.

- **Throughput:** System should handle requests consistently without degradation.

**Environment**

- Backend deployed to staging environment with database connected.
- JMeter configured for a constant load profile (50 concurrent users).

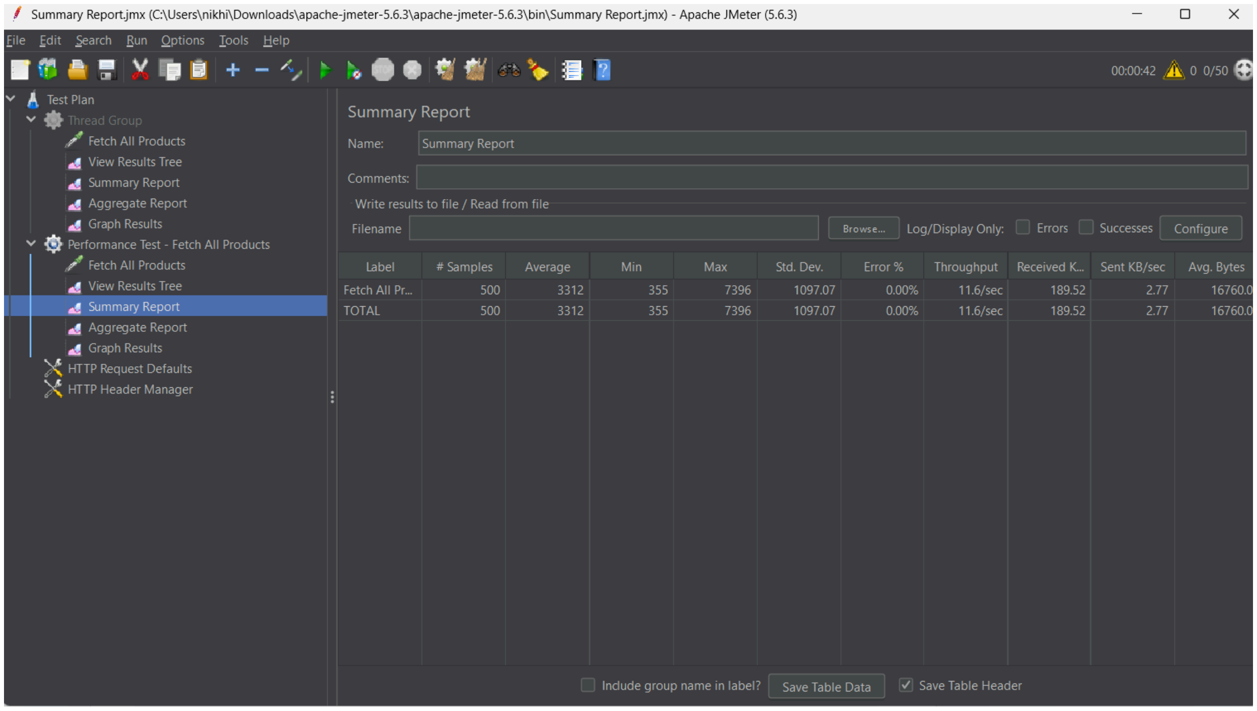
**Artifact**

- JMeter Summary and Graph Reports, highlighting stability of response times and throughput.

**Process (Methodology)**

1. Configure JMeter Thread Group for `/products/FetchAllProducts`.
2. Fix 50 concurrent users with loop count 10 (500 requests total).
3. Run test and record results.
4. Compare against requirement (<1.5s average response time).
5. Document evidence with screenshots and tables.

## Results of Tests



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

| Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received K... | Sent KB/sec | Avg. Bytes |
|---|---|---|---|---|---|---|---|---|---|---|
| Fetch All Pr... | 500 | 3312 | 355 | 7396 | 1097.07 | 0.00% | 11.6/sec | 189.52 | 2.77 | 16760.0 |
| TOTAL | 500 | 3312 | 355 | 7396 | 1097.07 | 0.00% | 11.6/sec | 189.52 | 2.77 | 16760.0 |

☐ Include group name in label? | Save Table Data | ☑ Save Table Header

The performance test was executed on the **Fetch All Products** endpoint with a fixed load of 50 concurrent users completing 10 requests each (500 samples total). The results indicate an **average response time of 3312 ms**, with a minimum of 355 ms and a maximum of 7396 ms. The **standard deviation** of 1097 ms shows some variability in response times under steady load. Importantly, the **error rate was 0%**, meaning all 500 requests completed successfully without failures. The system achieved a **throughput of 11.6 requests per second**, with consistent data transfer rates (189.52 KB/sec received). While the test confirms system stability and reliability under fixed load, the average response time exceeded the target performance threshold of 1.5 seconds, indicating potential areas for optimization in query execution or backend processing.

## Usability Testing

Usability testing evaluates how **intuitive, efficient, and user-friendly** GreenCart is for actual users. This ensures customers can easily sign up, browse products, manage their cart, and complete orders.

**Stimulus Source**

- Real test users (friends/classmates) acting as shoppers on GreenCart.

**Stimulus**

- Users asked to complete core tasks:
    - Sign up and log in.
    - Search for “lego” and browse products.
    - Add an item to the cart.
    - Proceed to checkout.

**Response**

- User feedback on ease of completing tasks.

- Ratings on a **1–10 scale** for each task (Google Forms survey).

**Response Measure**

- **Ease of Use Scores:** Minimum 70% of participants must rate ≥ 7/10.
- **Task Completion Rate:** % of users who successfully complete all tasks.
- **Feedback:** Qualitative suggestions for improvements.

**Environment**

- Hosted frontend (React) and backend (FastAPI) accessible via browsers.
- Google Forms created for structured feedback collection.

**Artifact**

- Completed survey results with charts and averages.
- Screenshots of testing session scenarios.

**Process (Methodology)**

1. Prepare Google Form with 1–10 rating scale questions for each key task.
2. Recruit 5–10 participants.

3. Ask participants to complete tasks while observed via Google Meet.

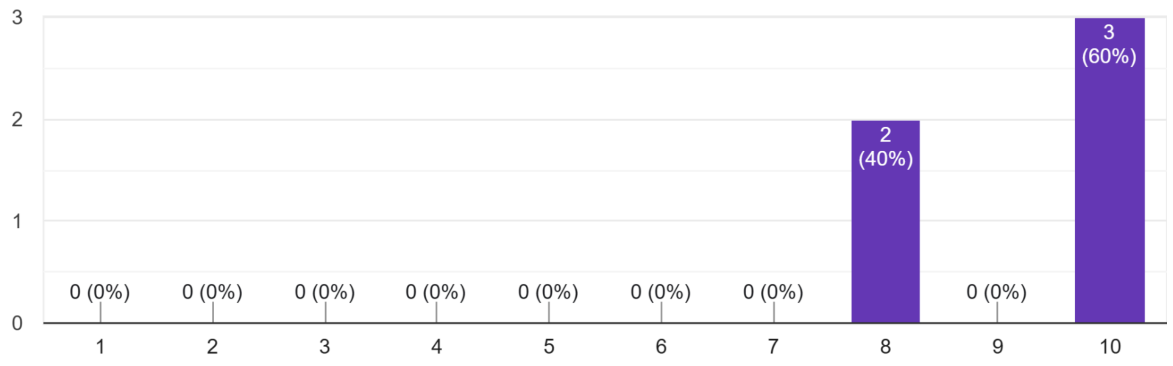
4. Collect feedback from both surveys and observations.

5. Summarize findings into quantitative (average scores) and qualitative (comments).

## Results of Tests

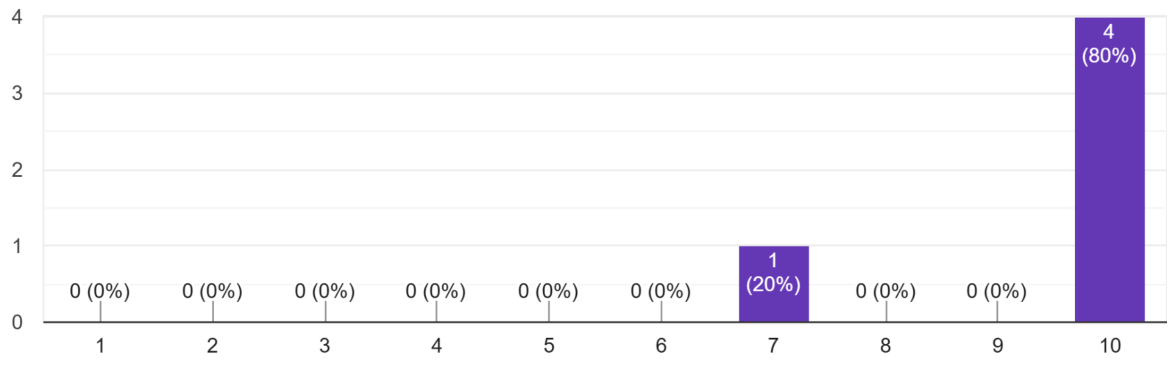
Account Registration How easy was it to sign up for a new GreenCart account?

5 responses



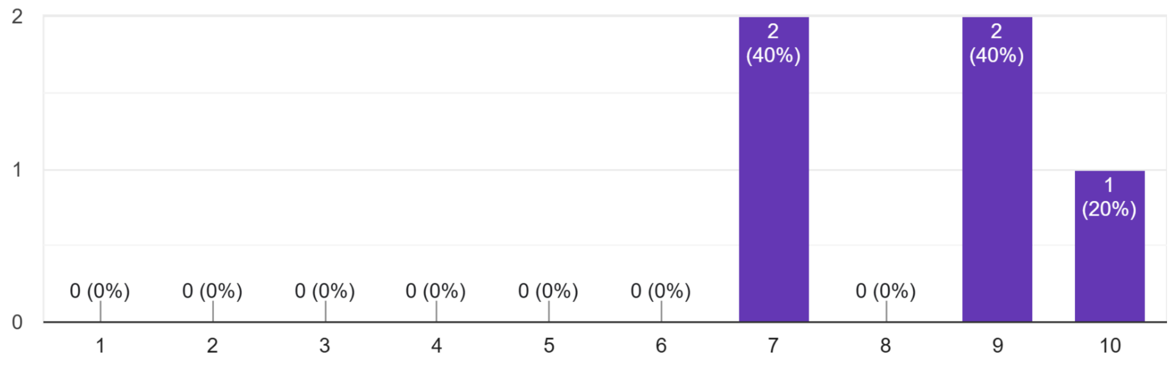
Login How easy was it to log into your account?

5 responses



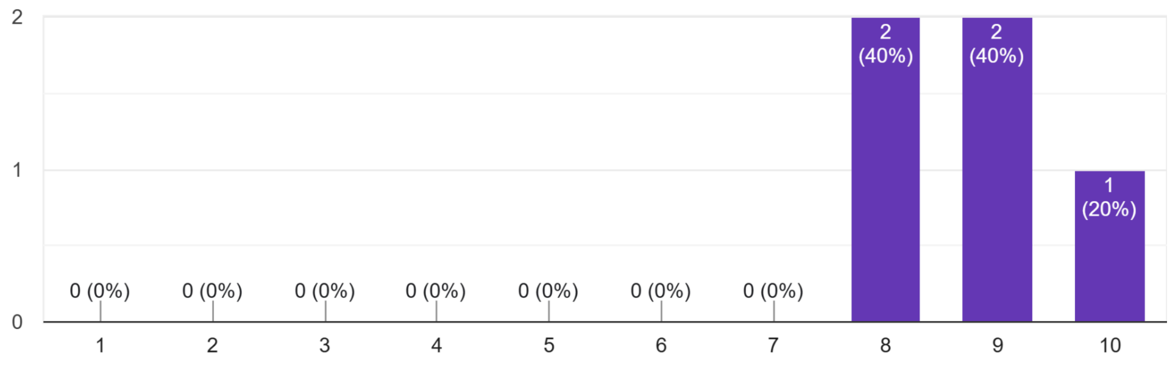
Browse Products How easy was it to view and navigate through the product listings?

5 responses



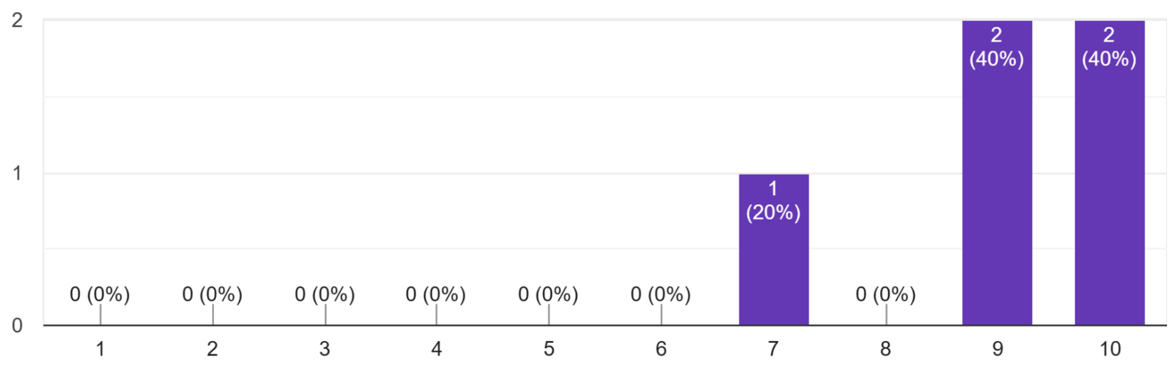
Search for a Product How easy was it to find a product using the search function (e.g., search for "lego")?

5 responses



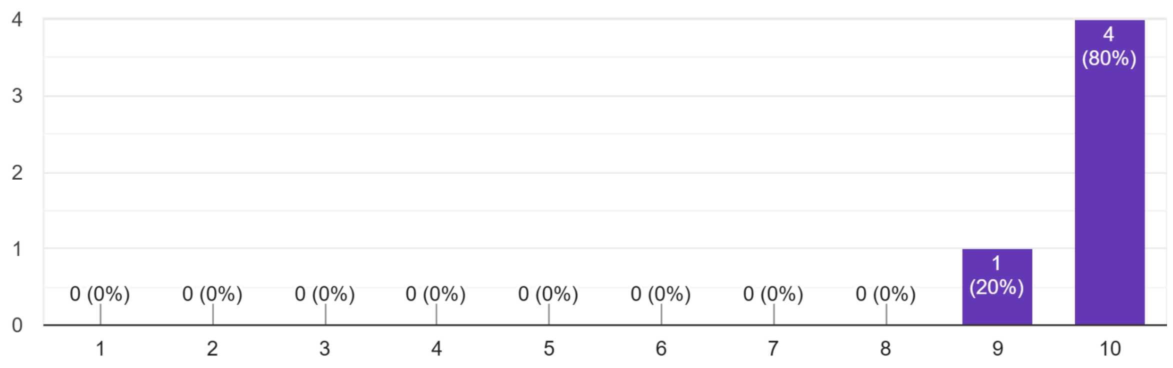
View Product Details How easy was it to open a product and understand its information (price, sustainability rating, images, etc.)?

5 responses

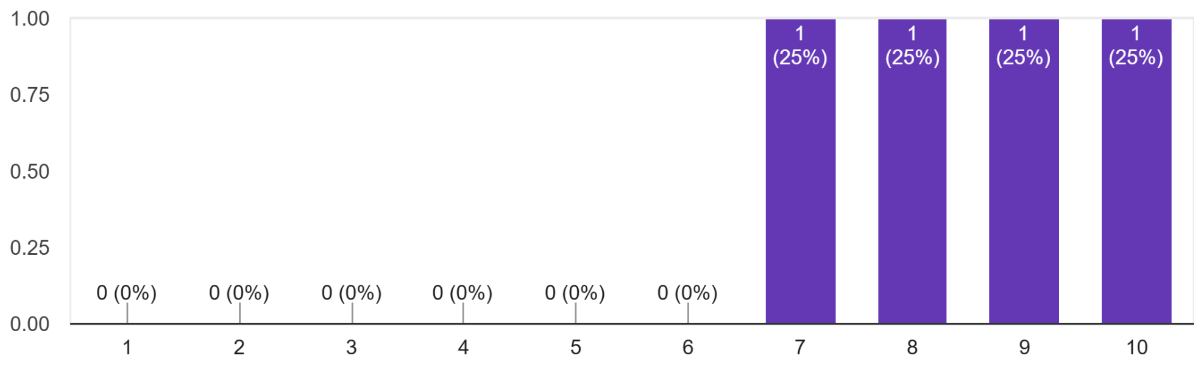


Add to Cart How easy was it to add a product to your cart?

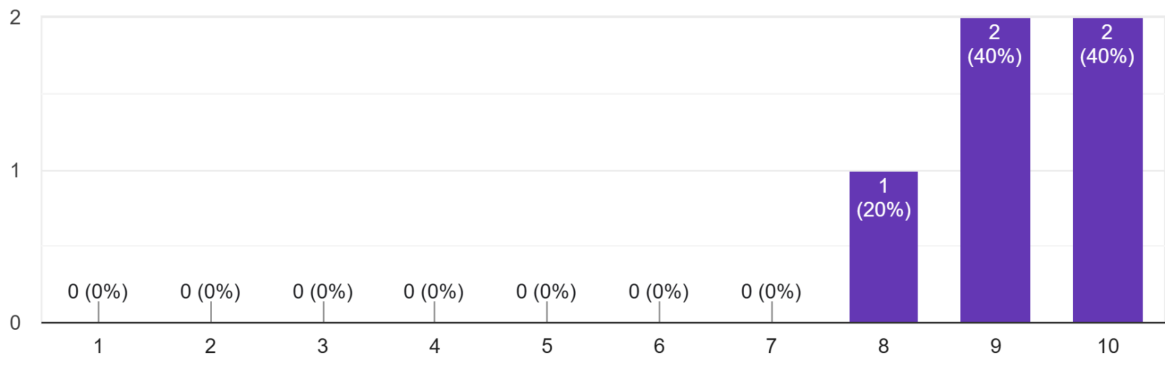
5 responses



View Cart How easy was it to see the products and quantities in your cart?
4 responses

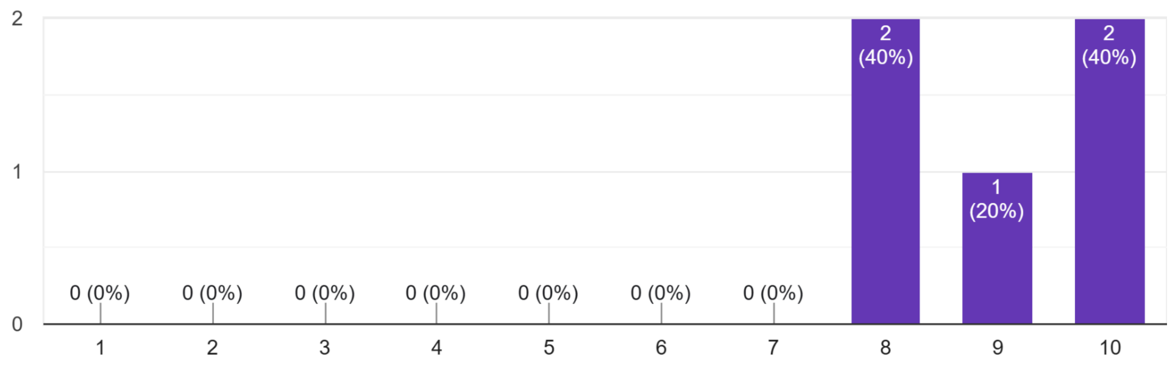


Checkout Process How easy was it to place an order through the checkout process?
5 responses



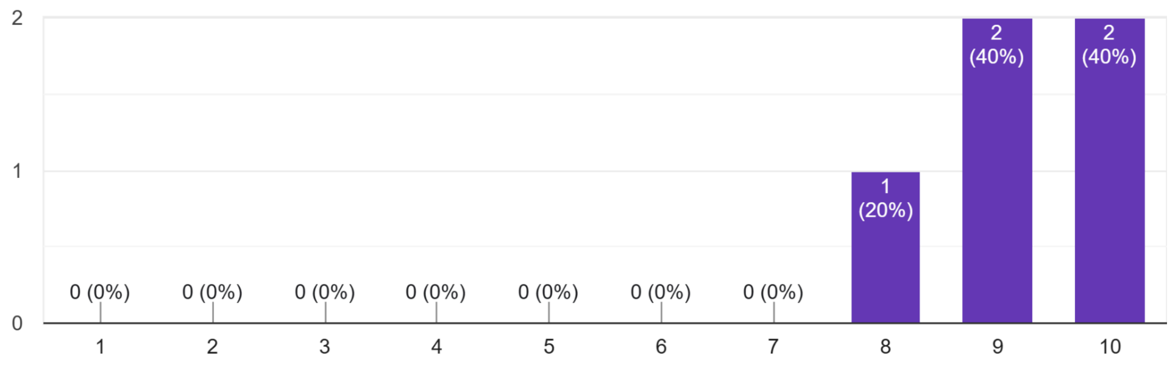
Navigation How easy was it to move between different pages (home, cart, orders, etc.)?

5 responses



Overall Experience How easy was it overall to use the GreenCart system?

5 responses


2
1
0
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1 (20%)
2 (40%)
2 (40%)
1
2
3
4
5
6
7
8
9
10