

---

# Testing Document

for  
**store.it**

Version 0.2

Prepared by

**Group #: 3**

**Group Name:** localhosts

<b>Name</b>	<b>Roll No.</b>	<b>Email</b>
Akanksha Singh	200070	akankshas20@iitk.ac.in
Antreev Singh Brar	190163	antreev@iitk.ac.in
Bhuvan Singla	180199	bhuvans@iitk.ac.in
Deepankur Kansal	180226	deepank@iitk.ac.in
Dipanshu Garg	190306	dipanshu@iitk.ac.in
Harshit Raj	200433	harshitr20@iitk.ac.in
Hitesh Anand	200449	ahitesh20@iitk.ac.in
Manas Gupta	200554	manasg20@iitk.ac.in
Priya Gole	200727	priyagole20@iitk.ac.in
Tushar	190915	tusharb@iitk.ac.in

**Course:** CS253A

**Mentor TA:** Mr. Swastik Maiti

**Date:** 3rd April, 2022

## Content

<b>CONTENTS</b>	<b>II</b>
<b>REVISIONS</b>	<b>II</b>
<b>1 INTRODUCTION</b>	<b>1</b>
<b>2 UNIT TESTING</b>	<b>2</b>
<b>3 INTEGRATION TESTING</b>	<b>5</b>
<b>4 FRONTEND TESTING</b>	<b>13</b>
<b>4 SYSTEM TESTING</b>	<b>14</b>
<b>5 CONCLUTION</b>	<b>20</b>
<b>APPENDIX A - GROUP LOG</b>	<b>21</b>

## Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.1	Akanksha Singh Antreev Singh Brar Bhuvan Singla Deepankur Kansal Dipanshu Garg Harshit Raj Hitesh Anand Manas Gupta Priya Gole Tushar	First draft	03/04/2022

Version	Primary Author(s)	Description of Version	Date Completed
0.2	Akanksha Singh Antreev Singh Brar Bhuvan Singla Deepankur Kansal Dipanshu Garg Harshit Raj Hitesh Anand Manas Gupta Priya Gole Tushar	Final draft	27/04/2022

# 1 Introduction

**Testing Strategy:** Unit tests and System tests are automated whereas Integration Tests are implemented manually.

**Testing Timeline:** The testing was conducted after the implementation was completed.

**Testers:** The testing process was done by the developers themselves.

**Coverage Criteria:** We used statement coverage criteria and branch coverage criteria for unit testing.

**Tools Used:**

- Unit Testing : Junit ( A library of Java for unit testing) was used.
- System Testing : ApacheBench, a single-threaded command line computer program used for benchmarking HTTP web servers, was used. ApacheBench ( ab ) measures the performance of a web server by inundating it with HTTP requests and recording metrics for latency and success.

## 2 Unit Testing

### 1. BuyerAuthController:

The test was to check Buyer Signup and Buyer Login components. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : BuyerAuthController, Functions: BuyerSignup. BuyerLogin

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated where as under wrong and empty inputs exceptions were raised.

Inputs: {email:buyer@gmail.com, name = buyer1@gmail.com , password= "####"}  
Output: Validation Key : sdfvb8937b [PASSED]

Inputs: {email:buyer@gmail.com, name = buyer1@gmail.com , password= "###"}  
Output: Password not valid [PASSED]

Inputs: {email:buyer@gmail.com, name = buyer1@gmail , password= "####"}  
Output: Buyer not found [PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

**Additional Comments:** Additional Tests regarding null inputs were conducted but not mentioned which can be found at [test code link](#).

### 2. Cart Controller:

The test was to check whether the user is able to add product to the cart or not. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : Cart Controller, Functions: addToCart

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated where as under wrong and empty inputs exceptions were raised.

Inputs: {auth key:1111, quantity: 2, storelug:"store1".productID:"123"}  
Output: Product added to cart[PASSED]

Inputs: {auth key:1111, quantity: 2, storelug:"store1".productID:"---"}  
Output: Product not found[PASSED]

Inputs: {auth key:1111, quantity: 2, storelug:"store\_not\_valid".productID:"123"}  
Output: store id not valid[PASSED]

Inputs: {auth key:1111, quantity: -2, storelug:"store1".productID:"123"}  
Output: Quantity not valid[PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

**Additional Comments:** Additional Tests regarding null inputs were conducted but not mentioned which can be found at [test code link](#).

### 3. CategoryController:

The test was to check whether the seller is able to add a category to the catalogue or not. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : CategoryController, Functions: addCategory

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated whereas under wrong and empty inputs exceptions were raised.

Inputs: {category object:"category1", storelug:"store1"}

Output: Category Added[PASSED]

Inputs: {category object:"category1", storelug:"store-1"}

Output: Invalid store[PASSED]

Inputs: {category object:"Null", storelug:"store1"}

Output: Invalid Category Name[PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

**Additional Comments:** Additional Tests regarding null inputs were conducted but not mentioned which can be found at [test code link](#).

### 4. SellerAuthController:

The test was to check Seller Signup and Buyer Login components. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : SellerAuthController, Functions: SellerSignup. SellerLogin

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated whereas under wrong and empty inputs exceptions were raised.

Inputs: {email:seller@gmail.com, name = seller1@gmail.com , password= "####"}

Output: Validation Key : sdifvb8937b [PASSED]

Inputs: {email:seller@gmail.com, name = seller1@gmail.com , password= "###"}

Output: Password not valid [PASSED]

Inputs: {email:seller@gmail.com, name = seller1@gmail , password= "####"}

Output: Seller not found [PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

**Additional Comments:** *Additional Tests regarding Sign Up were conducted but not mentioned which can be found at [test code link](#).*

## 5. OrderController:

The test was to check whether Buyer can change order status updates. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : OrderController.java, Functions:updateStatus

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated where as under wrong and empty inputs exceptions were raised.

Inputs: {orderId:"1234", updated order object , auth object= "####"}  
Output: Order updated [PASSED]

Inputs: {orderId:"0000", updated order object , auth object= "####"}  
Output: Order id invalid [PASSED]

Inputs: {irderID:"1234", wrong order object , auth object= "####"}  
Output: Order object invalid [PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

**Additional Comments:** *Additional Tests regarding Sign Up were conducted but not mentioned.*

## 6. OTPController:

The test was to check whether login otp is being generated or not. How the component is reacting under right, wrong and empty inputs.

The tests conducted passed successfully.

**Unit Details:** Class : OTPController.java, Functions: getOTP

**Test Owner:** Deepankur Kansal

**Test Date:** [20/03/22] - [21/03/22]

**Test Results:** Passed. Under right inputs, validation key was generated where as under wrong and empty inputs exceptions were raised.

Inputs: {email:"deepank.kansal@gmail.com"}  
Output: OTP sent [PASSED]

Inputs: {email:"not valid email"}  
Output: EMAIL INVALID[PASSED]

**Structural Coverage:** Code Coverage :100% and Branch Coverage :100%

### 3 Integration Testing

1. **SellerLogin.java** : Tested the Login feature for the seller.

**Module Details:** module containing the tests for seller login and calling API for authentication of input credentials.

**Test Owner:** Antreev Singh Brar

**Test Date:** 04/01/22 - 04/03/22

**Test Results:**

A new seller object was created and the seller credentials were hard coded as the input :

```
Seller seller = new Seller();

seller.setName("nope");
seller.setEmail("harshit9304@gmail.com");
seller.setPassword("123");
```

An API call was made to authenticate the input credentials.

The result was extracted and verified using print statements :

```
ResponseEntity<String> result = template.postForEntity(uri,request ,String.class);

//Verify request succeed
String body = result.getBody();
System.out.println("0");
System.out.println(body);
System.out.println(result.getStatusCodeValue());
System.out.println(result.getHeaders());
System.out.println(result.getBody());
Assert.assertEquals(200, result.getStatusCodeValue());
```



After successful login, the seller could view their dashboard and options to perform other functions of a seller.

**Additional Comments:** The test module can be viewed at the following link : [store-it-backend/SellerLogin.java at main · localghosts/store-it-backend \(github.com\)](https://github.com/localghosts/store-it-backend/blob/main/SellerLogin.java)

## 2. sellerStore.java : tests related to the store details of a seller

**Module Details:** module containing tests for whether a seller can view his/her store details such as store logo, products, banner, name, etc.

**Test Owner:** Antreev Singh Brar

**Test Date:** 04/01/22 - 04/03/22

**Test Results:**

A new seller object was created with hard coded input credentials, which were authenticated before successful login:

```

        Seller seller = new Seller();

        seller.setName("nope");
        seller.setEmail("harshit9304@gmail.com");
        seller.setPassword("123");

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        HttpEntity<Seller> request = new HttpEntity<Seller>(seller, headers);

        System.out.println(request.getHeaders());
        System.out.println(request.getClass());
        System.out.println(request.getBody().getEmail());
        System.out.println(request.getBody().getPassword());
        System.out.println(request.getBody().getName());

        ResponseEntity<String> result = template.postForEntity(uri,request ,String.class);

        //Verify request succeed
        String body = result.getBody();
        System.out.println("0");
        System.out.println(body);
        System.out.println(result.getStatusCodeValue());
        System.out.println(result.getHeaders());
        System.out.println(result.getBody());

        Assert.assertEquals(200, result.getStatusCodeValue());
```

```
JSONObject myobj = new JSONObject(body);
System.out.println(myobj.getString("token"));
System.out.println(myobj.getString("storeSlug"));
String bearerString = myobj.getString("token") ;
String storeslug = myobj.getString("storeSlug");
//String bearerString = result.getBody().get
////
TestRestTemplate templatestore = new TestRestTemplate();
final String baseUrlstore = "http://localhost:8080/store/" + storeslug;
URI uristore = new URI(baseUrlstore);
HttpHeaders headerstore = new HttpHeaders();
headerstore.setContentType(MediaType.APPLICATION_JSON);
headerstore.add("Authorization", bearerString);

ResponseEntity<String> entity = templatestore.exchange(
    uristore, HttpMethod.GET, new HttpEntity<Object>(headerstore),
    String.class);
System.out.print(entity.getBody());
Assert.assertEquals(HttpStatus.OK, entity.getStatusCode());
}
```

As a result, the seller can view the details of their store such as store name, logo, banner, etc.

**Additional Comments:** The test module for this test can be viewed at the following link : [store-it-backend/sellerStore.java at main · localghosts/store-it-backend \(github.com\)](https://github.com/localghosts/store-it-backend/blob/main/sellerStore.java)

### 3. BuyerLogin.java : tested the login feature for the buyer.

**Module Details:** module containing the tests for buyer login and calling API for authentication of input credentials.

**Test Owner:** Antreev Singh Brar

**Test Date:** 04/01/22 - 04/03/22

**Test Results:**

A new buyer object was created with hard coded credentials :

```
Buyer buyer = new Buyer();

buyer.setEmail("antdany1298@gmail.com");
buyer.setPassword("123");
```

The user credentials were authenticated, the result was extracted and verified:

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
HttpEntity<Buyer> request = new HttpEntity<Buyer>(buyer, headers);

System.out.println(request.getHeaders());
System.out.println(request.getClass());
System.out.println(request.getBody().getEmail());
System.out.println(request.getBody().getPassword());
// System.out.println(request.getBody().getName());

ResponseEntity<String> result = template.postForEntity(uri,request ,String.class);

//Verify request succeed
String body = result.getBody();
System.out.println("0");
System.out.println(body);
System.out.println(result.getStatusCodeValue());
System.out.println(result.getHeaders());
System.out.println(result.getBody());
Assert.assertEquals(200, result.getStatusCodeValue());
}
```

**Additional Comments:** The test module for this test can be viewed at the following link : [store-it-backend/BuyerLogin.java at main · localghosts/store-it-backend \(github.com\)](https://github.com/store-it-backend/BuyerLogin.java)

#### 4. *BuyerOrders.java* : tests for the Orders view for the buyer.

**Module Details:** module contains the tests to check whether the buyer can view his/her placed orders through the Orders view in their dashboard.

**Test Owner:** Antreev Singh Brar

**Test Date:** 04/01/22 - 04/03/22

**Test Results:**

A new buyer object is created and their input credentials are hard coded, and authenticated before successful login:

```
Buyer buyer = new Buyer();

buyer.setEmail("antdany1298@gmail.com");
buyer.setPassword("123");

HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);
HttpEntity<Buyer> request = new HttpEntity<Buyer>(buyer, headers);

System.out.println(request.getHeaders());
System.out.println(request.getClass());
System.out.println(request.getBody().getEmail());
System.out.println(request.getBody().getPassword());
System.out.println(request.getBody().getName());

ResponseEntity<String> result = template.postForEntity(uri,request ,String.class);

//Verify request succeed
String body = result.getBody();
System.out.println("");
System.out.println(body);
System.out.println(result.getStatusCodeValue());
System.out.println(result.getHeaders());
System.out.println(result.getBody());
Assert.assertEquals(200, result.getStatusCodeValue());
```

The orders view can be viewed by the buyer where they can view the status of their placed orders :

```
JSONObject myobj = new JSONObject(body);
System.out.println(myobj.getString("token"));
System.out.println(myobj.getString("storeSlug"));
String bearerString = myobj.getString("token") ;

//String bearerString = result.getBody().get
////
TestRestTemplate templatestore = new TestRestTemplate();
final String baseUrlstore = "http://localhost:8080/orders";
URI uristore = new URI(baseUrlstore);
HttpHeaders headerstore = new HttpHeaders();
headerstore.setContentType(MediaType.APPLICATION_JSON);
headerstore.add("Authorization", bearerString);

ResponseEntity<String> entity = templatestore.exchange(
    uristore, HttpMethod.GET, new HttpEntity<Object>(headerstore),
    String.class);
System.out.print(entity.getBody());
Assert.assertEquals(HttpStatus.OK, entity.getStatusCode());

}
```

**Additional Comments:** The test module for this test can be viewed at the following link : [store-it-backend/BuyerOrders.java at main · localghosts/store-it-backend \(github.com\)](https://github.com/localghosts/store-it-backend/blob/main/src/test/java/BuyerOrders.java)

## 5. BuyergetCategories.java : tests for the Categories view for the buyer.

**Module Details:** module contains the tests to check whether the buyer can view the categories through the Categories view in their dashboard.

**Test Owner:** Antreev Singh Brar

**Test Date:** 04/01/22 - 04/03/22

**Test Results:**

A new buyer object was created with hard coded input credentials which were authenticated before successful login :

```
    Buyer buyer = new Buyer();

    buyer.setEmail("antdany1298@gmail.com");
    buyer.setPassword("123");

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_JSON);
    HttpEntity<Buyer> request = new HttpEntity<Buyer>(buyer, headers);

    System.out.println(request.getHeaders());
    System.out.println(request.getClass());
    System.out.println(request.getBody().getEmail());
    System.out.println(request.getBody().getPassword());
    System.out.println(request.getBody().getName());

    ResponseEntity<String> result = template.postForEntity(uri,request ,String.class);

    //Verify request succeed
    String body = result.getBody();
    System.out.println("");
    System.out.println(body);
    System.out.println(result.getStatusCodeValue());
    System.out.println(result.getHeaders());
    System.out.println(result.getBody());
    Assert.assertEquals(200, result.getStatusCodeValue());
```

The categories view can be viewed by the buyer where they can view different categories available in different stores :

```
JSONObject myobj = new JSONObject(body);
System.out.println(myobj.getString("token"));
System.out.println(myobj.getString("storeSlug"));
String bearerString = myobj.getString("token") ;
String storeslug = "goodfoods";
//String bearerString = result.getBody().get
////
TestRestTemplate templatestore = new TestRestTemplate();
final String baseUrlstore = "http://localhost:8080/store/" + storeslug+"/category";
URI uristore = new URI(baseUrlstore);
HttpHeaders headerstore = new HttpHeaders();
headerstore.setContentType(MediaType.APPLICATION_JSON);
headerstore.add("Authorization", bearerString);

ResponseEntity<String> entity = templatestore.exchange(
    uristore, HttpMethod.GET, new HttpEntity<Object>(headerstore),
    String.class);
System.out.print(entity.getBody());
Assert.assertEquals(HttpStatus.OK, entity.getStatusCode());

}
```

As a result, the buyer is able to view different categories through the Categories view after successful login.

**Additional Comments:** The tet modules for this test can be viewed at the following link : [store-it-backend/BuyergetCategories.java at main · localghosts/store-it-backend \(github.com\)](https://github.com/store-it-backend/BuyergetCategories.java)

....

## 4 Frontend Testing

These tests have been added after the beta testing phase of our project in compliance with the issues reported.

- Alphanumeric Password with necessary special characters and length within 8-15 should be accepted

**Test Type:** Manual

**Test Owner:** Antreev Singh Brar

**Test Date:** 27/04/22

**Test Result:**

The screenshot shows a registration form with two tabs: 'BUYER' (selected) and 'SELLER'. The form contains the following fields and messages:

- Name \***: Input field containing 'ant'.
- Email \***: Input field containing 'antdany@gmail.com'.
- Password \***: Input field containing three dots. Below it, a red message reads: 'Use password having atleast one special character, 1 lowercase letter, 1 uppercase letter and 1 numeric character of 8-15 character length'.
- Confirm Password**: Input field with a red border and a tooltip that says 'Please fill in this field.' Below it, a red message reads: 'Missing field'.
- SIGN UP**: A dark brown button.
- ALREADY A BUYER? LOG IN!**: A link with a user icon.



- OTP section needed to be hidden  
**Test Type:** Manual  
**Test Owner:** Antreev Singh Brar  
**Test Date:** 27/04/22  
**Test Result:**

The screenshot shows a mobile application interface for OTP verification. At the top, there are two tabs: 'BUYER' and 'SELLER', with 'BUYER' being the active tab. Below the tabs, a green success message box displays a checkmark icon and the text 'OTP sent to email!' with a close 'X' button. The form contains five input fields: 'Name \*' with the text 'ant', 'Email \*' with 'antdany@gmail.com', 'Password \*' with masked dots, 'Confirm Password \*' with masked dots, and 'OTP \*' with masked dots and a cursor. A brown 'VERIFY OTP' button is positioned at the bottom of the form.

BUYER SELLER

✓ OTP sent to email! ✕

Name \*  
ant

Email \*  
antdany@gmail.com

Password \*  
.....

Confirm Password \*  
.....

OTP \*  
.....|

VERIFY OTP

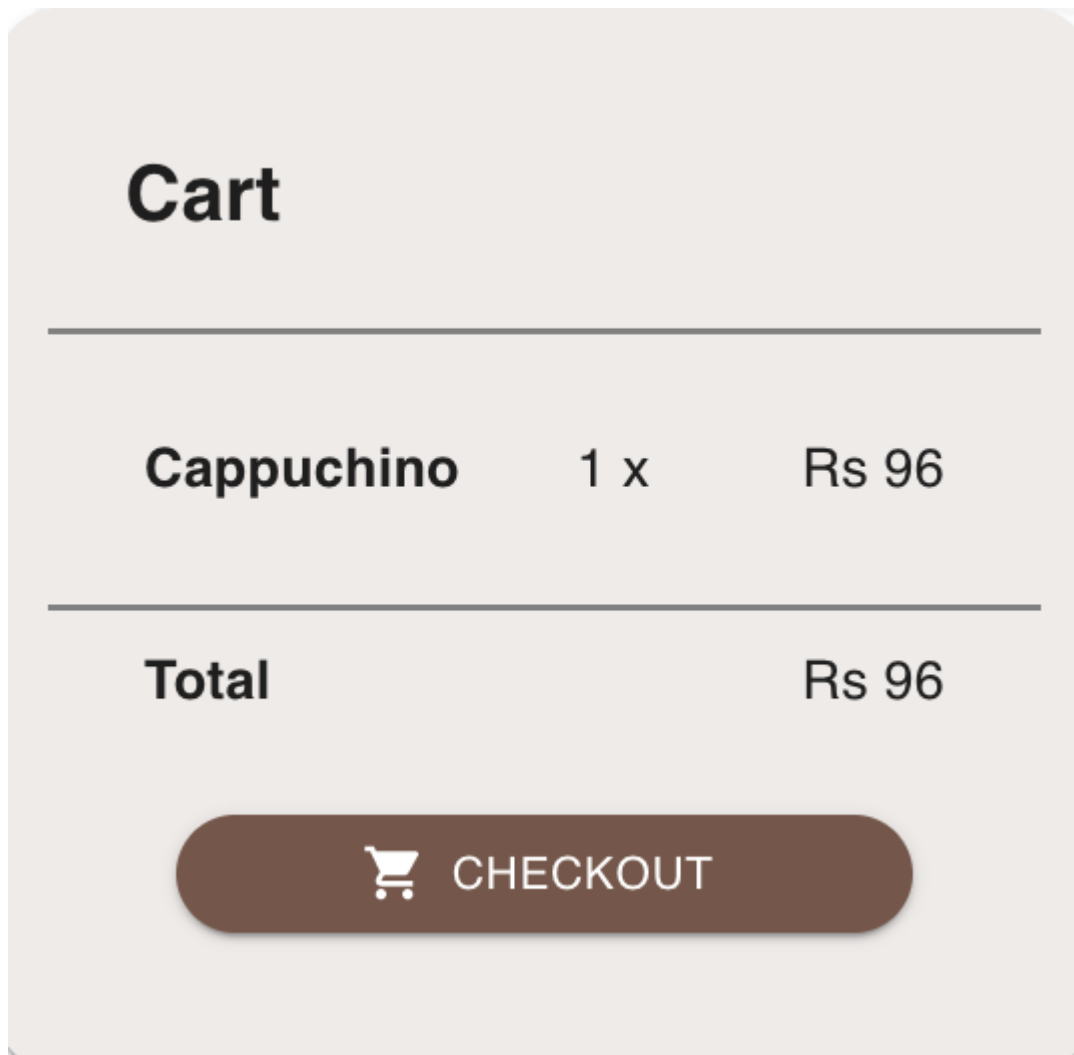
- Cart's section items should be non-overlapping

**Test Type:** Manual

**Test Owner:** Antreev Singh Brar

**Test Date:** 27/04/22

**Test Result:**



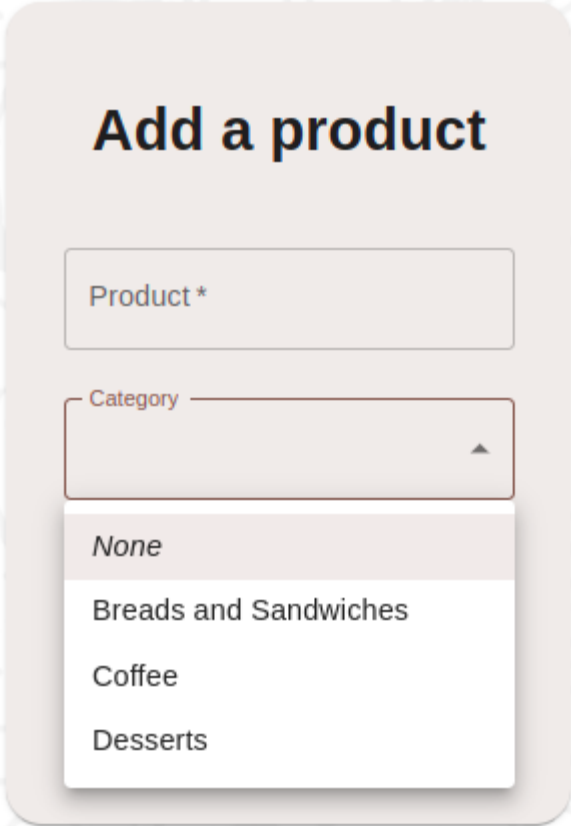
- Product category drop down list should in the alphabetical order in seller section.

**Test Type:** Manual

**Test Owner:** Antreev Singh Brar

**Test Date:** 27/04/22

**Test Result:**



The image shows a mobile application form titled "Add a product". It features a text input field for "Product\*" and a dropdown menu for "Category". The dropdown menu is open, showing four options: "None", "Breads and Sandwiches", "Coffee", and "Desserts". The form is set against a background with a faint, repeating pattern of coffee-related icons.

## Add a product

Product\*

Category

- None
- Breads and Sandwiches
- Coffee
- Desserts

## 5 System Testing

### 1. Requirement: Performance Requirements (Frontend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the frontend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

#### Test Results:

- Number of complete requests : 100
- Concurrency Level : 20

#### Test summary :

```

Concurrency Level:      20
Time taken for tests:    7.502 seconds
Complete requests:      100
Failed requests:        0
Total transferred:      204500 bytes
HTML transferred:       166700 bytes
Requests per second:    13.33 [#/sec] (mean)
Time per request:       1500.426 [ms] (mean)
Time per request:       75.021 [ms] (mean, across all concurrent requests)
Transfer rate:          26.62 [Kbytes/sec] received

```

```

Connection Times (ms)
      min  mean[+/-sd] median   max
Connect:    679    789 117.8    763   1768
Processing:  232    284  98.1    261   1191
Waiting:    232    283  98.2    261   1191
Total:      916   1072 164.9   1023   2088

```

#### Percentage of the requests served within a certain time (ms)

```

50%    1023
66%    1048
75%    1106
80%    1141
90%    1212
95%    1237
98%    2025
99%    2088
100%   2088 (longest request)

```

**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs). This test is included in the “frontend” subfolder.

## 2. Requirement: Performance Requirements (Frontend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the frontend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

### Test Results:

- Number of Complete requests : 20000
- Concurrency Level : 1000

Test summary :

```
Concurrency Level:      1000
Time taken for tests:   6.017 seconds
Complete requests:      20000
Failed requests:        0
Total transferred:      41100000 bytes
HTML transferred:       34180000 bytes
Requests per second:    3324.05 [#/sec] (mean)
Time per request:       300.838 [ms] (mean)
Time per request:       0.301 [ms] (mean, across all concurrent requests)
Transfer rate:          6670.81 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	47 288.7	3	3062
Processing:	38	241 29.5	237	334
Waiting:	1	188 31.4	191	278
Total:	38	288 285.9	241	3299

#### Percentage of the requests served within a certain time (ms)

50%	241
66%	249
75%	260
80%	270
90%	290
95%	327
98%	1183
99%	1233
100%	3299 (longest request)

**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs). This test is included in the “frontend” subfolder.

### 3. Requirement: Performance Requirements (Frontend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the frontend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

#### Test Results:

- Number of complete requests : 3000
- Concurrency level : 100

#### Test summary :

```
Concurrency Level:      100
Time taken for tests:   0.987 seconds
Complete requests:     3000
Failed requests:       0
Total transferred:     6165000 bytes
HTML transferred:     5127000 bytes
Requests per second:   3040.90 [#/sec] (mean)
Time per request:      32.885 [ms] (mean)
Time per request:      0.329 [ms] (mean, across all concurrent requests)
Transfer rate:         6102.59 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	1 0.5	1	4
Processing:	2	32 6.1	30	59
Waiting:	2	22 5.6	21	45
Total:	2	33 6.2	31	59

#### Percentage of the requests served within a certain time (ms)

50%	31
66%	33
75%	35
80%	36
90%	41
95%	43
98%	58
99%	58
100%	59 (longest request)

**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests :

<https://github.com/localhosts/store-it-backend/tree/main/src/test/java/com/localhosts/storeit>. This test is included in the “frontend” subfolder.

#### 4. Requirement: Performance Requirements (Backend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the backend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

#### Test Results:

- Number of complete requests : 100
- Concurrency level : 20

Test summary :

```
Concurrency Level:      20
Time taken for tests:    0.242 seconds
Complete requests:      100
Failed requests:        15
    (Connect: 0, Receive: 0, Length: 15, Exceptions: 0)
Total transferred:      25682 bytes
HTML transferred:       8982 bytes
Requests per second:    412.45 [#/sec] (mean)
Time per request:       48.490 [ms] (mean)
Time per request:       2.425 [ms] (mean, across all concurrent requests)
Transfer rate:          103.44 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	2	4 1.6	3	11
Processing:	3	11 30.8	6	228
Waiting:	2	8 21.7	6	221
Total:	5	14 30.8	9	231

#### Percentage of the requests served within a certain time (ms)

50%	9
66%	10
75%	11
80%	11
90%	17
95%	18
98%	224
99%	231
100%	231 (longest request)

**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs). This test is included in the “backend” subfolder.

## 5. Requirement: Performance Requirements (Backend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the backend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

### Test Results:

- Number of complete requests : 20000
- Concurrency level : 1000

Test summary :

```
Concurrency Level:      1000
Time taken for tests:   10.692 seconds
Complete requests:      20000
Failed requests:        2001
    (Connect: 0, Receive: 0, Length: 2001, Exceptions: 0)
Total transferred:      5137774 bytes
HTML transferred:       1797774 bytes
Requests per second:    1870.51 [#/sec] (mean)
Time per request:       534.615 [ms] (mean)
Time per request:       0.535 [ms] (mean, across all concurrent requests)
Transfer rate:          469.25 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	3	110 224.5	63	3207
Processing:	13	113 222.3	75	7147
Waiting:	3	111 219.0	74	7147
Total:	16	223 310.3	142	7188

#### Percentage of the requests served within a certain time (ms)

50%	142
66%	162
75%	173
80%	179
90%	361
95%	1110
98%	1169
99%	1196
100%	7188 (longest request)



**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs). This test is included in the “backend” subfolder.

## 6. Requirement: Performance Requirements (Backend component)

The number of requests executed per second, time taken per request, number of failed requests were tested for the backend component. Finally, the percentage of requests served within a certain time is also reported.

**Test Owner:** Harshit Raj

**Test Date:** 04/01/2022 - 04/03/2022

### Test Results:

- Number of complete requests : 3000
- Concurrency level : 100

### Test summary:

```
Concurrency Level:      100
Time taken for tests:    1.539 seconds
Complete requests:      3000
Failed requests:        2739
    (Connect: 0, Receive: 0, Length: 2739, Exceptions: 0)
Total transferred:      770659 bytes
HTML transferred:       269659 bytes
Requests per second:    1948.73 [#/sec] (mean)
Time per request:       51.315 [ms] (mean)
Time per request:       0.513 [ms] (mean, across all concurrent requests)
Transfer rate:          488.87 [Kbytes/sec] received
```

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      2    31 149.7      7   1014
Processing:   2    13  26.5      9    702
Waiting:      2    12  24.4      8    702
Total:        4    44 151.6     17   1041
```

```
Percentage of the requests served within a certain time (ms)
 50%    17
 66%    21
 75%    23
 80%    25
 90%    32
 95%    62
 98%   1010
 99%   1015
100%   1041 (longest request)
```

**Additional Comments:** The following github repo contains the test module(tests.sh) and results for the system tests : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs). This test is included in the “backend” subfolder.

....

## 6 Conclusion

### ***How Effective and exhaustive was the testing?***

Most of the unit tests had 100% code coverage and 100% branch coverage. Integration tests covered some major features such as seller login, buyer login, interaction among different views such as login, categories, orders, etc. Stress testing (a type of testing that is so harsh, it is expected to push the program to failure) was done at multiple concurrency levels and thread counts.

### ***Which components have not been tested adequately?***

Some of the components which involved mocking have not been tested using an automation suite. However it was tested thoroughly manually with the help of some extensions.

### ***What difficulties have you faced during testing?***

Large interconnection among different modules and views made it difficult to perform the integration testing.

### ***How could the testing process be improved?***

Testing process could be improved by adding CI tests on every pull request and by having even better code coverage. Some specific test suites must exist to cater our specific need that can be looked into.

## Appendix A - Group Log

- The group activities during implementation were asynchronous, where we kept constant communication via our discord server and WhatsApp group.
- Stress testing was done using ApacheBench and the test module and the test results were pushed to the test-logs github repo, the link to the access the same is : [localghosts/store-it-test-logs \(github.com\)](https://github.com/localghosts/store-it-test-logs)

localghosts / store-it-test-logs Private

Watch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Security Insights

main 1 branch 0 tags

Go to file Add file Code

About

No description, website, or topics provided.

Readme

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

1-Harshit Update Readme a54b052 3 hours ago 4 commits

backend	Final commit	3 hours ago
frontend	Final commit	3 hours ago
README.md	Update Readme	3 hours ago
tests.sh	Final commit	3 hours ago

README.md

### Stress Testing

Stress testing refers to a type of testing that is so harsh, it is expected to push the program to failure. Here we flood our web application with data, connections, and so on until it finally crashes. The fact of the crash might be unremarkable.

- Integration tests were written manually and pushed on the Github repo created for the backend components. The tests can be viewed at : [store-it-backend/src/test/java/com/localghosts/storeit at main · localghosts/store-it-backend \(github.com\)](https://github.com/localghosts/store-it-backend/tree/main/src/test/java/com/localghosts/storeit)

The screenshot shows the GitHub interface for the repository 'localghosts / store-it-backend'. The repository is public and has 1 watch, 0 forks, and 0 stars. The navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, and Insights. The file browser shows the path 'main / store-it-backend / src / test / java / com / localghosts / storeit /'. A file named 'antreev-brar more test' is highlighted, with a commit hash 'c2cc816' and a timestamp '2 hours ago'. Below this, a list of files is shown, including 'BuyerLogin.java', 'BuyerOrders.java', 'BuyergetCategories.java', 'SellerLogin.java', and 'sellerStore.java', each with its commit hash and timestamp.

File Name	Commit Hash	Timestamp
BuyerLogin.java	c2cc816	2 hours ago
BuyerOrders.java	c2cc816	2 hours ago
BuyergetCategories.java	c2cc816	2 hours ago
SellerLogin.java	c2cc816	2 hours ago
sellerStore.java	c2cc816	2 hours ago

- Testing was completed on 3 Apr, 2022. The testing process was documented in parallel with the testing.
- The documentation was completed on 3 Apr, 2022 after coordinating with the unit, integration and system testers.
- The final updates were completed on 27th April, 2022.