VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

Winter Semester 2022-2023

SWE2005 - SOFTWARE TESTING

J-COMPONENT PROJECT

REVIEW – 3

TOPIC

TESTING A PIZZA BILLING SYSTEM

FACULTY

PRASANNA M

SLOT

B1 + TB1

Team members

Yagavi K - 21MIS0051
Pranay Gorantla - 21MIS0123
Shakthi Shamruth Dhamotharan - 21MIS0244
Muthukumar S - 21MIS0272

Problem Statement:

In today's fast-paced society, many customers order and buy many things. For every order, a certain cost is fixed and the bill is generated. Also, as the day passes and the customer's interest increases, the orders will increase, more data will be stored, and many bills need to be generated. So, with the increase in popularity among the customers, the orders will increase which in turn increases the data need to be processed. To be more productive in order processing, he/she needs a solution that can facilitate their current processes with the use of technology and software. A billing system is required to do billing for increased number of customers. A billing system needs to be very accurate because it involves money value for every digit. It is very hard to go through all backtracking orders. Hence the billing needs to be perfect and accurate. A small change in bill amount may cause a problem for the customer and if the bill amount is assigned to the wrong customer then the customer may claim the wrong bill and fake orders. Many issues may arise if the billing system does not function properly. If there is any complaint or review of any order, it takes a large amount of effort and time to backtrack and fix the problem. This results in a loss of resources. To reduce these problems, we need to test the billing system before launching it for use. So, in this project, we are developing a pizza billing system and we test the developed system. Pizza is one of the most popular foods in the world, and many people prefer to order it online rather than going to the restaurant or calling in their order. However, the current pizza ordering and billing process can be cumbersome and prone to errors, leading to customer dissatisfaction and revenue loss for the pizza restaurant. The traditional process involves customers selecting their pizza and toppings through a website or mobile app, then placing their order online. However, if the website is not user-friendly or if there are technical issues, customers may become frustrated and abandon their order. Additionally, errors can occur during the ordering process, such as incorrect pizza toppings or inaccurate delivery information, leading to delays and unhappy customers. The billing process is also problematic, as customers may have to wait in line to pay, or the payment process may be confusing and time-consuming. This can lead to long wait times and dissatisfied customers, resulting in loss of revenue for the pizza restaurant. Therefore, an efficient and accurate online pizza billing system is needed to streamline the ordering and payment process, reduce errors, and enhance customer satisfaction. This system should be user friendly for both customers and employees, allowing for quick and accurate orders and payments.

By implementing such a system, pizza restaurants can increase efficiency, reduce errors, and improve customer satisfaction, leading to increased revenue and growth.

The pizza billing software should be easy to use, reliable, and secure, with a user-friendly interface and support for multiple languages and currencies. It should also be compatible with different hardware devices, such as printers, cash registers, and card readers.

Here, we develop a pizza billing software that automates the process of generating bills for pizza orders.

The software contains the following features:

Order management: The software should allow restaurant staff to input orders. The orders should include information about the pizza type, size, toppings, and quantity.

Menu management: The software should allow restaurant staff to update the menu, including pizza (veg/non-veg), pizza base, toppings, and prices.

Pricing and discounts: The software should be able to calculate the price of an order based on the menu items selected and any applicable discounts, such as discounts for large orders or coupons.

Payment processing: The software should be able to process payments, including cash, credit/debit cards, and other payment methods.

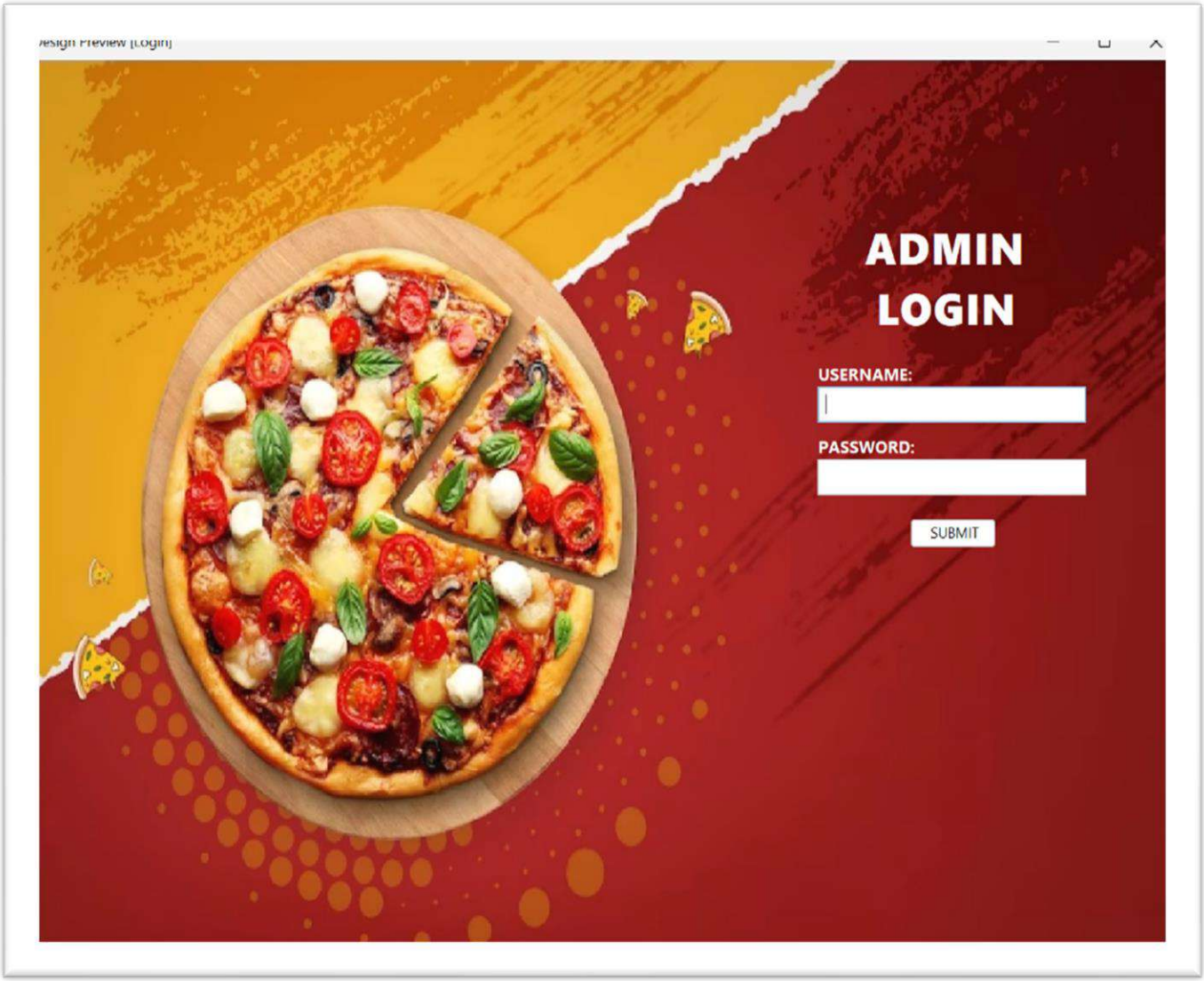
Billing and invoicing: The software should generate bills for each order, including a breakdown of the items ordered, any discounts applied, and the total amount due.

Modules:

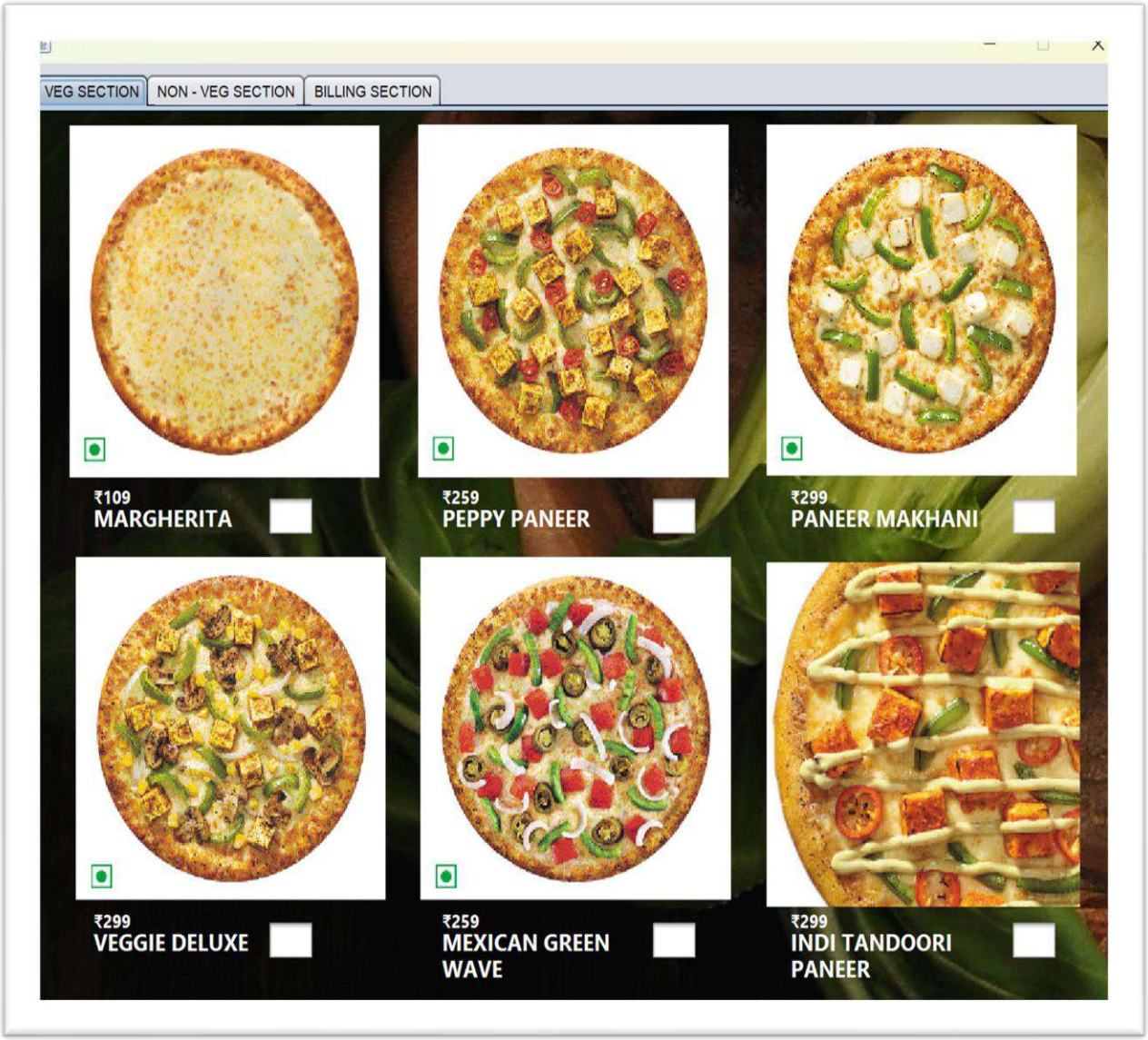
- Login
- Veg section
- Non-veg section
- Billing section

Pizza Billing System Interface:

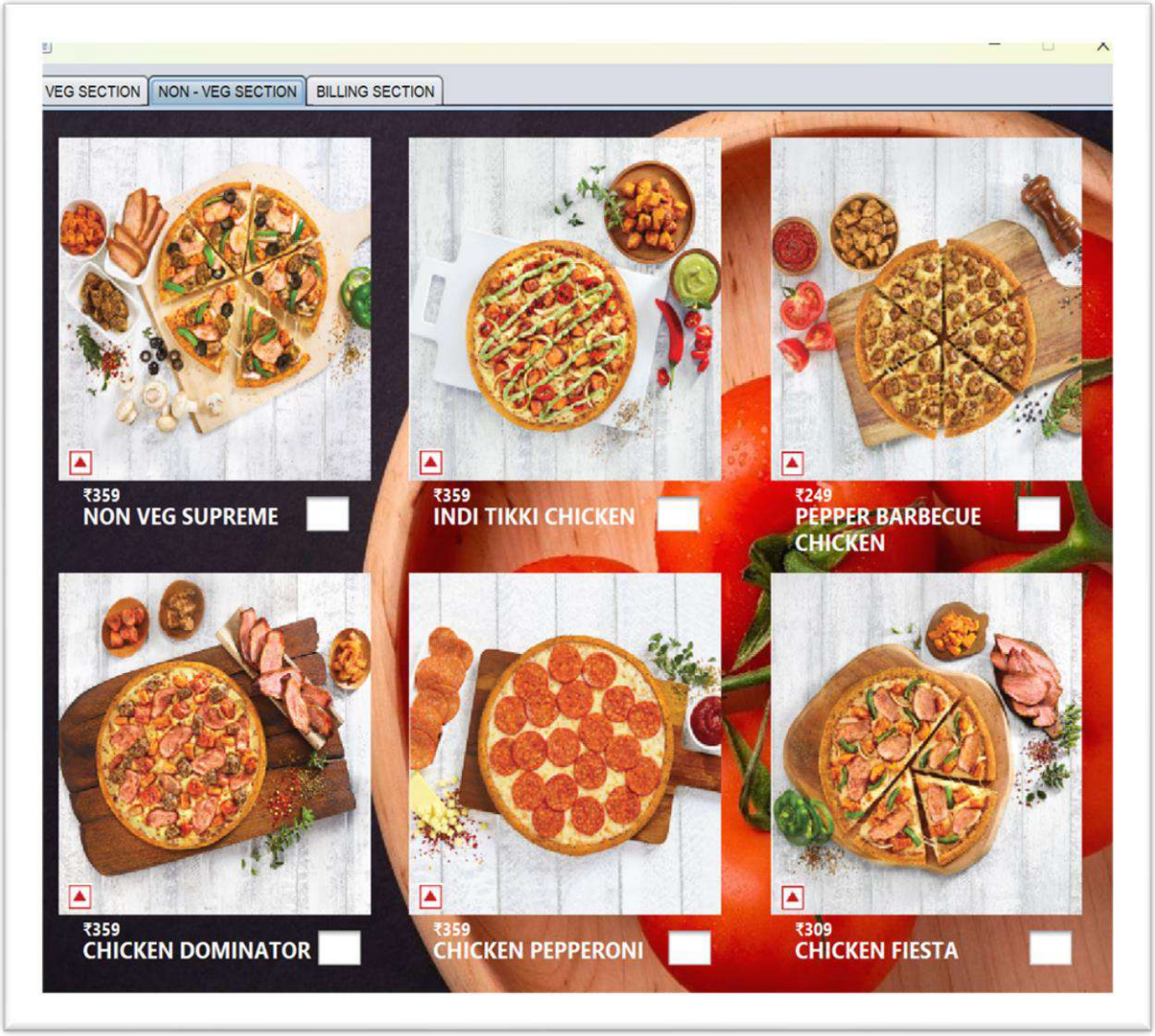
LOGIN



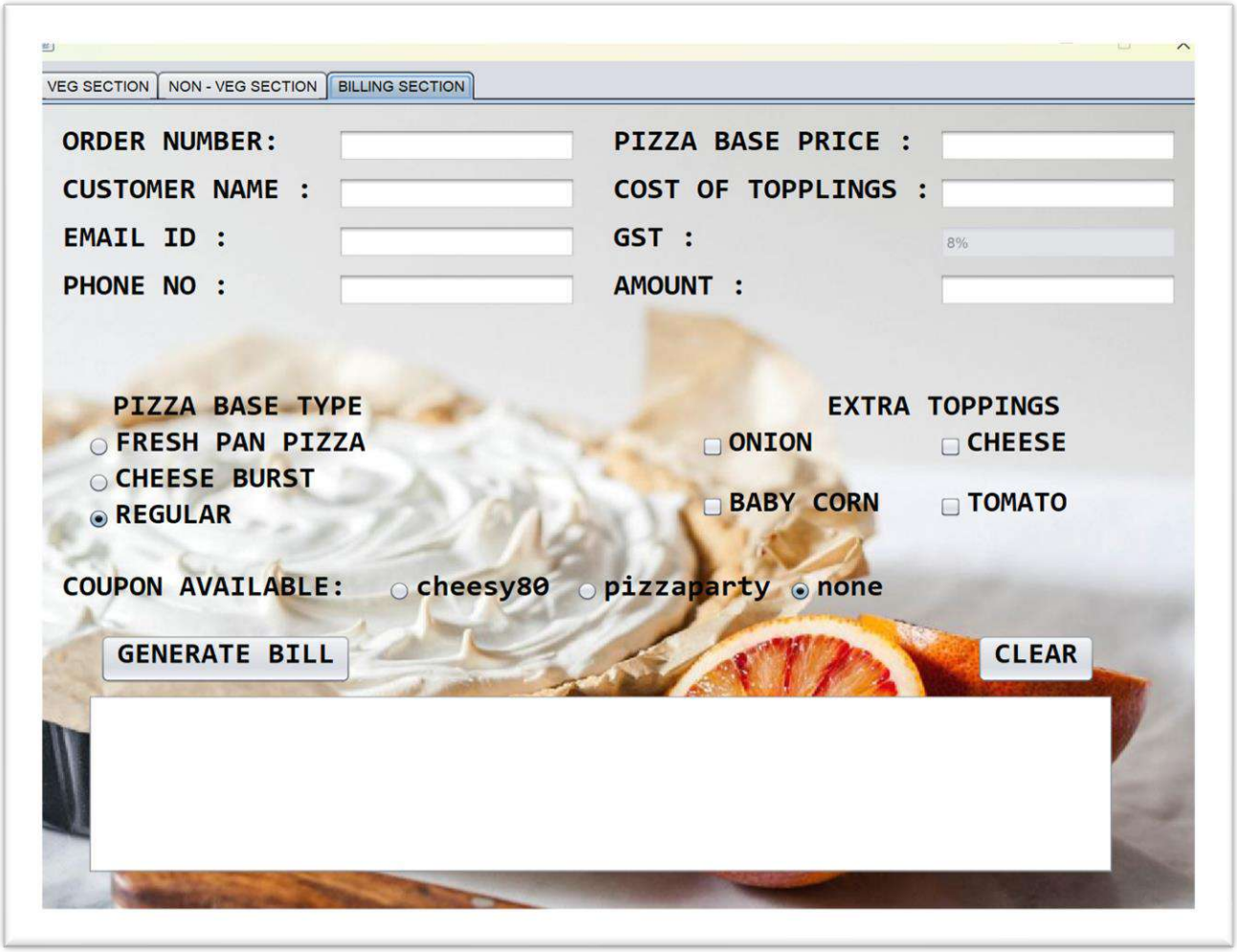
VEG SECTION


VEG SECTION
NON - VEG SECTION
BILLING SECTION
₹109
MARGHERITA
₹259
PEPPY PANEER
₹299
PANEER MAKHANI
₹299
VEGGIE DELUXE
₹259
MEXICAN GREEN WAVE
₹299
INDI TANDOORI PANEER

NON-VEG SECTION



BILLING SECTION



Black box testing techniques applied:

i) Login – Random testing:

TESTID	USERNAME	PASSWORD	EXPECTED
1	ADMIN	ADMIN	(Allow access)
2	ADMIN	XYZ	(Don't allow access)
3	ADMIN	abc	(Don't allow access)
4	ADMIN	123	(Don't allow access)
5	ADMIN	Abc123	(Don't allow access)
6	admin	admin	(Don't allow access)
7	admin	xyz	(Don't allow access)
8	admin	ADMIN	(Don't allow access)
9	123	123	(Don't allow access)
10	xyz	abc	(Don't allow access)

ii) Veg Pizza - Boundary value analysis:

Test cases are designed by holding one variable at its extreme value and other variables at their nominal values in the input domain. The variable at its extreme value can be selected at:

- ✓ Minimum value (Min) - 0
- ✓ Value just above the minimum value (Min+) - 1
- ✓ Maximum value (Max) - 5
- ✓ Value just below the maximum value (Max-) - 4
- ✓ Nominal or Average Value – 3
- ✓ BVA: Number of testcases formula: 4n+1

  4n+1→ 4(6) + 1 =25

  where n (number of variables) = 6

Therefore, 25 testcases need to be designed for boundary value analysis.

Test matrix table (No extra toppings with regular base for veg pizza)

TESTCASE ID	MARGHERITA	PEPPY PANEER	PANEER MAKHANI	VEGGIE DELUXE	MEXICAN GREEN WAVE	INDI TANDOORI PANEER	EXPECTED OUTPUT
1	3	3	3	3	3	0	3969
2	3	3	3	3	3	1	4292
3	3	3	3	3	3	3	4938
4	3	3	3	3	3	4	5261
5	3	3	3	3	3	5	5584
6	3	3	3	3	0	3	4099
7	3	3	3	3	1	3	4378
8	3	3	3	3	4	3	5217
9	3	3	3	3	5	3	5497
10	3	3	3	0	3	3	3969
11	3	3	3	1	3	3	4292
12	3	3	3	4	3	3	5261
13	3	3	3	5	3	3	5584
14	3	3	0	3	3	3	3969
15	3	3	1	3	3	3	4292
16	3	3	4	3	3	3	5261
17	3	3	5	3	3	3	5584
18	3	0	3	3	3	3	4099
19	3	1	3	3	3	3	4378
20	3	4	3	3	3	3	5217
21	3	5	3	3	3	3	5497
22	0	3	3	3	3	3	4585
23	1	3	3	3	3	3	4702
24	4	3	3	3	3	3	5055
25	5	3	3	3	3	3	5173

iii) Non-Veg Pizza - Boundary value analysis:

Test cases are designed by holding one variable at its extreme value and other variables at their nominal values in the input domain. The variable at its extreme value can be selected at:

- ✓ Minimum value (Min) - 0
- ✓ Value just above the minimum value (Min+) - 1
- ✓ Maximum value (Max) - 5
- ✓ Value just below the maximum value (Max-) - 4
- ✓ Nominal or Average Value – 3
- ✓ BVA: Number of testcases formula: 4n+1

  4n+1→ 4(6) + 1 =25

  where n (number of variables) = 6

  Therefore, 25 testcases need to be designed for boundary value analysis.

Test matrix table (No extra toppings with regular toppings for non-veg pizza)

TESTCASE ID	NON-VEG SUPREME	INDI TIKKI CHICKEN	PEPPER BARBECUE CHICKEN	CHICKEN DOMINATOR	CHICKEN PEPPERNONI	CHICKEN FIESTA	EXPECTED OUTPUT
1	3	3	3	3	3	0	5459
2	3	3	3	3	3	1	5793
3	3	3	3	3	3	3	6461
4	3	3	3	3	3	4	6794
5	3	3	3	3	3	5	7128
6	3	3	3	3	0	3	5297
7	3	3	3	3	1	3	5685
8	3	3	3	3	4	3	6848
9	3	3	3	3	5	3	7236
10	3	3	3	0	3	3	5297
11	3	3	3	1	3	3	5685
12	3	3	3	4	3	3	6848
13	3	3	3	5	3	3	7236

14	3	3	0	3	3	3	5654
15	3	3	1	3	3	3	5923
16	3	3	4	3	3	3	6729
17	3	3	5	3	3	3	6998
18	3	0	3	3	3	3	5297
19	3	1	3	3	3	3	5685
20	3	4	3	3	3	3	6848
21	3	5	3	3	3	3	7236
22	0	3	3	3	3	3	5297
23	1	3	3	3	3	3	5685
24	4	3	3	3	3	3	6848
25	5	3	3	3	3	3	7236

iv) Billing System

The decision table and test matrix table to validate the features including email id, phone number, order number, customer name, pizza ordered, toppings, in a pizza billing system using decision table black box testing technique.

Decision Table:

CONDITION	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6	RULE 7	RULE 8
Email ID	T	T	T	T	F	F	F	F
Phone Number	T	T	F	F	T	T	F	F
Customer Name	X	X	X	X	X	X	X	X
Pizza base type	T	F	T	F	T	F	T	F
Extra Toppings	X	X	X	X	X	X	X	X
Coupon	X	X	X	X	X	X	X	X
Action-Result	T	F	F	F	T	F	F	F

v) Veg Pizza - Robustness:

- A value just greater than the Maximum value (Max+)
- A value just less than Minimum value (Min-)
- When test cases are designed considering above points in addition to BVC, it is called Robustness testing.
- Min.Value -1, Min. Value, Min. Value+1, Nominal or Average Value, Max. Value-1, Max. Value, Max. Value +1
- It can be generalized that for n input variables in a module, 6n+1 test cases are designed with Robustness testing.

  6n+1→ 6(6) + 1 =37

  where n (number of variables) = 6

  Therefore, 37 testcases need to be designed for boundary value analysis.

Test matrix table (No extra toppings with regular toppings for veg pizza)

TESTCASE ID	MARGHERITA	PEPPY PANEER	PANEER MAKHANI	VEGGIE DELUXE	MEXICAN GREEN WAVE	INDI TANDOORI PANEER	EXPECTED OUTPUT
1	3	3	3	3	3	-1	X
2	3	3	3	3	3	0	3969
3	3	3	3	3	3	1	4292
4	3	3	3	3	3	3	4938
5	3	3	3	3	3	4	5261
6	3	3	3	3	3	5	5584
7	3	3	3	3	3	6	out of range(5907)
8	3	3	3	3	-1	3	X
9	3	3	3	3	0	3	4099
10	3	3	3	3	1	3	4378
11	3	3	3	3	4	3	5217
12	3	3	3	3	5	3	5497
13	3	3	3	3	6	3	out of range(11126)
14	3	3	3	-1	3	3	X

15	3	3	3	0	3	3	3969
16	3	3	3	1	3	3	4292
17	3	3	3	4	3	3	5261
18	3	3	3	5	3	3	5584
19	3	3	3	6	3	3	out of range(16845)
20	3	3	-1	3	3	3	X
21	3	3	0	3	3	3	3969
22	3	3	1	3	3	3	4292
23	3	3	4	3	3	3	5261
24	3	3	5	3	3	3	5584
25	3	3	6	3	3	3	out of range(22314)
26	3	-1	3	3	3	3	X
27	3	0	3	3	3	3	4099
28	3	1	3	3	3	3	4378
29	3	4	3	3	3	3	5217
30	3	5	3	3	3	3	5497
31	3	6	3	3	3	3	out of range(27173)
32	-1	3	3	3	3	3	X
33	0	3	3	3	3	3	4585
34	1	3	3	3	3	3	4702
35	4	3	3	3	3	3	5055
36	5	3	3	3	3	3	5173
37	6	3	3	3	3	3	out of range(29786)

vi) Non-Veg Pizza – Robustness:

- A value just greater than the Maximum value (Max+)
- A value just less than Minimum value (Min-)

- When test cases are designed considering above points in addition to BVC, it is called Robustness testing.
- Min.Value -1, Min. Value, Min. Value+1, Nominal or Average Value, Max. Value-1, Max. Value, Max. Value +1
- It can be generalized that for an input variable in a module, 6n+1 test cases are designed with Robustness testing.

  6n+1→ 6(6) + 1 =37

  where n (number of variables) = 6

  Therefore, 37 testcases need to be designed for boundary value analysis.

Test matrix table (No extra toppings with regular toppings for non-veg pizza)

TESTCASE ID	NON-VEG SUPREME	INDI TIKKI CHICKEN	PEPPER BARBECUE CHICKEN	CHICKEN DOMINATOR	CHICKEN PEPPERNONI	CHICKEN FIESTA	EXPECTED OUTPUT
1	3	3	3	3	3	-1	X
2	3	3	3	3	3	0	5297
3	3	3	3	3	3	1	5685
4	3	3	3	3	3	3	6461
5	3	3	3	3	3	4	6848
6	3	3	3	3	3	5	7236
7	3	3	3	3	3	6	out of range(7462)
8	3	3	3	3	-1	3	X
9	3	3	3	3	0	3	5297
10	3	3	3	3	1	3	5685
11	3	3	3	3	4	3	6848
12	3	3	3	3	5	3	7236
13	3	3	3	3	6	3	out of range(14683)
14	3	3	3	-1	3	3	X
15	3	3	3	0	3	3	5654
16	3	3	3	1	3	3	5923

17	3	3	3	4	3	3	6729
18	3	3	3	5	3	3	6998
19	3	3	3	6	3	3	out of range(21742)
20	3	3	-1	3	3	3	X
21	3	3	0	3	3	3	5297
22	3	3	1	3	3	3	5685
23	3	3	4	3	3	3	6848
24	3	3	5	3	3	3	7236
25	3	3	6	3	3	3	out of range(27454)
26	3	-1	3	3	3	3	X
27	3	0	3	3	3	3	5297
28	3	1	3	3	3	3	5685
29	3	4	3	3	3	3	6848
30	3	5	3	3	3	3	7236
31	3	6	3	3	3	3	out of range(35860)
32	-1	3	3	3	3	3	X
33	0	3	3	3	3	3	5459
34	1	3	3	3	3	3	5793
35	4	3	3	3	3	3	6794
36	5	3	3	3	3	3	7128
37	6	3	3	3	3	3	out of range(42919)

TEST PLAN DOCUMENT, TEST LOG REPORT AND TEST INCIDENT REPORT FOR EACH MODULE:

LOGIN MODULE – RANDOM TESTING:

TEST LOG REPORT

- Description

  Login module which allows the user to access the billing system (accesses granted by admin only).
  Testing login module's username and password text field using random testing.

- Activity and Event Entries

  Mention the following:

  - Date: 1/04/23
  - Author of test: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - Testcase ID: 1 -10
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

TESTCASEID	INPUT		RESULT	Status
	USERNAME	PASSWORD		
1	ADMIN	ADMIN	(Allow access)	PASS
2	ADMIN	XYZ	(Don't allow access)	PASS
3	ADMIN	abc	(Don't allow access)	PASS
4	ADMIN	123	(Don't allow access)	PASS
5	ADMIN	Abc123	(Don't allow access)	PASS
6	admin	admin	(Don't allow access)	PASS
7	admin	xyz	(Don't allow access)	PASS
8	admin	ADMIN	(Don't allow access)	PASS
9	123	123	(Don't allow access)	PASS
10	xyz	abc	(Don't allow access)	PASS

  - Report any anomalous unexpected event before or after the execution.
  No anomalies detected.

TEST INCIDENT REPORT

- Incident Description

  It describes the following:

  - Date and Time: 1/04/23
  - Testing personnel names: Muthukumar S
  - Environment: Apache Netbeans (Junit)
  - Test IDs: 1 – 10 (all PASS)
  - Anomalies detected during the test: No anomalies
  - Attempts to repeat the same test: Same result

Testcase ID	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
1	Allow access	Allow access	No anomalies detected	Same result
2	Don't allow access	Don't allow access	No anomalies detected	Same result
3	Don't allow access	Don't allow access	No anomalies detected	Same result
4	Don't allow access	Don't allow access	No anomalies detected	Same result
5	Don't allow access	Don't allow access	No anomalies detected	Same result
6	Don't allow access	Don't allow access	No anomalies detected	Same result
7	Don't allow access	Don't allow access	No anomalies detected	Same result
8	Don't allow access	Don't allow access	No anomalies detected	Same result
9	Don't allow access	Don't allow access	No anomalies detected	Same result
10	Don't allow access	Don't allow access	No anomalies detected	Same result

VEG SECTION – BOUNDARY VALUE ANALYSIS TESTING USING Junit:

TEST PLAN COMPONENTS

- Introduction – To test a module of a Pizza Billing system.
- Test-Item to be tested Features to be tested – JtextField (6)
- Features not to be tested – veg quantity textfield

- Approach - Junit
- Item Pass/Fail Criteria – Amount generated should be accurate
- Environmental needs – Apache NetBeans Junit
- Risks and contingencies – If the amount displayed is incorrect it will cause inconvenience to the customers
- Testing costs – N/A

TEST LOG REPORT

- Description
  Veg section is the module where user entries veg pizza quantity of each pizza (if quantity = 0 leave it empty).
  To test veg section we use boundary value analysis.

- Activity and Event Entries

  - Date: 1/04/23
  - Author of test: Yagavi K
  - Testcase ID: 1 - 25
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

TESTCASE ID	MARGHERITA	PEPPY PANEER	PANEER MAKHANI	VEGGIE DELUXE	MEXICAN GREEN WAVE	INDI TANDOORI PANEER	EXPECTED OUTPUT	TEST CASE PASS / FAIL
1	3	3	3	3	3	0	3969	PASS
2	3	3	3	3	3	1	4292	PASS
3	3	3	3	3	3	3	4938	PASS
4	3	3	3	3	3	4	5261	PASS
5	3	3	3	3	3	5	5584	PASS
6	3	3	3	3	0	3	4099	PASS
7	3	3	3	3	1	3	4378	PASS
8	3	3	3	3	4	3	5217	PASS
9	3	3	3	3	5	3	5497	PASS
10	3	3	3	0	3	3	3969	PASS

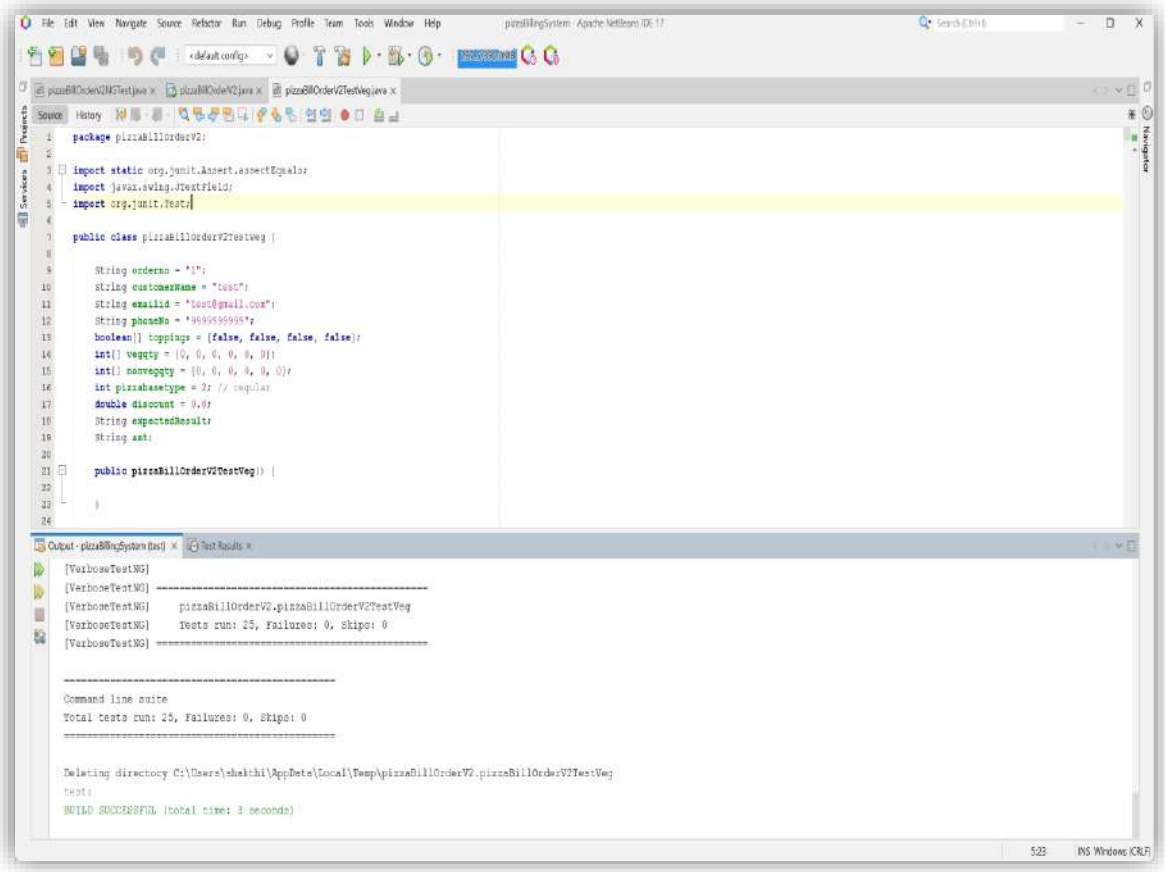
11	3	3	3	1	3	3	4292	PASS
12	3	3	3	4	3	3	5261	PASS
13	3	3	3	5	3	3	5584	PASS
14	3	3	0	3	3	3	3969	PASS
15	3	3	1	3	3	3	4292	PASS
16	3	3	4	3	3	3	5261	PASS
17	3	3	5	3	3	3	5584	PASS
18	3	0	3	3	3	3	4099	PASS
19	3	1	3	3	3	3	4378	PASS
20	3	4	3	3	3	3	5217	PASS
21	3	5	3	3	3	3	5497	PASS
22	0	3	3	3	3	3	4585	PASS
23	1	3	3	3	3	3	4702	PASS
24	4	3	3	3	3	3	5055	PASS
25	5	3	3	3	3	3	5173	PASS

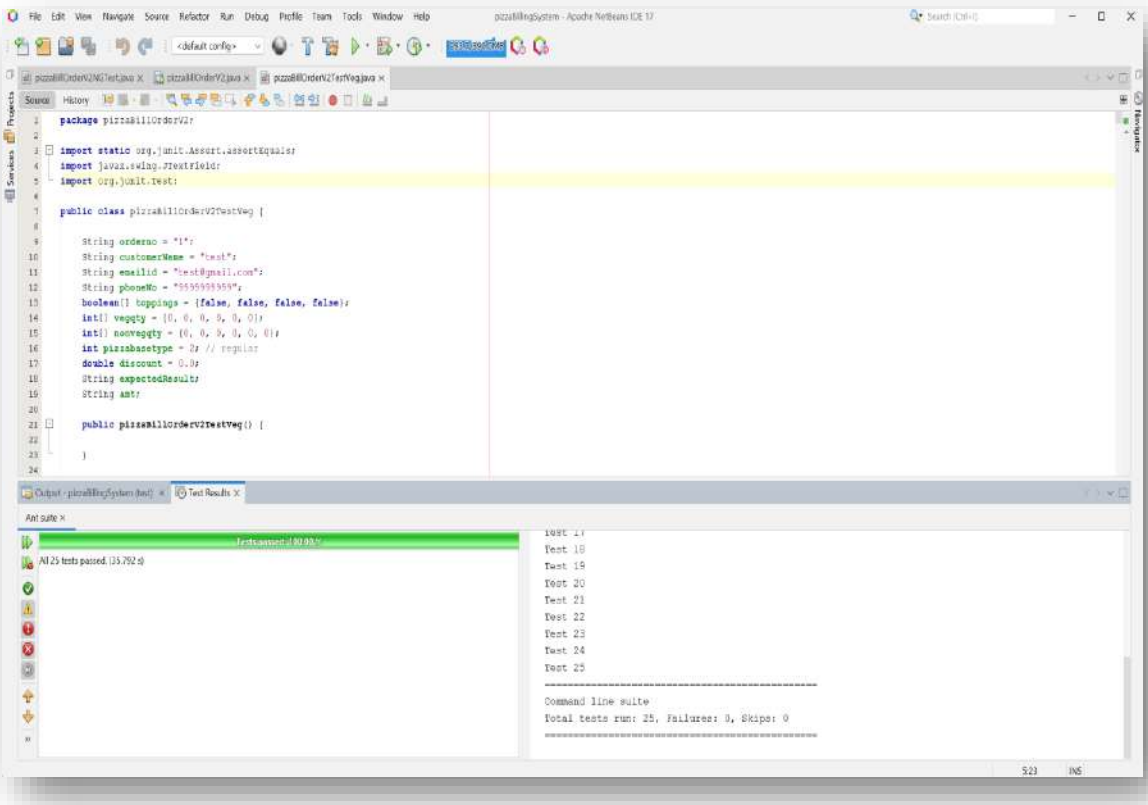
- Report any anomalous unexpected event before or after the execution.
  No anomalies.

TEST INCIDENT REPORT

- Test incident report identifier
- Summary
  No anomalies.
- Incident Description
  It describes the following:
  - Date and Time: 1/04/23
  - Testing personnel names: Yagavi K
  - Environment: Apache NetBeans Junit
  - Test IDs: 1 – 25 (all PASS)
  - Expected outputs: correct amount
  - Actual outputs: correct amount
  - Anomalies detected during the test: No anomalies
  - Attempts to repeat the same test: Same result

Test case ID	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
1 - 25	Correct Amount	Correct Amount	No anomalies	Same result





VEG SECTION – ROBUSTNESS TESTING USING TestNG

TEST PLAN COMPONENTS

- Introduction – To test a module of a Pizza Billing system.
- Test-Item to be tested Features to be tested – JtextField (6)
- Features not to be tested – veg quantity textfield
- Approach - TestNG
- Item Pass/Fail Criteria – Amount generated should be accurate
- Environmental needs – Apache NetBeans
- Risks and contingencies – If the amount displayed is incorrect it will cause inconvenience to the customers
- Testing costs – N/A

TEST LOG REPORT

- Description
  Veg section is the module where user entries veg pizza quantity of each pizza (if quantity = 0 leave it empty).
  To test veg section we use robustness testing.
- Activity and Event Entries
  - Date: 1/04/23
  - Author of test: Shakthi Shamruth D
  - Testcase ID: 1, 7, 8, 13, 14, 19, 20, 25, 26, 31, 32, 37 (FAIL)
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

TESTCASE ID	MARGHERITA	PEPPY PANEER	PANEER MAKHANI	VEGGIE DELUXE	MEXICAN GREEN WAVE	INDI TANDOORI PANEER	EXPECTED OUTPUT	TEST CASE PASSED / FAILED
1	3	3	3	3	3	-1	Out of range	FAIL
2	3	3	3	3	3	0	3969	PASS
3	3	3	3	3	3	1	4292	PASS
4	3	3	3	3	3	3	4938	PASS
5	3	3	3	3	3	4	5261	PASS
6	3	3	3	3	3	5	5584	PASS
7	3	3	3	3	3	6	out of range(5907)	FAIL
8	3	3	3	3	-1	3	Out of range	FAIL
9	3	3	3	3	0	3	4099	PASS
10	3	3	3	3	1	3	4378	PASS
11	3	3	3	3	4	3	5217	PASS
12	3	3	3	3	5	3	5497	PASS
13	3	3	3	3	6	3	out of range(11126)	FAIL
14	3	3	3	-1	3	3	Out of range	FAIL
15	3	3	3	0	3	3	3969	PASS
16	3	3	3	1	3	3	4292	PASS
17	3	3	3	4	3	3	5261	PASS
18	3	3	3	5	3	3	5584	PASS
19	3	3	3	6	3	3	out of range(16845)	FAIL
20	3	3	-1	3	3	3	Out of range	FAIL
21	3	3	0	3	3	3	3969	PASS
22	3	3	1	3	3	3	4292	PASS
23	3	3	4	3	3	3	5261	PASS
24	3	3	5	3	3	3	5584	PASS

25	3	3	6	3	3	3	out of range(22314)	FAIL
26	3	-1	3	3	3	3	Out of range	FAIL
27	3	0	3	3	3	3	4099	PASS
28	3	1	3	3	3	3	4378	PASS
29	3	4	3	3	3	3	5217	PASS
30	3	5	3	3	3	3	5497	PASS
31	3	6	3	3	3	3	out of range(27173)	FAIL
32	-1	3	3	3	3	3	Out of range	FAIL
33	0	3	3	3	3	3	4585	PASS
34	1	3	3	3	3	3	4702	PASS
35	4	3	3	3	3	3	5055	PASS
36	5	3	3	3	3	3	5173	PASS
37	6	3	3	3	3	3	out of range(29786)	FAIL

- Report any anomalous unexpected event before or after the execution.

Anomalies detected. No boundary set for quantity.

TEST INCIDENT REPORT

- Summary

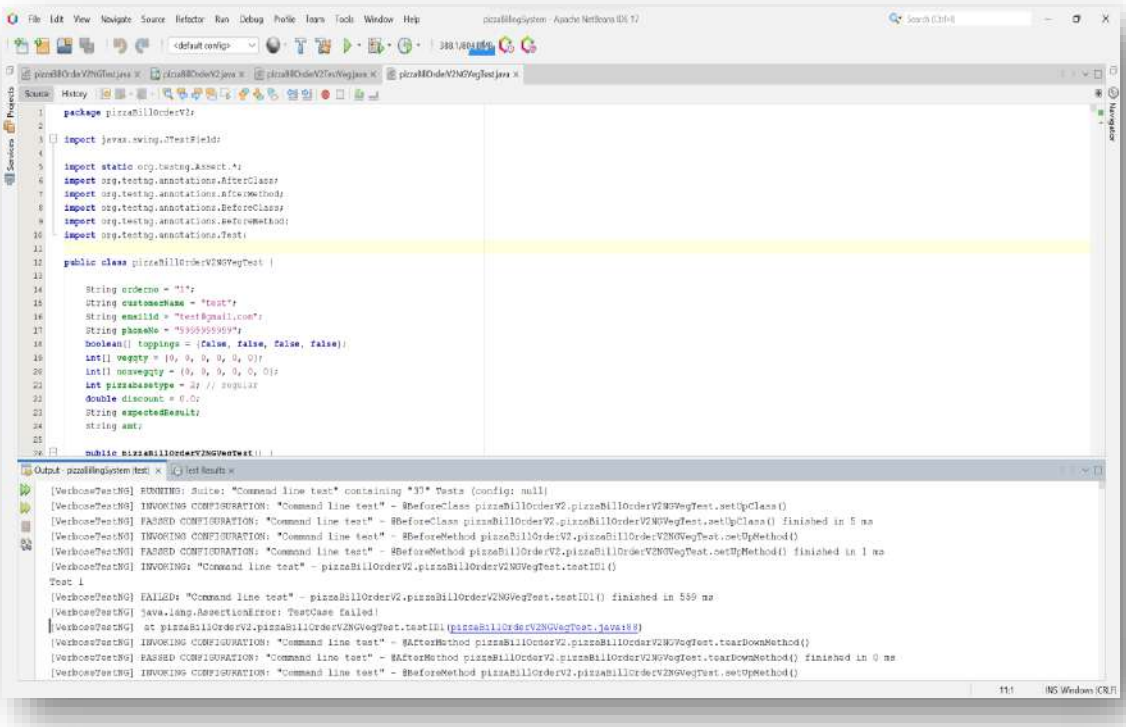
  When giving out of range inputs instead of showing error or prompting error application calculated incorrect amount.

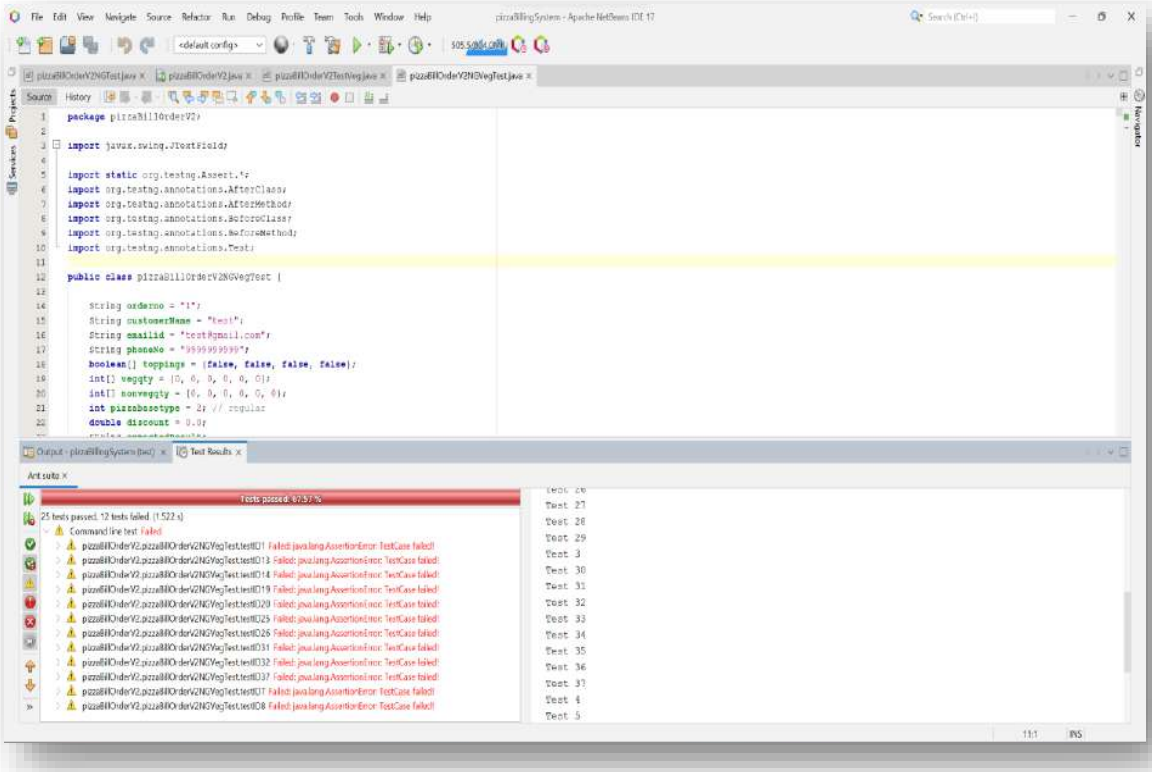
- Incident Description

  It describes the following:

- Date and Time: 1/04/23
- Testing personnel names: Shakthi Shamruth D
- Environment: Apache NetBeans TestNG
- Test IDs: 1, 7, 8, 13, 14, 19, 20, 25, 26, 31, 32, 37 (FAIL)
- Expected outputs: Amount should be invalid for the above test cases.
- Actual outputs: Amount generated is a negative number.
- Anomalies detected during the test: The amount to be paid is a negative number, No boundary set for the quantity inputs.
- Attempts to repeat the same test: Same result

Test case ID	Expected outputs (Amount)	Anomalies detected	Attempts to repeat the same test
1	Invalid Input	No boundary set for quantity textfield	Same result
7	Invalid Input	No boundary set for quantity textfield	Same result
8	Invalid Input	No boundary set for quantity textfield	Same result
13	Invalid Input	No boundary set for quantity textfield	Same result
14	Invalid Input	No boundary set for quantity textfield	Same result
19	Invalid Input	No boundary set for quantity textfield	Same result
20	Invalid Input	No boundary set for quantity textfield	Same result
25	Invalid Input	No boundary set for quantity textfield	Same result
26	Invalid Input	No boundary set for quantity textfield	Same result
31	Invalid Input	No boundary set for quantity textfield	Same result
32	Invalid Input	No boundary set for quantity textfield	Same result





- Impact
  As invalid amount is generated, no system crashes is experienced.

- Solution
  Prompt the user to give valid inputs.

NON-VEG SECTION – BOUNDARY VALUE ANALYSIS USING JUnit

TEST PLAN COMPONENTS

- Introduction – To test a module of a Pizza Billing system.
- Test-Item to be tested Features to be tested – JtextField (6)
- Features not to be tested – non veg quantity textfield
- Approach - Junit
- Item Pass/Fail Criteria – Amount generated should be accurate
- Environmental needs – Apache NetBeans Junit
- Risks and contingencies – If the amount displayed is incorrect it will cause inconvenience to the customers
- Testing costs – N/A

TEST LOG REPORT

- Description
  Non-Veg section is the module where user entries non veg pizza quantity of each pizza (if quantity = 0 leave it empty).
  To test non veg section we use boundary value analysis.

- Activity and Event Entries
  - Date: 1/04/23
  - Author of test: Yagavi K
  - Testcase ID: 1 - 25
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

TESTCASE ID	NON-VEG SUPREME	INDI TIKKI CHICKEN	PEPPER BARBECUE CHICKEN	CHICKEN DOMINATOR	CHICKEN PEPPERNONI	CHICKEN FIESTA	EXPECTED OUTPUT	TEST CASE PASS / FAIL
1	3	3	3	3	3	0	5459	PASS
2	3	3	3	3	3	1	5793	PASS
3	3	3	3	3	3	3	6461	PASS
4	3	3	3	3	3	4	6794	PASS
5	3	3	3	3	3	5	7128	PASS
6	3	3	3	3	0	3	5297	PASS
7	3	3	3	3	1	3	5685	PASS
8	3	3	3	3	4	3	6848	PASS
9	3	3	3	3	5	3	7236	PASS
10	3	3	3	0	3	3	5297	PASS
11	3	3	3	1	3	3	5685	PASS
12	3	3	3	4	3	3	6848	PASS
13	3	3	3	5	3	3	7236	PASS
14	3	3	0	3	3	3	5654	PASS
15	3	3	1	3	3	3	5923	PASS
16	3	3	4	3	3	3	6729	PASS
17	3	3	5	3	3	3	6998	PASS
18	3	0	3	3	3	3	5297	PASS

19	3	1	3	3	3	3	5685	PASS
20	3	4	3	3	3	3	6848	PASS
21	3	5	3	3	3	3	7236	PASS
22	0	3	3	3	3	3	5297	PASS
23	1	3	3	3	3	3	5685	PASS
24	4	3	3	3	3	3	6848	PASS
25	5	3	3	3	3	3	7236	PASS

- Report any anomalous unexpected event before or after the execution.

No anomalies.

TEST INCIDENT REPORT

- Summary

  When giving out of range inputs instead of showing error or prompting error application calculated incorrect amount.

- Incident Description

  It describes the following:

- Date and Time: 1/04/23
- Testing personnel names: Shakthi Shamruth D
- Environment: Apache Junit
- Expected outputs: correct amount
- Actual outputs: correct amount
- Anomalies detected during the test: No Anomalies
- Attempts to repeat the same test: Same result

Test case ID	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
1 - 25	Correct Amount	Correct Amount	No anomalies	Same result

```
package pizzaBillOrderV2;

import static org.junit.Assert.assertEquals;
import javax.swing.JTextField;
import org.junit.Test;

public class pizzaBillOrderV2TestNonVeg {

    String orderno = "1";
    String customerName = "test";
    String emailid = "test@gmail.com";
    String phoneNo = "5999999999";
    boolean[] toppings = {false, false, false, false};
    int[] veqqty = {0, 0, 0, 0, 0, 0};
    int[] nonveqqty = {0, 0, 0, 0, 0, 0};
    int pizzabasetype = 2; // regular
    double discount = 0.0;
    String expectedResult;
    String amt;

    public pizzaBillOrderV2TestNonVeq() {
```

```
[VerboseTestNG] INVOKING: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID22()
Test 22
[VerboseTestNG] PASSED: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID22() finished in 1256 ms
[VerboseTestNG] INVOKING: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID23()
Test 23
[VerboseTestNG] PASSED: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID23() finished in 1287 ms
[VerboseTestNG] INVOKING: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID24()
Test 24
[VerboseTestNG] PASSED: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID24() finished in 1319 ms
[VerboseTestNG] INVOKING: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID25()
Test 25
[VerboseTestNG] PASSED: "pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq" - pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq.testID25() finished in 1360 ms
[VerboseTestNG]
[VerboseTestNG] ===============================================
[VerboseTestNG]     pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq
[VerboseTestNG]     Tests run: 25, Failures: 0, Skips: 0
[VerboseTestNG] ===============================================

===============================================
Command line suite
Total tests run: 25, Failures: 0, Skips: 0
===============================================

Deleting directory C:\Users\shakthi\AppData\Local\Temp\pizzaBillOrderV2.pizzaBillOrderV2TestNonVeq
test:
BUILD SUCCESSFUL (total time: 2 seconds)
```

NON-VEG SECTION – ROBUSTNESS TESTING USING TestNG

TEST PLAN COMPONENTS

- Introduction – To test a module of a Pizza Billing system.
- Test-Item to be tested Features to be tested – JtextField (6)
- Features not to be tested – veg quantity textfield
- Approach - TestNG
- Item Pass/Fail Criteria – Amount generated should be accurate
- Suspension criteria and resumption requirements
- Environmental needs – Apache NetBeans TestNG
- Risks and contingencies – If the amount displayed is incorrect it will cause inconvenience to the customers
- Testing costs – N/A

TEST LOG REPORT

- Description
  Non-Veg section is the module where user entries non veg pizza quantity of each pizza (if quantity = 0 leave it empty).
  To test non veg section we use robustness testing.
- Activity and Event Entries
  - Date: 1/04/23
  - Author of test: Shakthi Shamruth D
  - Testcase ID: 1, 7, 8, 13, 14, 19, 20, 25, 26, 31, 32, 37 (FAIL)
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

TESTCASE ID	NON-VEG SUPREME	INDI TIKKI CHICKEN	PEPPER BARBECUE CHICKEN	CHICKEN DOMINATOR	CHICKEN PEPPERNONI	CHICKEN FIESTA	EXPECTED OUTPUT	TEST CASE PASS/FAIL
1	3	3	3	3	3	-1	X	FAIL
2	3	3	3	3	3	0	5459	PASS
3	3	3	3	3	3	1	5793	PASS
4	3	3	3	3	3	3	6461	PASS
5	3	3	3	3	3	4	6794	PASS
6	3	3	3	3	3	5	7128	PASS
7	3	3	3	3	3	6	out of range(7462)	FAIL
8	3	3	3	3	-1	3	X	FAIL

9	3	3	3	3	0	3	5297	PASS
10	3	3	3	3	1	3	5685	PASS
11	3	3	3	3	4	3	6848	PASS
12	3	3	3	3	5	3	7236	PASS
13	3	3	3	3	6	3	out of range(14683)	FAIL
14	3	3	3	-1	3	3	X	FAIL
15	3	3	3	0	3	3	5297	PASS
16	3	3	3	1	3	3	5685	PASS
17	3	3	3	4	3	3	6848	PASS
18	3	3	3	5	3	3	7236	PASS
19	3	3	3	6	3	3	out of range(21742)	FAIL
20	3	3	-1	3	3	3	X	FAIL
21	3	3	0	3	3	3	5654	PASS
22	3	3	1	3	3	3	5923	PASS
23	3	3	4	3	3	3	6729	PASS
24	3	3	5	3	3	3	6998	PASS
25	3	3	6	3	3	3	out of range(27454)	FAIL
26	3	-1	3	3	3	3	X	FAIL
27	3	0	3	3	3	3	5297	PASS
28	3	1	3	3	3	3	5685	PASS
29	3	4	3	3	3	3	6848	PASS
30	3	5	3	3	3	3	7236	PASS
31	3	6	3	3	3	3	out of range(35860)	FAIL
32	-1	3	3	3	3	3	X	FAIL
33	0	3	3	3	3	3	5297	PASS
34	1	3	3	3	3	3	5685	PASS
35	4	3	3	3	3	3	6848	PASS
36	5	3	3	3	3	3	7236	PASS
37	6	3	3	3	3	3	out of range(42919)	FAIL

TEST INCIDENT REPORT

- Test incident report identifier
- Summary

  When giving out of range inputs instead of showing error or prompting error application calculated incorrect amount.

- Incident Description

  It describes the following:

  - Date and Time: 1/04/23
  - Testing personnel names: Shakthi Shamruth D

- Environment: Apache TestNG
- Test IDs: 1, 7, 8, 13, 14, 19, 20, 25, 26, 31, 32, 37 (FAIL)
- Expected outputs: Amount should be invalid for the above test cases.
- Actual outputs: Amount generated is a negative number.
- Anomalies detected during the test: The amount to be paid is a negative number, no boundary set for the quantity inputs.
- Attempts to repeat the same test: same result

Test case ID	Expected outputs (Amount)	Anomalies detected	Attempts to repeat the same test
1	Invalid Input	No boundary set for quantity textfield	Same result
7	Invalid Input	No boundary set for quantity textfield	Same result
8	Invalid Input	No boundary set for quantity textfield	Same result
13	Invalid Input	No boundary set for quantity textfield	Same result
14	Invalid Input	No boundary set for quantity textfield	Same result
19	Invalid Input	No boundary set for quantity textfield	Same result
20	Invalid Input	No boundary set for quantity textfield	Same result
25	Invalid Input	No boundary set for quantity textfield	Same result
26	Invalid Input	No boundary set for quantity textfield	Same result
31	Invalid Input	No boundary set for quantity textfield	Same result
32	Invalid Input	No boundary set for quantity textfield	Same result

- Impact
  As invalid amount is generated, no system crashes is experienced.

- Solution
  Prompt the user to give valid inputs.

```
package pizzaBillOrderV2;

import javax.swing.JTextField;

import static org.testng.Assert.*;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class pizzaBillOrderV2NGNonVegTest {

    String orderno = "1";
    String customerName = "test";
    String emailid = "test@gmail.com";
    String phoneNo = "9999999999";
    boolean[] toppings = {false, false, false, false};
    int[] veggty = {0, 0, 0, 0, 0, 0};
    int[] nonveggty = {0, 0, 0, 0, 0, 0};
    int pizzabasetype = 2; // regular
    double discount = 0.0;
    String expectedResult;
    String amt;

    public pizzaBillOrderV2NGNonVegTest() {
```

```
[VerboseTestNG] RUNNING: Suite: "Command line test" containing "37" Tests (config: null)
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeClass pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.setUpClass()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeClass pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.setUpClass() finished in 9 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.setUpMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @BeforeMethod pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.setUpMethod() finished in 2 ms
[VerboseTestNG] INVOKING: "Command line test" - pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID1()
Test 1
[VerboseTestNG] FAILED: "Command line test" - pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID1() finished in 901 ms
[VerboseTestNG] java.lang.AssertionError: TestCase failed!
[VerboseTestNG]  at pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID1(pizzaBillOrderV2NGNonVegTest.java:98)
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @AfterMethod pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.tearDownMethod()
[VerboseTestNG] PASSED CONFIGURATION: "Command line test" - @AfterMethod pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.tearDownMethod() finished in 1 ms
[VerboseTestNG] INVOKING CONFIGURATION: "Command line test" - @BeforeMethod pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.setUpMethod()
```

```
package pizzaBillOrderV2;

import javax.swing.JTextField;

import static org.testng.Assert.*;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class pizzaBillOrderV2NGNonVegTest {

    String orderno = "1";
    String customerName = "test";
    String emailid = "test@gmail.com";
    String phoneNo = "9999999999";
    boolean[] toppings = {false, false, false, false};
    int[] veggty = {0, 0, 0, 0, 0, 0};
    int[] nonveggty = {0, 0, 0, 0, 0, 0};
    int pizzabasetype = 2; // regular
    double discount = 0.0;
    String expectedResult;
    String amt;

    public pizzaBillOrderV2NGNonVegTest() {
```

```
Tests passed: 67.57 %
25 tests passed, 12 tests failed. (2.307 s)
Command line test  Failed
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID1  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID13  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID14  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID19  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID20  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID25  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID31  Failed: java.lang.AssertionError: TestCase failed!
pizzaBillOrderV2.pizzaBillOrderV2NGNonVegTest.testID32  Failed: java.lang.AssertionError: TestCase failed!
```

```
Test 34
Test 35
Test 36
Test 37
Test 4
Test 5
Test 6
Test 7
Test 8
Test 9
---------------------------------------------
Command line suite
```

BILLING SECTION – DECISION TABLE TESTING USING TestNG

TEST PLAN COMPONENTS

- Test Plan Identifier
- Introduction - To test billing module of pizza billing system which calculates total amount the customer needs to pay including the conditions like discount, pizza base price,
- Test-Item to be tested - JtextField (Customer name, email ID, Phone No, pizza base type, coupons and discounts)
- Features to be tested - Billing module
- Approach - Decision based testing TestNG
- Item Pass/Fail Criteria - Pass criteria the amount generated by the billing module should be accurate
- Environmental needs - Apache NetBeans TestNG
- Testing costs – N/A

TEST LOG REPORT

- Description

  Billing section is the module which generates total amount to be paid by the customer including the description of the customer's name, their email ID, phone no, order no, items selected and total amount to be paid.
- Activity and Event Entries

  Mention the following:

  - Date: 1/04/23
  - Author of test: Muthukumar S
  - Testcase ID: 1 to 8
  - Name of the personnel involved in testing: Yagavi K, Pranay Gorantla, Shakthi Shamruth D, Muthukumar S
  - For each execution, record the results and mention pass/fail status. The function was tested with the following inputs:

CONDITION	RULE 1	RULE 2	RULE 3	RULE 4	RULE 5	RULE 6	RULE 7	RULE 8
Email ID	T	T	T	T	F	F	F	F
Phone Number	T	T	F	F	T	T	F	F
Customer Name	X	X	X	X	X	X	X	X

Pizza base type	T	F	T	F	T	F	T	F
Extra Toppings	X	X	X	X	X	X	X	X
Coupon	X	X	X	X	X	X	X	X
Action-Result	T	F	F	F	T	F	F	F
Test Case Passed/Failed	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS

- Report any anomalous unexpected event before or after the execution.
  No anomalous

TEST INCIDENT REPORT

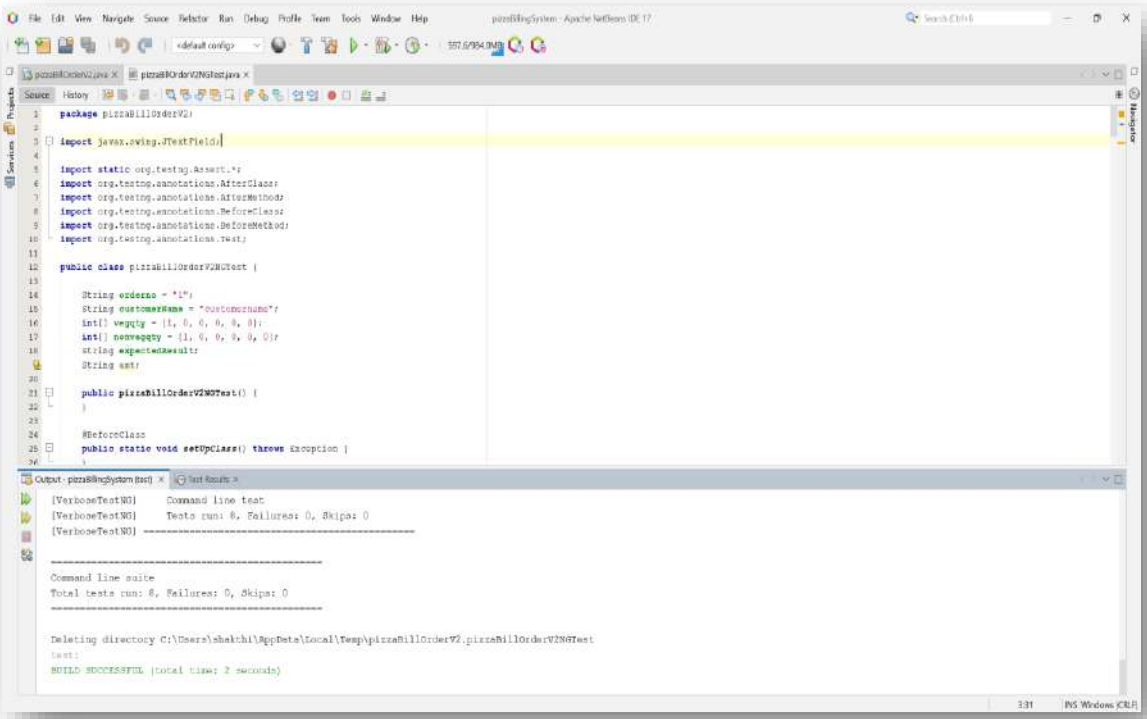
- Summary
  All test cases passed.

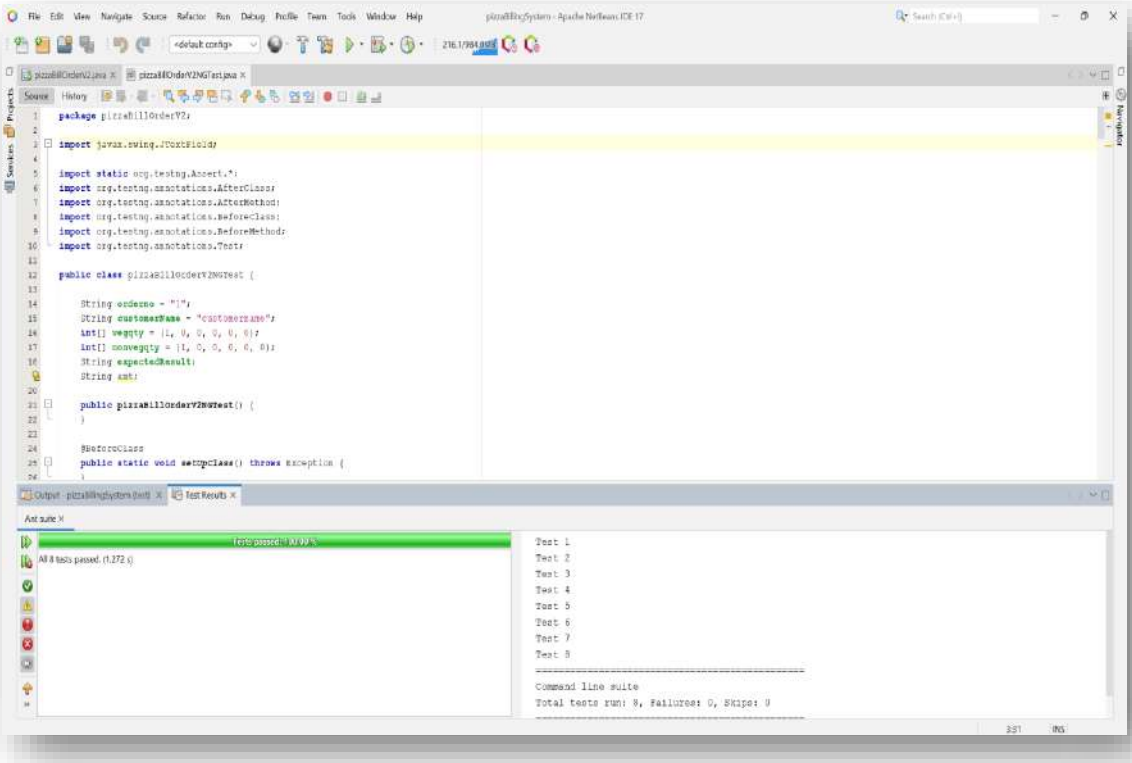
- Incident Description
  It describes the following:
  - Date and Time: 1/04/23
  - Testing personnel names: Shakthi Shamruth D
  - Environment: Apache TestNG
  - Test IDs: 1 to 8
  - Expected outputs: correct amount
  - Actual outputs: correct amount
  - Anomalies detected during the test: No anomalies
  - Attempts to repeat the same test

Test case ID	Inputs	Expected outputs	Actual outputs	Anomalies detected	Attempts to repeat the same test
1	abcd@gmail.com, 9999999999, toppings = {false, false, false, false} pizzabasetype = 2, discount = 0.0	Hellocustomername, Your Order Id is: 1 Email ID: abcd@gmail.com Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are:	Hellocustomername, Your Order Id is: 1 Email ID: abcd@gmail.com Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are:	No anomalies detected	Same result

		Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	Non veg Supreme nonveg pizza :1 Margherita veg pizza :1		
2	abcd@gmail.com, 9999999999, toppings = {false, false, false, false} pizzabasetype = 4, discount = 0.0	Hellocustomername, Your Order Id is: 1 Email ID: abcd@gmail.com Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	Hellocustomername, Your Order Id is: 1 Email ID: abcd@gmail.com Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	No anomalies detected	Same result
3	abcd@gmail.com, 99999, toppings = {false, false, false, false} pizzabasetype = 2, discount = 0.0	Bill Not Generated	Bill Not Generated	No anomalies detected	Same result
4	abcd@gmail.com, 99999, toppings = {false, false, false, false} pizzabasetype = 4, discount = 0.0	Bill Not Generated	Bill Not Generated	No anomalies detected	Same result
5	wrongemail, 9999999999, toppings = {false, false, false, false} pizzabasetype = 2, discount = 0.0	Hellocustomername, Your Order Id is: 1 Email ID: **InvalidEmailID Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	Hellocustomername, Your Order Id is: 1 Email ID: **InvalidEmailID Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0 The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	No anomalies detected	Same result
6	wrongemail, 9999999999, toppings = {false, false, false, false} pizzabasetype = 4, discount = 0.0	Hellocustomername, Your Order Id is: 1 Email ID: **InvalidEmailID Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0	Hellocustomername, Your Order Id is: 1 Email ID: **InvalidEmailID Phone number: 9999999999 AMOUNT PAYABLE IS: 505.0	No anomalies detected	Same result

		The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1	The list of items ordered are: Non veg Supreme nonveg pizza :1 Margherita veg pizza :1		
7	wrongemail, 99999, toppings = {false, false, false, false} pizzabasetype = 2, discount = 0.0	Bill Not Generated	Bill Not Generated	No anomalies detected	Same result
8	wrongemail, 99999, toppings = {false, false, false, false} pizzabasetype = 4, discount = 0.0	Bill Not Generated	Bill Not Generated	No anomalies detected	Same result

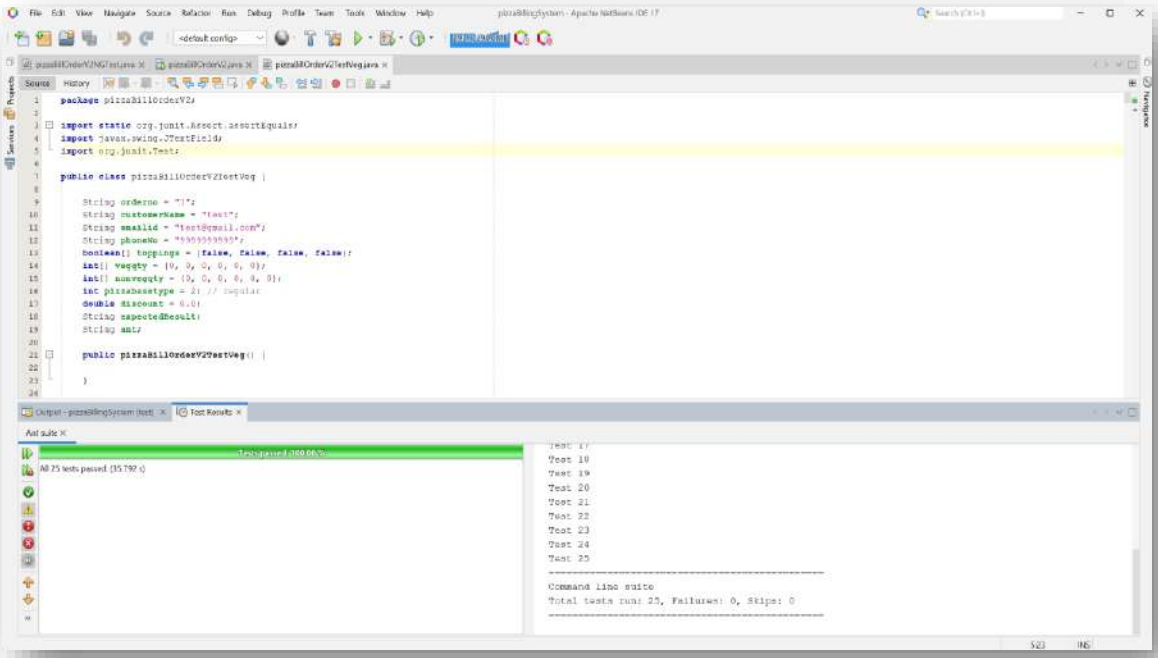




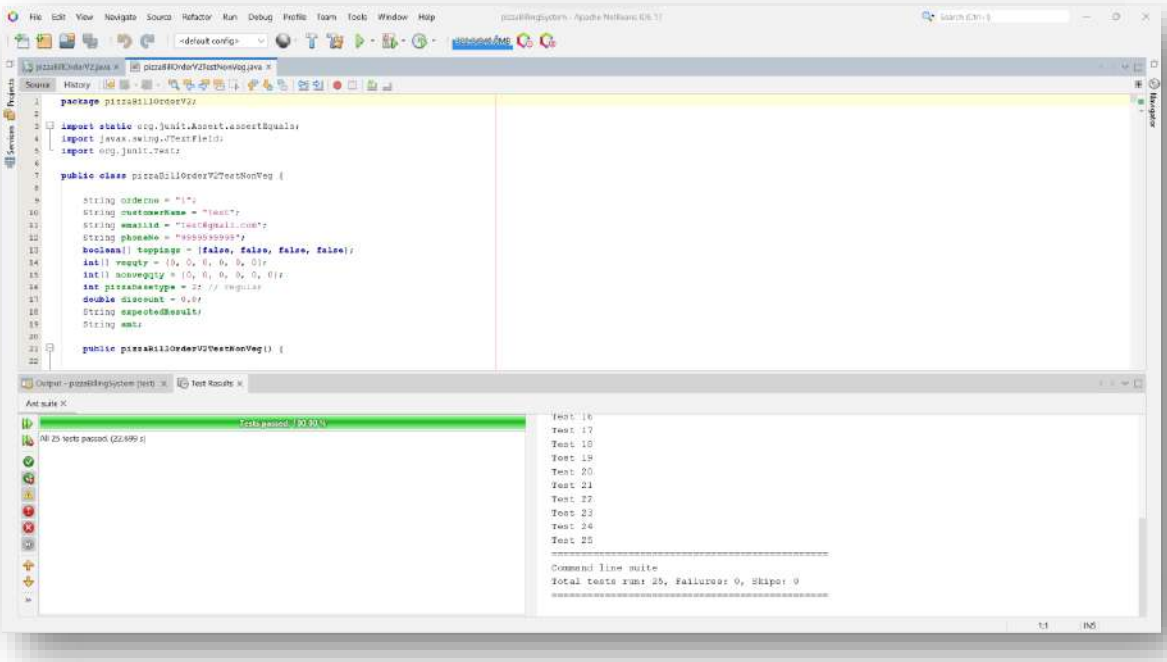
FUNCTIONAL TESTING:

i) Using junit:

Veg Module

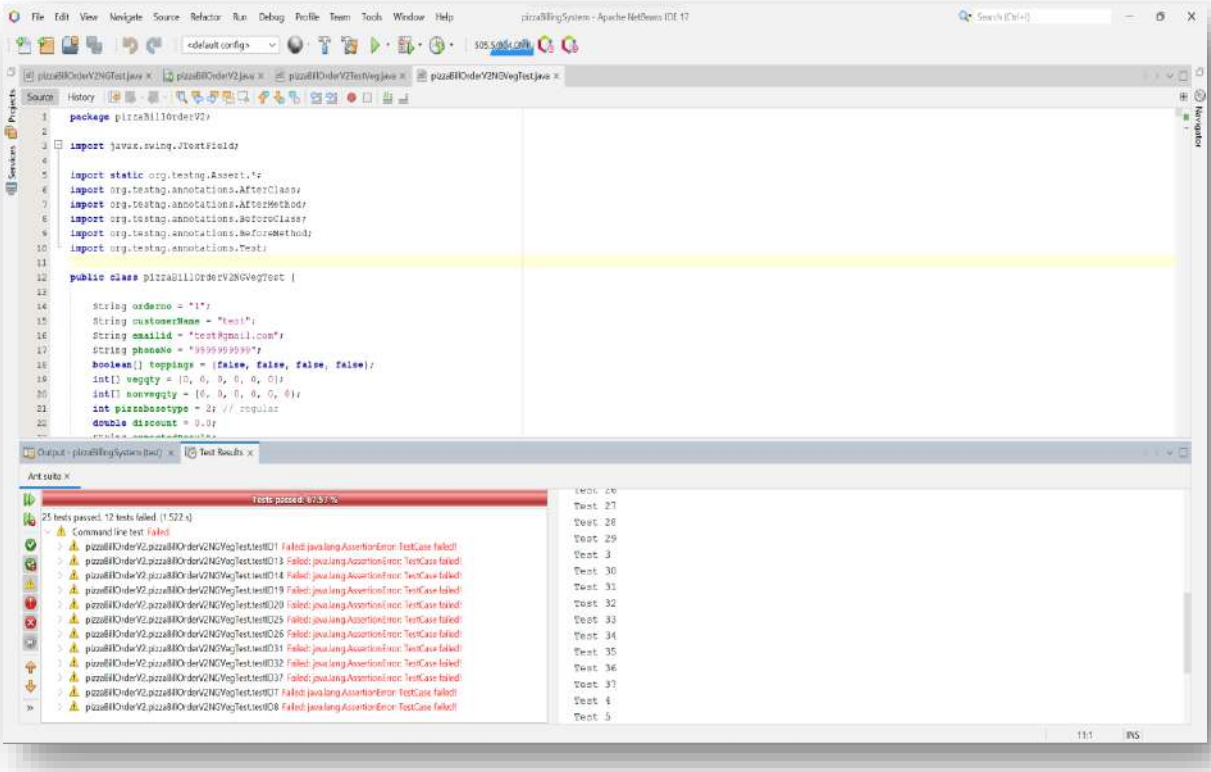


Non-veg Module

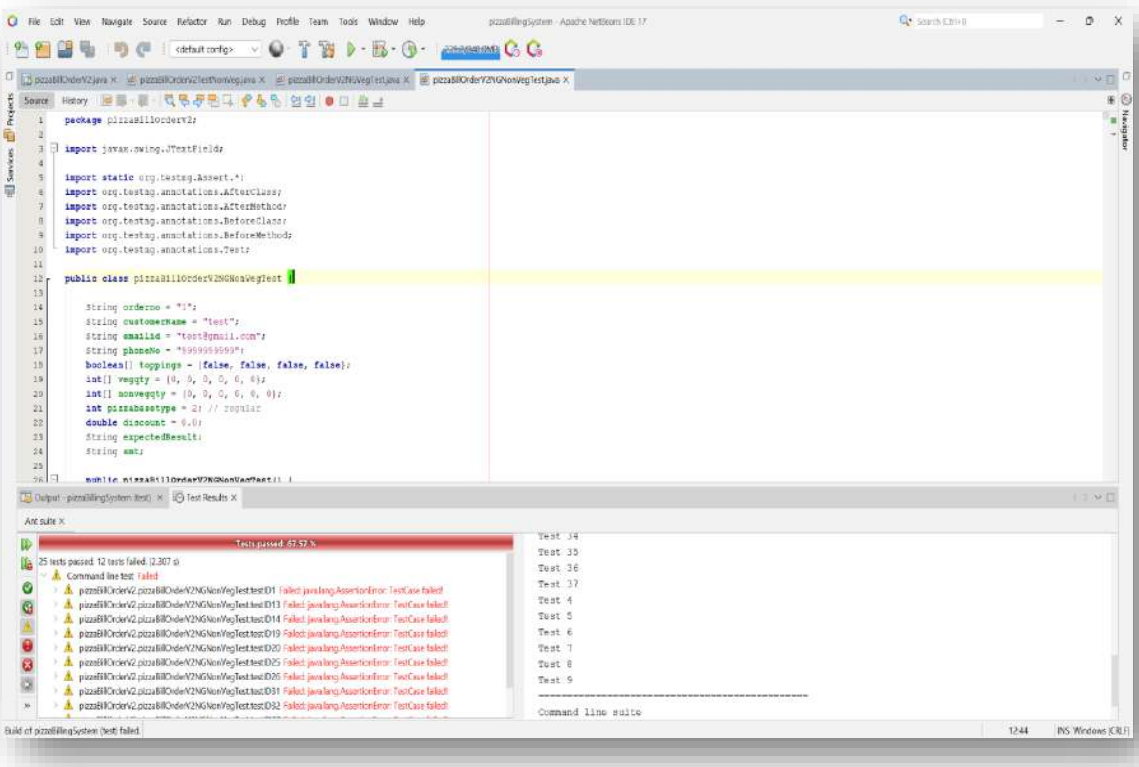


ii) Using TestNG:

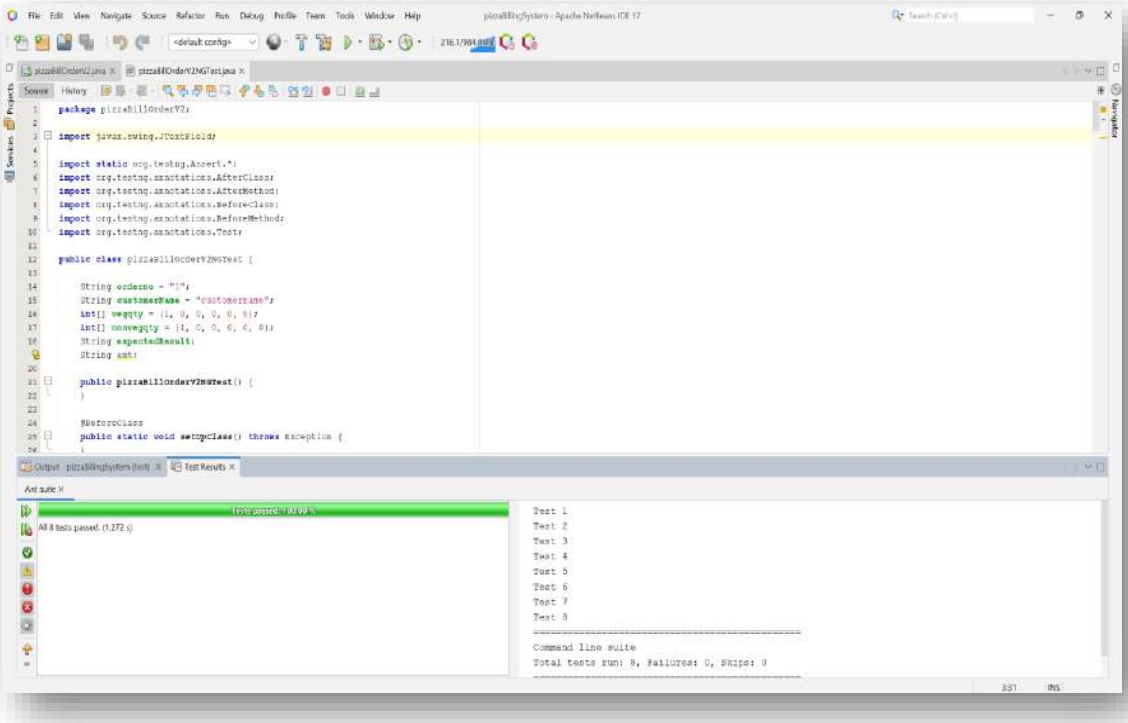
Veg Module



Non-veg Module



Billing Module

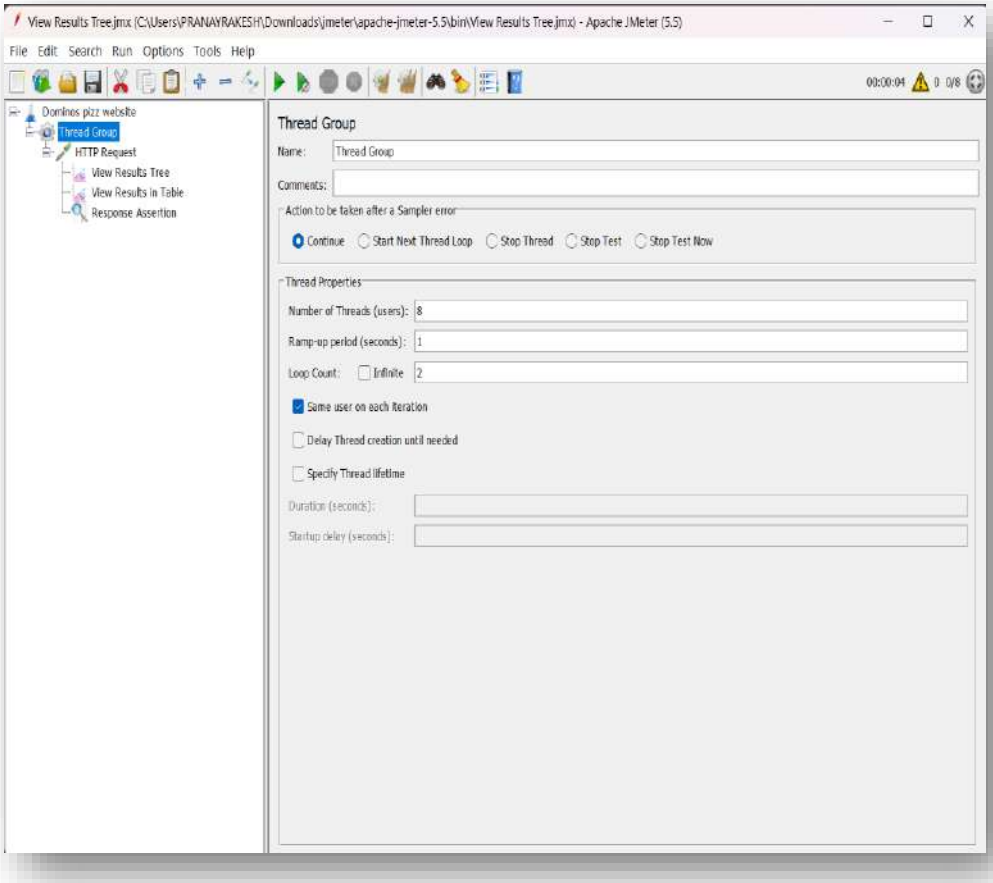


NON-FUNCTIONAL TESTING:

Load Testing using jmeter

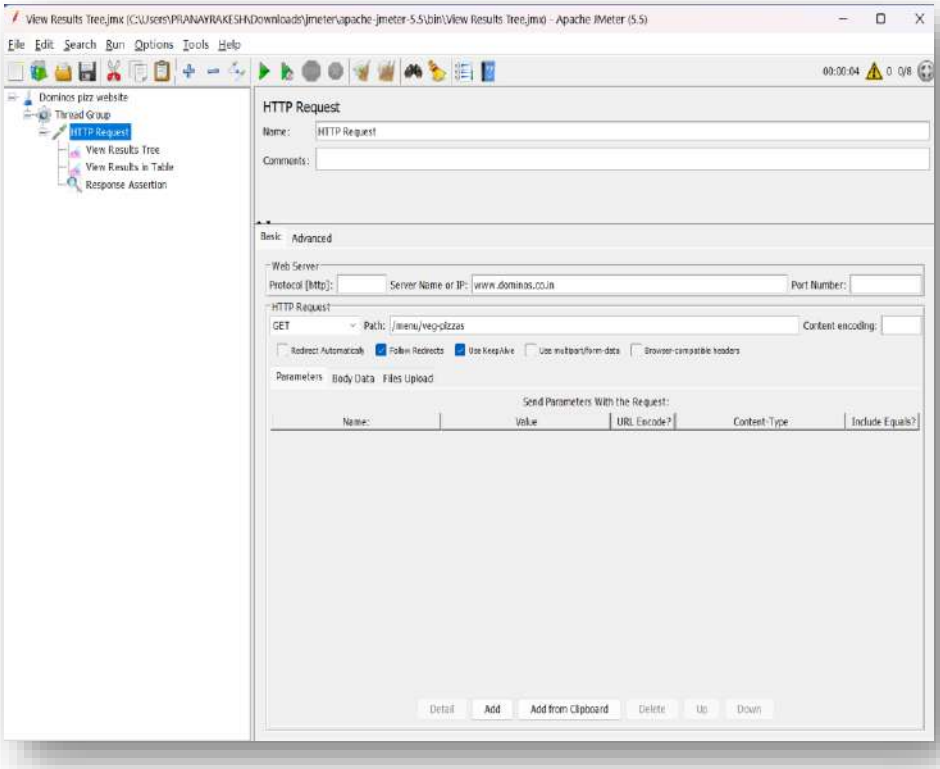
Configuring the load testing parameter in Jmeter:

Setting the number of threads accessing the dominos website:



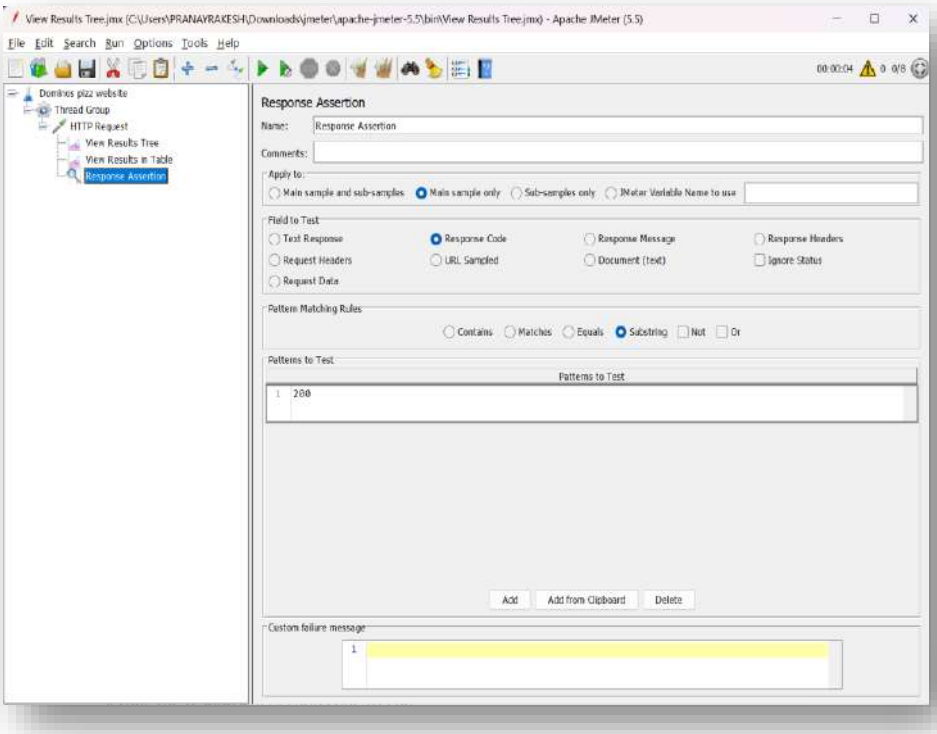
For this instance, we are setting the thread users to 8 and number of times they are accessing the website is twice.

Setting the server's name or IP to dominos website and endpoint of dominos website to /menus/vegpizza i.e (veg pizza menus)

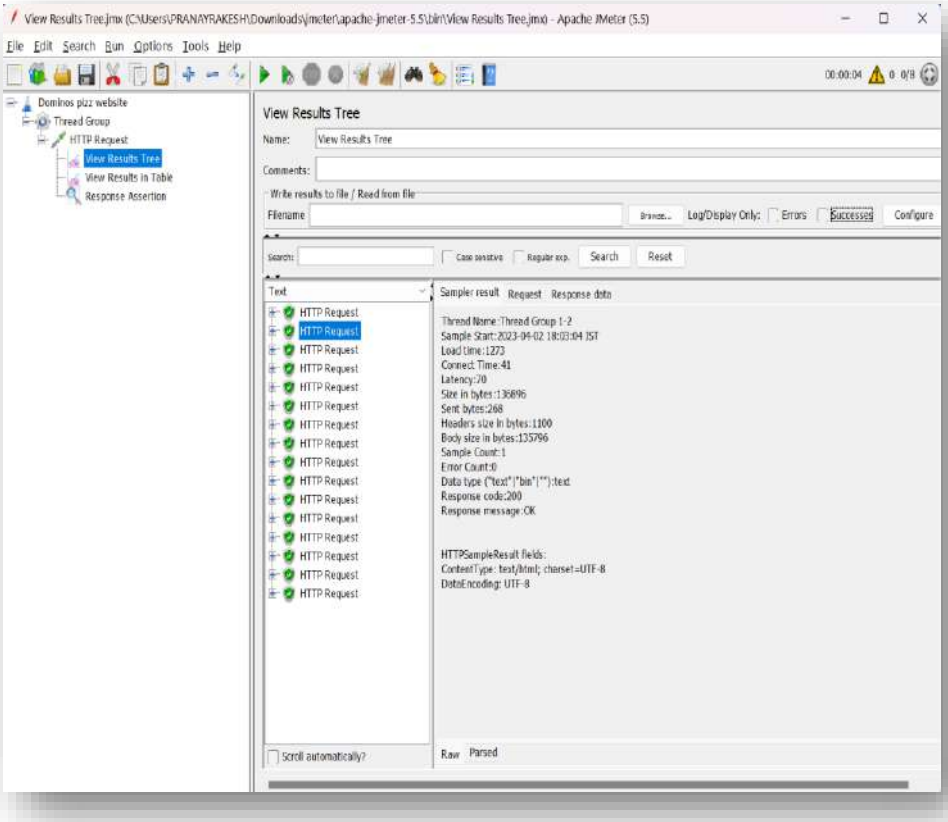


Setting the response assertion code to 200:

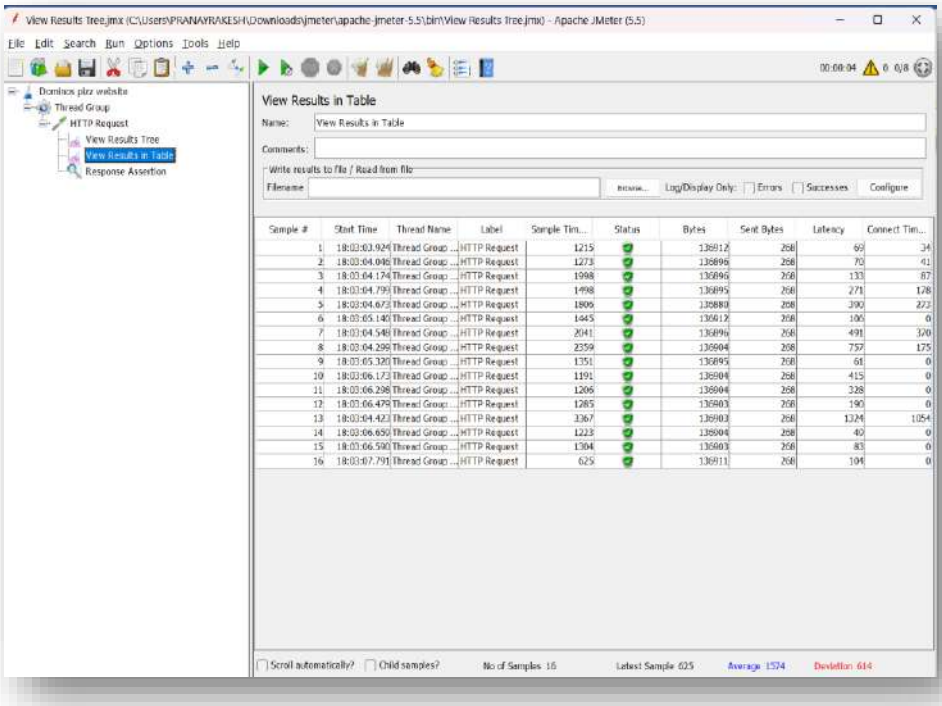
This parameter helps us to accept only those responses as successful testcase which have a respond code to our request as 200.



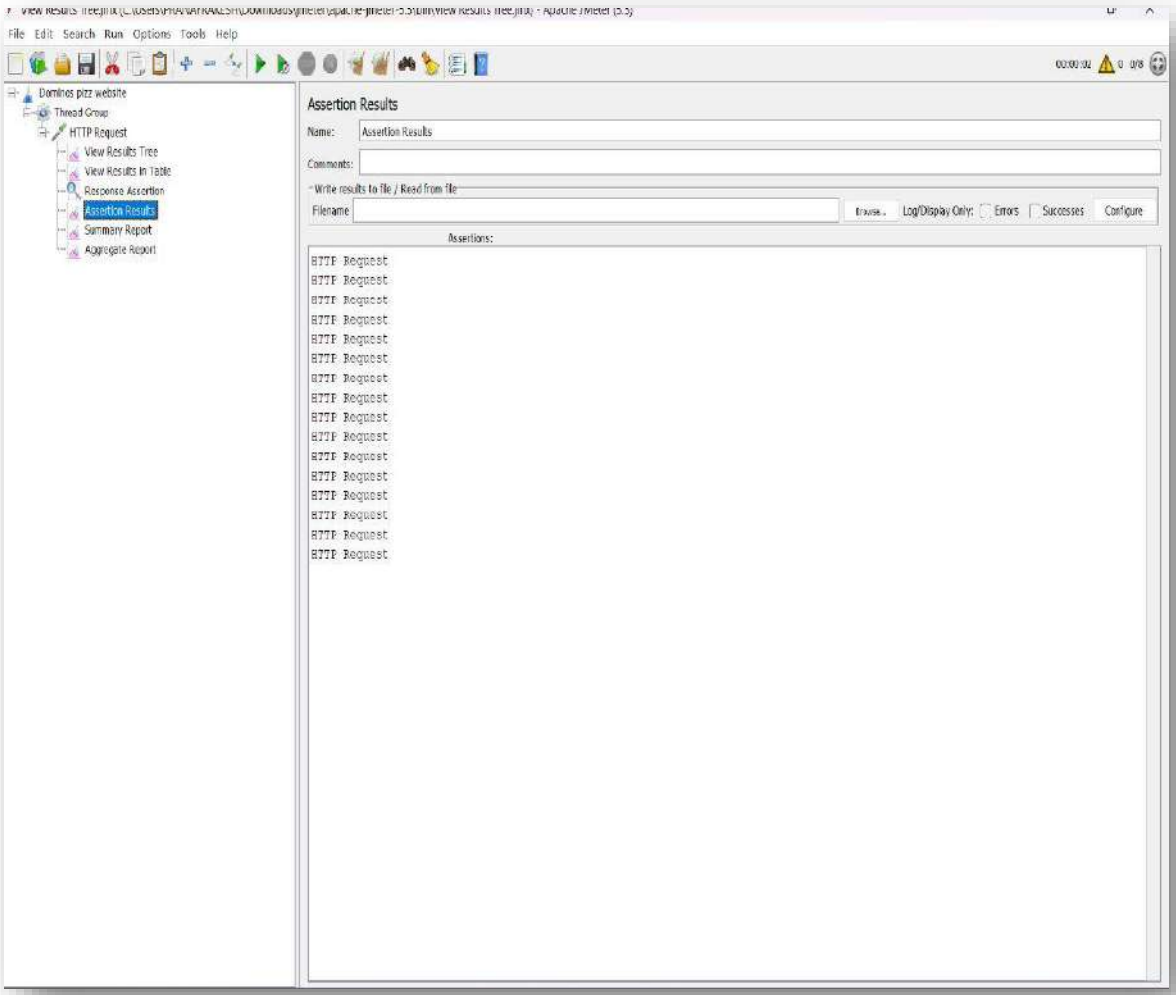
Viewing the results in result tree:

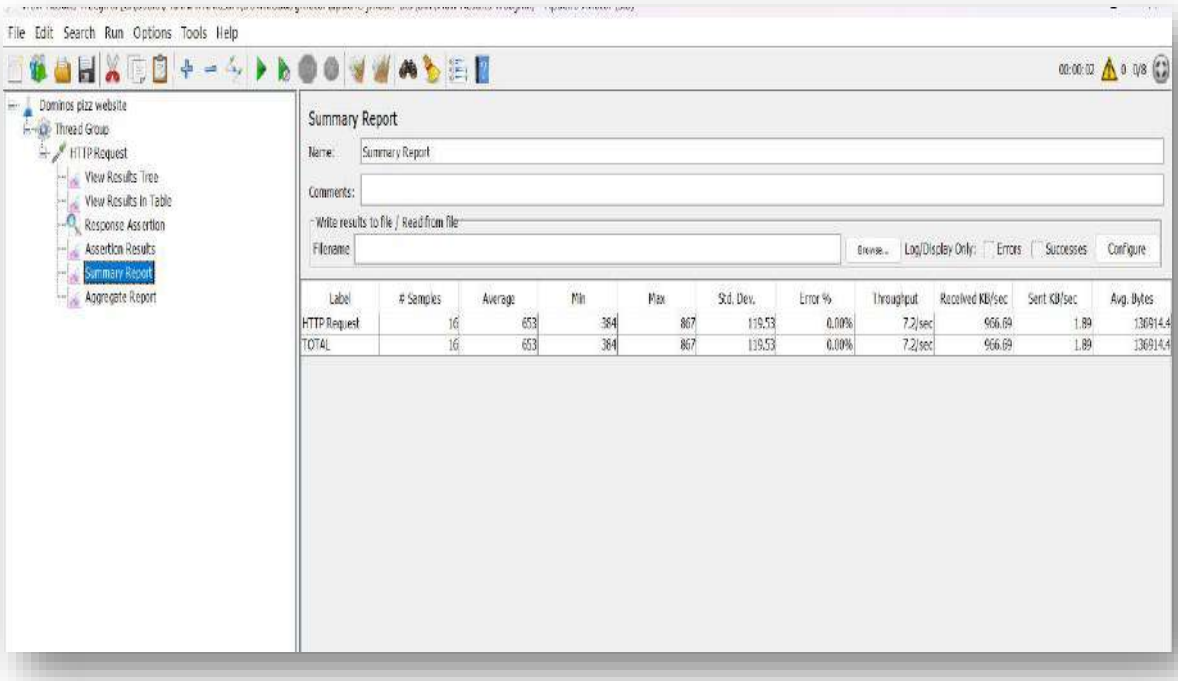


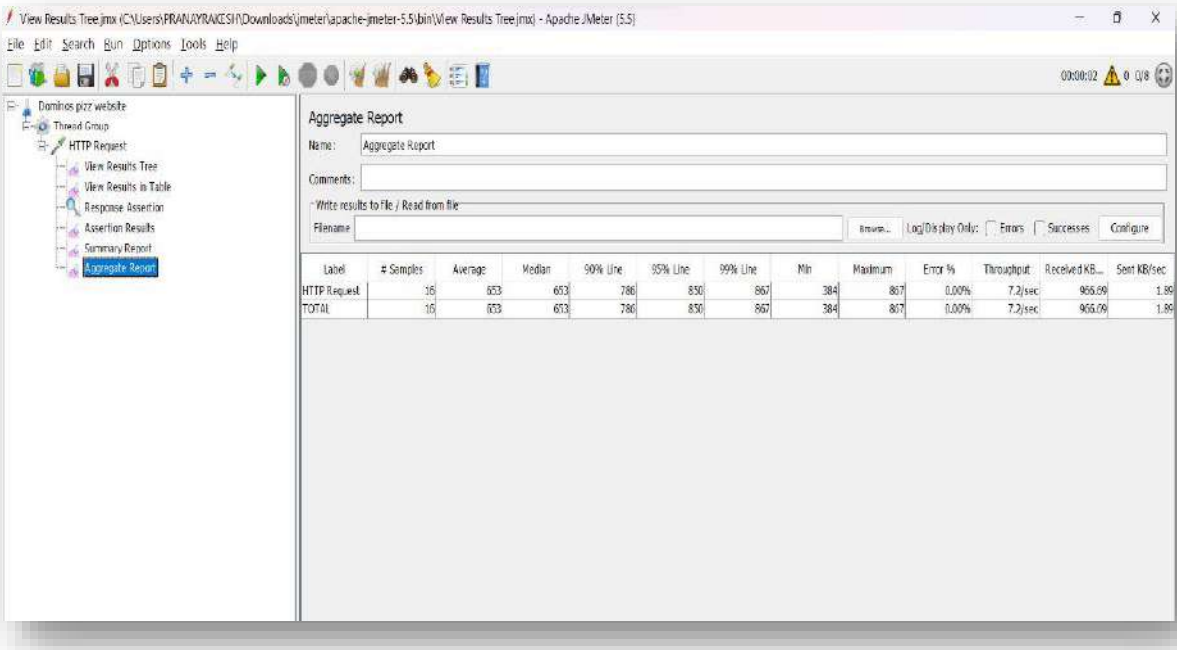
Viewing the results in table:



From the results above, It is confirmed that Domino's pizza website passes all the 16 testcases. Hence, The Dominos website can handle all 16 users' requests concurrently.




File Edit Search Run Options Tools Help
Dominos pizz website
Thread Group
HTTP Request
View Results Tree
View Results In Table
Response Assertion
Assertion Results
Summary Report
Aggregate Report
Summary Report
Name: Summary Report
Comments:
Write results to file / Read from file
Filename
Log/Display Only: Errors Successes Configure
Label | # Samples | Average | Min | Max | Std. Dev. | Error % | Throughput | Received KB/sec | Sent KB/sec | Avg. Bytes
HTTP Request | 16 | 653 | 384 | 867 | 119.53 | 0.00% | 7.2/sec | 966.69 | 1.89 | 136914.4
TOTAL | 16 | 653 | 384 | 867 | 119.53 | 0.00% | 7.2/sec | 966.69 | 1.89 | 136914.4


View Results Tree.jmx (C:\Users\PRANAYRAKESH\Downloads\jmeter\apache-jmeter-5.5\bin\View Results Tree.jmx) - Apache JMeter (5.5)
File Edit Search Run Options Tools Help
Dominos pizz website
Thread Group
HTTP Request
View Results Tree
View Results In Table
Response Assertion
Assertion Results
Summary Report
Aggregate Report
Aggregate Report
Name: Aggregate Report
Comments:
Write results to file / Read from file
Filename
Log/Display Only: Errors Successes Configure
Label | # Samples | Average | Median | 90% Line | 95% Line | 99% Line | Min | Maximum | Error % | Throughput | Received KB... | Sent KB/sec
HTTP Request | 16 | 653 | 653 | 786 | 850 | 867 | 384 | 867 | 0.00% | 7.2/sec | 966.69 | 1.89
TOTAL | 16 | 653 | 653 | 786 | 850 | 867 | 384 | 867 | 0.00% | 7.2/sec | 966.69 | 1.89

Test Report

EXECUTED	PASSED	118	
	FAILED	24	
	(Total) TESTS EXECUTED (PASSED+FAILED)		142
PENDING			10
IN PROGRESS			0
BLOCKED			0
(Sub-Total) TEST PLANNED			**142 + 10**
(PENDING + IN PROGRESS +BLOCKED +TEST EXECUTED)			

FUNCTION	DESCRIPTION	%TCS EXECUTED	%TCS PASSED	TCS PENDING	PRIORITY
VEG	BVC	100%	100%	0	MEDIUM
NON-VEG	BVC	100%	100%	0	MEDIUM
VEG	ROBUSTNESS	100%	67.57%	0	MEDIUM
NON - VEG	ROBUSTNESS	100%	67.57%	0	MEDIUM
BILLING	DECISION TABLE	100%	100%	0	HIGH

TEST SUMMARY REPORT

1. PURPOSE:

This document explains the various activities performed as part of Testing of ‘Self ordering and pizza billing kiosk’ application.

2. APPLICATION OVERVIEW:

A pizza billing self-ordering kiosk is a software application designed to help pizza restaurants streamline their ordering process and reduce wait times for customers. The kiosk is typically placed at the entrance of the restaurant or in a designated area, allowing customers to place their orders quickly and easily.

The main features of a pizza billing self-ordering kiosk include menu management, order customization, payment processing, and order tracking.

With menu management, customers can browse the menu, view item descriptions and prices, and make their selections. The kiosk should display the available options, including any customizations or add-ons, and allow customers to make their choices with ease.

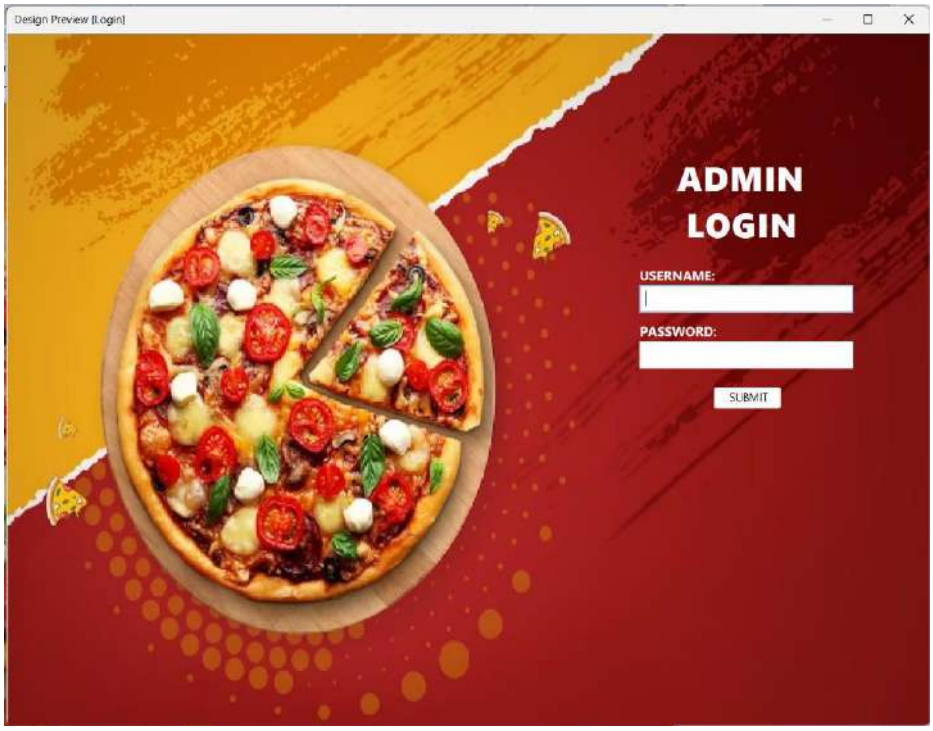
Order customization is another critical feature of a pizza billing self-ordering kiosk. It allows customers to tailor their orders to their specific preferences, such as selecting toppings, crust types, and sauces. This feature helps ensure that customers receive exactly what they want, leading to higher customer satisfaction and repeat business.

3. TESTING SCOPE:

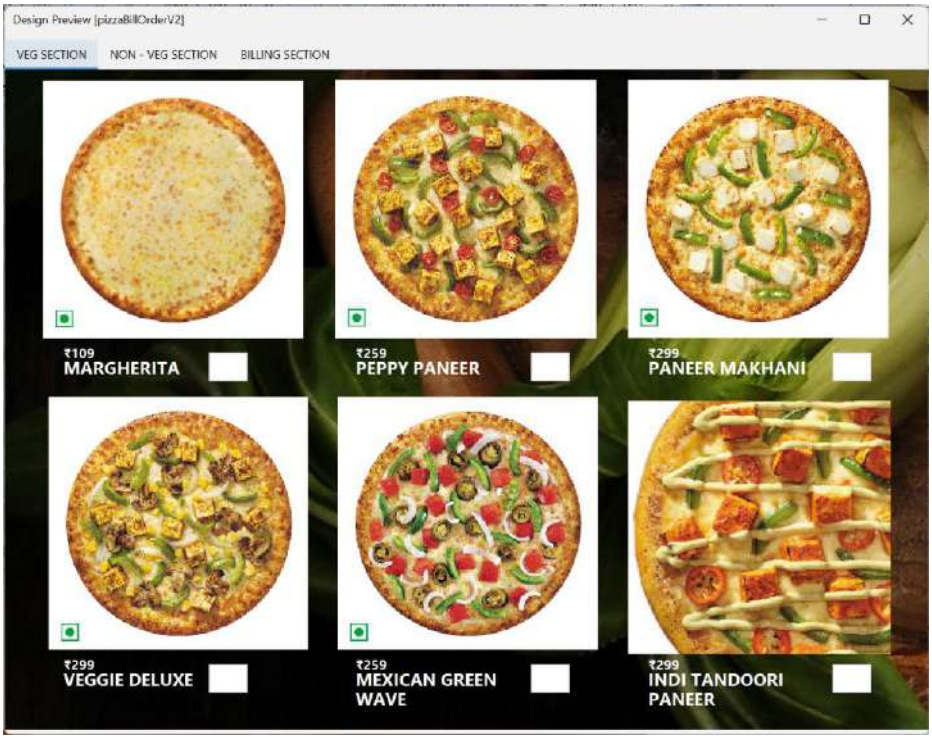
a) In Scope:

Functional testing for the following modules is in scope of Testing

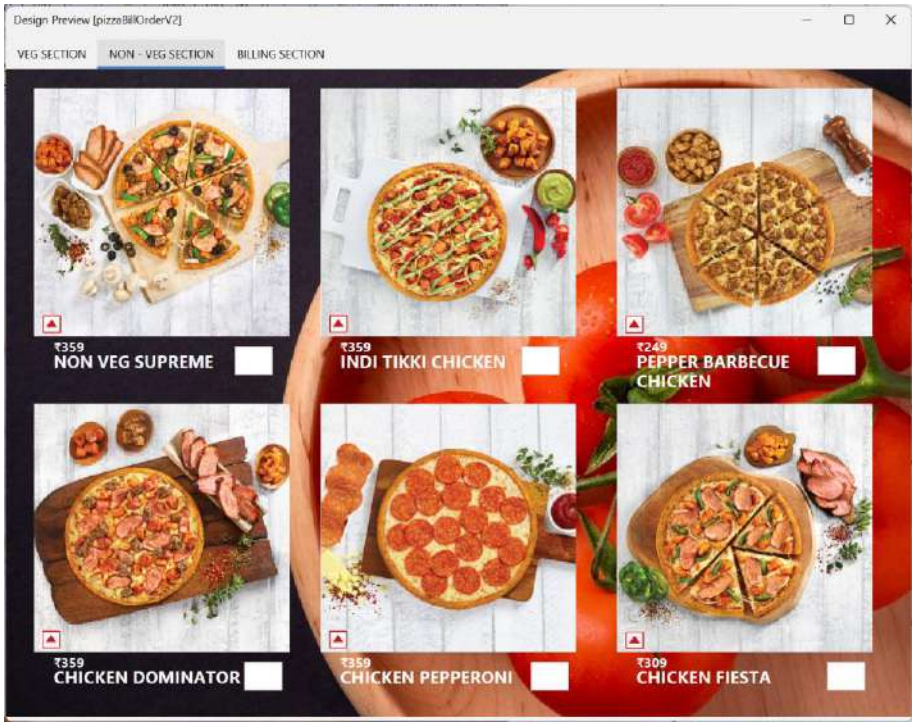
- Admin Login



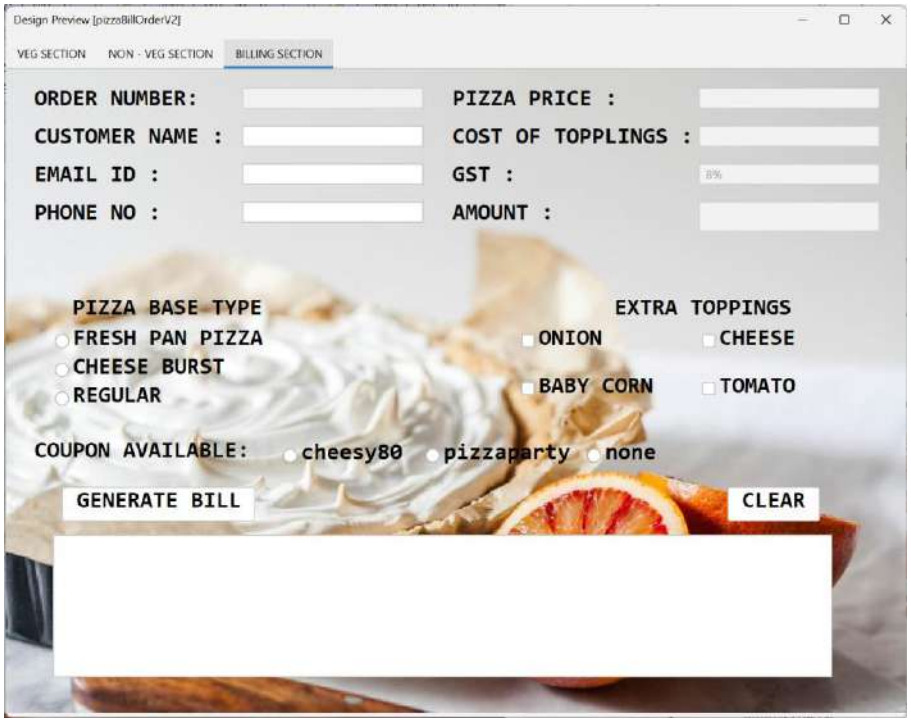
- Veg section



- Non-veg section



- Billing section



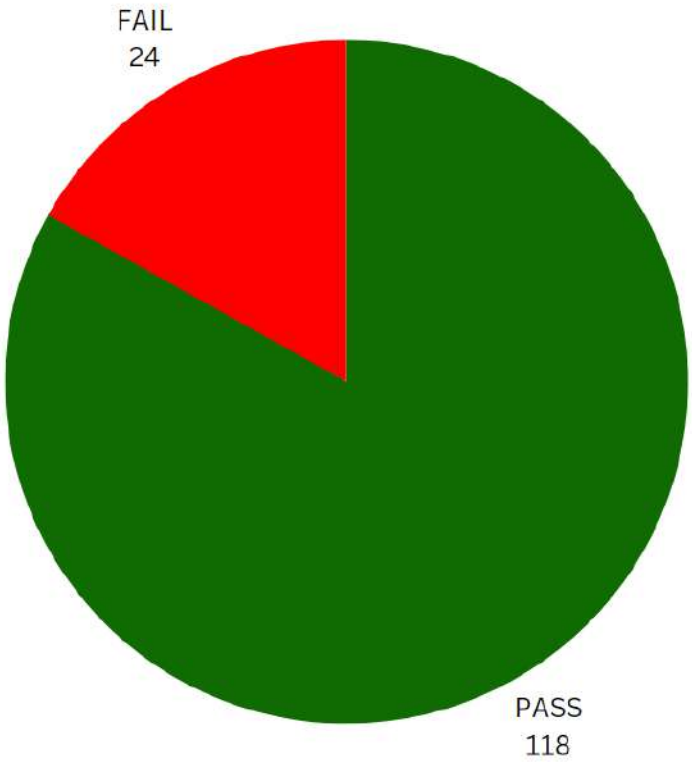
b) Out of Scope:

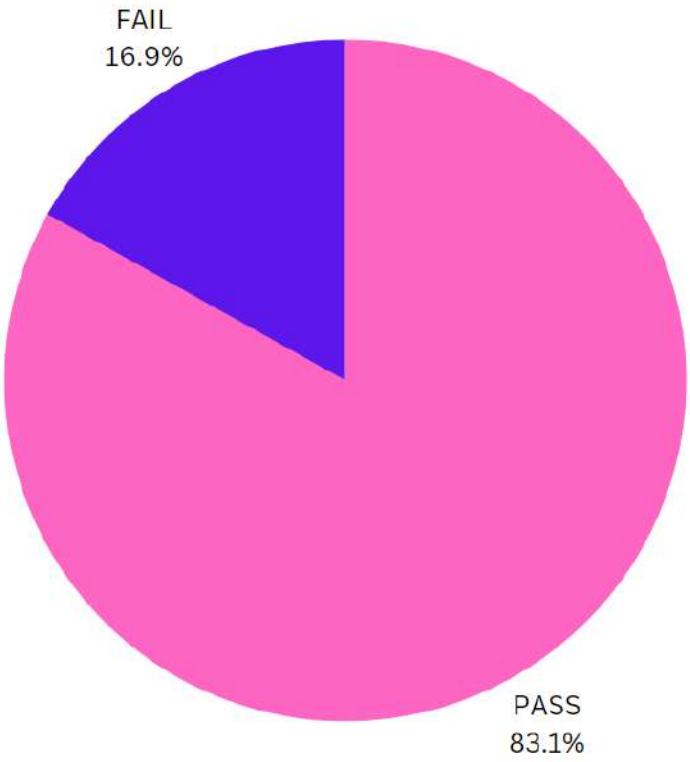
Performance testing.

c) Items not Tested:

1. Login Module
2. Database connectivity between inventory and application.
3. Payment Methods.
4. Connecting customer's email-ID to show all available discounts/coupon to customers.

4. METRICS





5. TYPES OF TESTING

Black box testing techniques:

- Login – Random Testing
- Veg section – Boundary Value Analysis Testing
- Veg section – Robustness Testing
- Non-veg section – Boundary Value Analysis Testing
- Non-veg section – Robustness Testing
- Billing section – Decision Table Testing

White box testing techniques:

Load Testing (non-functional testing) using jmeter

6. TEST ENVIRONMENT AND TOOLS

Junit and TestNG (Apache NetBeans IDE 17)
JMeter

7. LESSONS LEARNT

How to do testing in Junit:

IDE used to test: Apache NetBeans

Tool used to test: Junit (JunitWebsite)

What is Junit?

- The main feature of JUnit-Tools is to **generate JUnit test elements** (packages, classes, methods, test cases, mocks) **depending on an existing java class and logic**.

- It supports the Test-After Development: Create JUnit-tests after writing or modifying application code.

- Other input channels (e.g. UML models) for the generation of test elements to support Test-Driven Development are possible but not available yet.

- JUnit-tests usually have a **similar structure**: Create the preconditions, run the class under test, and validate the postconditions.

- The created test elements for the classes and methods should have the **same name-conventions** to find easily the corresponding tests. These are **good reasons for a generator**.

- The main difference to other tools is, that JUnit-Tools is **completely open source** and that it is **easy to adapt the structure and generated output** to the own needs and requirements.

- There are many extension points and interfaces to change and contribute to the base implementations.

Unit testing:

Unit Testing is a test that tests every single module of the software to check for errors. This is mainly done to discover errors in the code of the Billing System. The main goal of the unit testing would be to isolate each part of the program and to check the correctness of the code. In the case of the Billing System, there are many benefits to this unit testing:

• The unit testing facilitates change in the code.

• It allows testing to be done in a bottom-up fashion.

At the same time, unit testing has some disadvantages such as, it might not identify each error in the system.

How to test the program using Junit (Apache Netbeans Language: Java, Build System: Maven):

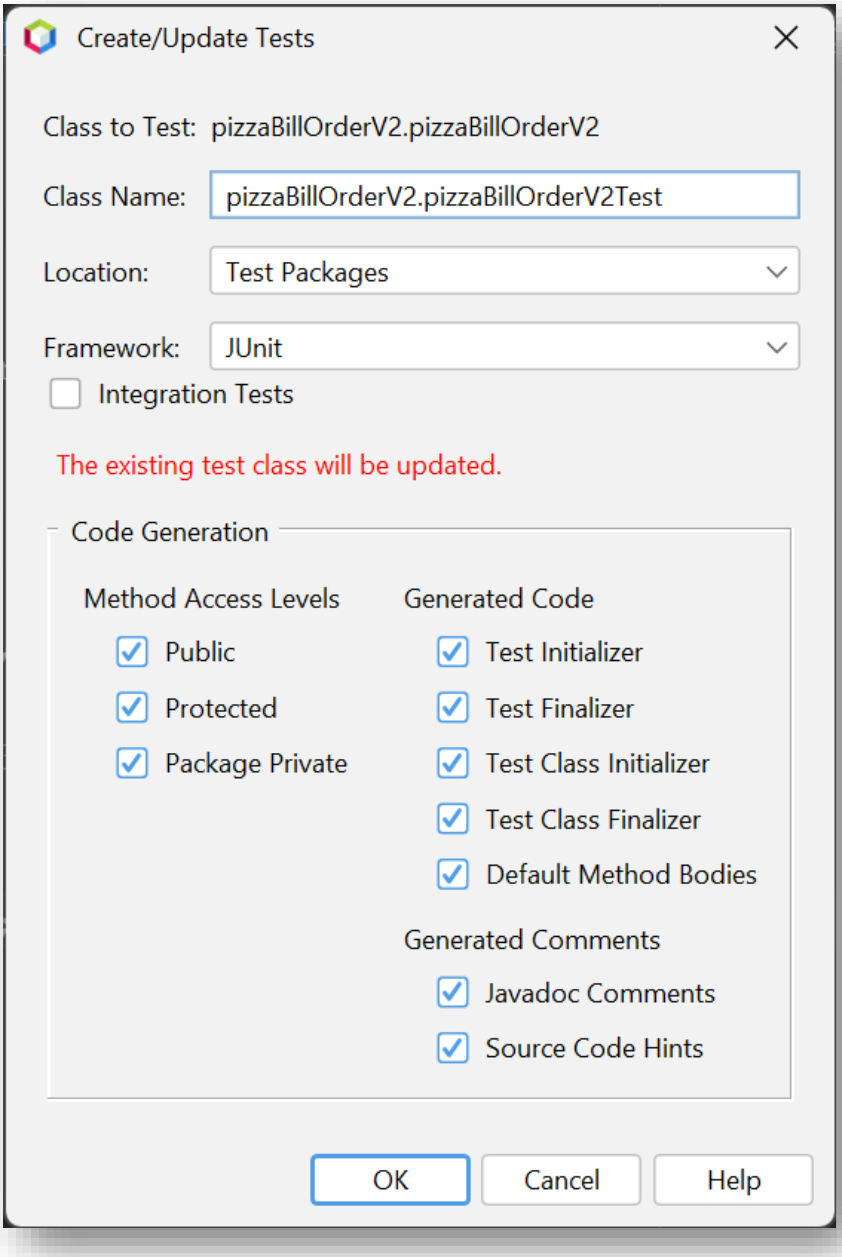
Writing JUnit Unit Tests:

The IDE prompts you to choose a JUnit version the first time that you use the IDE to create tests for you in the project. The version that you select becomes the default JUnit version and the IDE will generate all subsequent tests and test suites for that version.

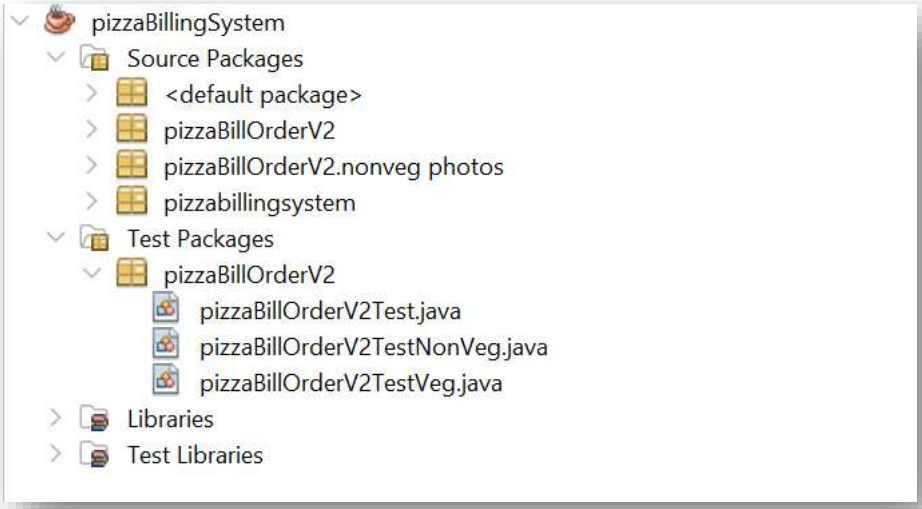
Creating a Test Class for pizzaBillOrderV2.java

1. Right-click pizzaBillOrderV2.java and choose Tools > Create Tests.
2. Modify the name of the test class to pizzaBillOrderV2Test.java in the Create Tests dialog.
3. Select JUnit in the Framework dropdown list.

4. Deselect Test Initializer and Test Finalizer. Click OK.



When you click Select, the IDE creates the pizzaBillOrderV2Test.java test class in the sample package under the Test Packages node in the Projects window.




```java
package pizzaBillOrderV2;

import javax.swing.JTextField;
import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class pizzaBillOrderV2Test {

    public pizzaBillOrderV2Test() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }

    /**
     * Test of eventButtonClicked method, of class pizzaBillOrderV2.
     */
    @Test
    public void testEventButtonClicked() {
        System.out.println(x: "eventButtonClicked");
```

In JUnit 4 you can use annotations to mark the following types of initializer and finalizer methods.

Test Class Initializer. The @BeforeClass annotation marks a method as a test class initialization method. A test class initialization method is run only once, and before any of the other methods in the test class. For example, instead of creating

a database connection in a test initializer and creating a new connection before each test method, you may want to use a test class initializer to open a connection before running the tests. You could then close the connection with the test class finalizer.

Test Class Finalizer. The @AfterClass annotation marks a method as a test class finalizer method. A test class finalizer method is run only once, and after all of the other methods in the test class are finished.

Test Initializer. The @Before annotation marks a method as a test initialization method. A test initialization method is run before each test case in the test class. A test initialization method is not required to run tests, but if you need to initialize some variables before you run a test, you use a test initializer method.

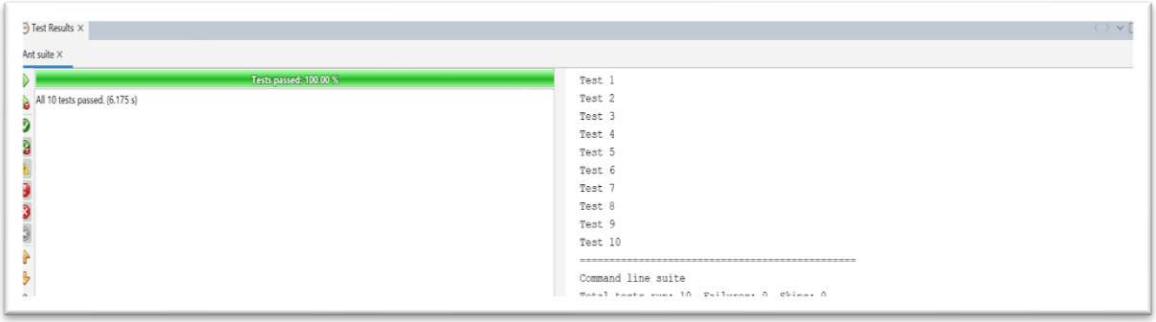
Test Finalizer. The @After annotation marks a method as a test finalizer method. A test finalizer method is run after each test case in the test class. A test finalizer method is not required to run tests, but you may need a finalizer to clean up any data that was required when running the test cases.

Running the Tests:

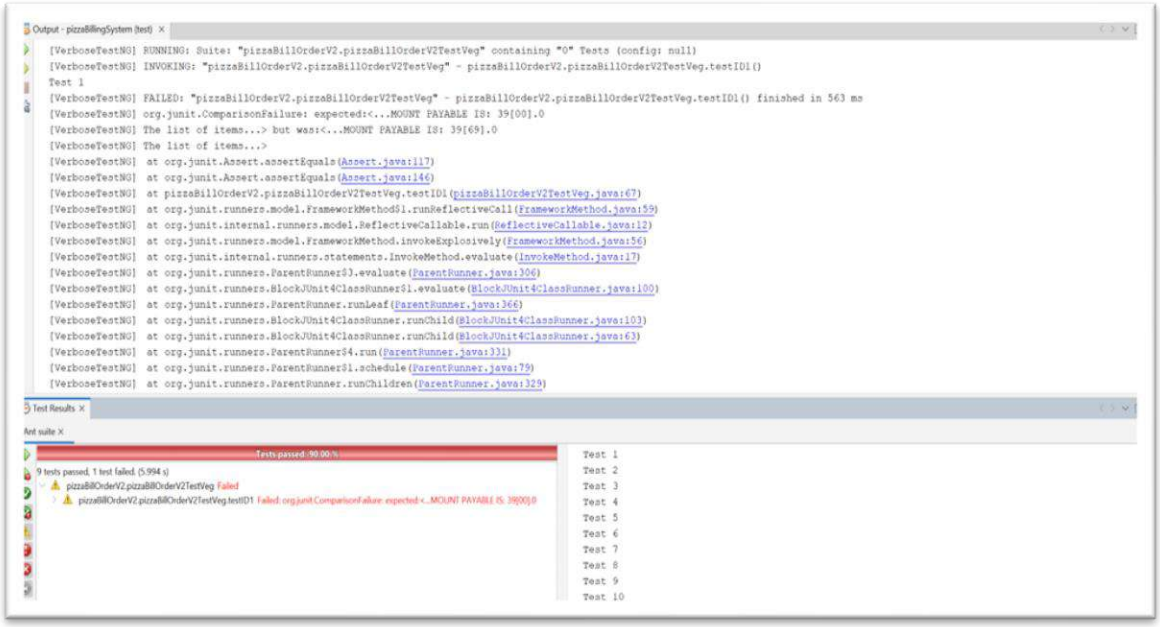
When you run a JUnit test the results are displayed in the Test Results window of the IDE. You can run individual JUnit test classes or you can choose Run > Test PROJECT_NAME from the main menu to run all the tests for the project. If you choose Run > Test, the IDE runs all the test classes in the Test Packages folder. To run an individual test class, right-click the test class under the Test Packages node and choose Run File.

1. Choose Run > Set Main Project in the main menu and select the JUnit-Sample project.
2. Choose Run > Test Project (JUnit-Sample) from the main menu.
3. Choose Window > IDE Tools > Test Results to open the Test Results window.

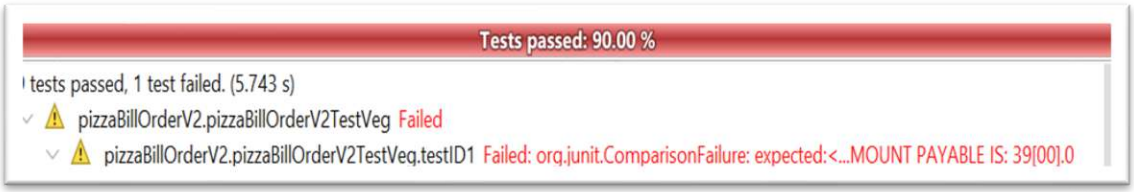
When you run the test, you will see one of the following results in the JUnit Test Results window.



In this image you can see that the project passed all the tests. The left pane displays the results of the individual test methods and the right pane displays the test output. If you look at the output you can see the order that the tests were run. The println that you added to each of the test methods printed out the name of the test to the output window. You can also see that in UtilJUnit3Test the setUpClass() method was run before each test method and the tearDownClass() method was run after each method.



In this image you can see that the project failed one of the tests. As the expected result does not match with actual result.



The next step after you creates your unit test classes is to create test suites. See Creating JUnit Test Suites to see how to run specified tests as a group so you do not have to run each test individually.

Conclusion:

This tutorial has given you a basic introduction to creating JUnit unit tests and test suites in NetBeans IDE. The IDE supports JUnit 3 and JUnit 4, and this document demonstrated some of the changes introduced in JUnit 4 that are designed to make creating and running tests simpler.

As demonstrated in this tutorial, one of the main improvements in JUnit 4 is support for annotations. In JUnit 4 you can now use annotations to do the following:

- Identify a test using the @Test annotation instead of the naming convention
- Identify setUpClass() and tearDownClass() methods with @Before and @After annotations
- Identify setUpClass() and tearDownClass() methods that apply to the entire test class. Methods annotated with @BeforeClass are run only once before any test methods in the class are run. Methods annotated with @AfterClass are also run only once after all the test methods have finished.
- Identify expected exceptions
- Identify tests that should be skipped using the @Ignore annotation

Specify a timeout parameter for a test

Testing code often helps ensure that small changes made in the code do not break the application. Automated testing tools like JUnit streamline the process of testing and frequent testing can help catch coding errors early.

8. RECOMMENDATIONS

a. While testing individual modules set other variables to be default or valid values as other variables are independent of the module we are testing.

9. EXIT CRITERIA

a. All test cases should be executed – Yes
b. All defects in Critical, Major, Medium severity should be verified and closed – No.
c. Any open defects in trivial severity – Yes

10. CONCLUSION/SIGN OFF

As the Exit criteria was **NOT** met and satisfied as mentioned in Section 10, this application is **NOT** suggested to 'Go Live' by the Testing team.

11. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

BVA – Boundary Value Analysis

CONCLUSION:

Based on the testing done for the pizza billing system, it can be concluded that the system did not function as expected. The tests were carried out to check the system's accuracy, reliability, and user-friendliness. We used junit, jmeter and testNG testing tools. By comparing the generated bills with the actual orders submitted, the system's accuracy was confirmed, and some disparities were discovered in veg and non-veg modules using robustness testing. The system was proven to be dependable because it could manage a high number of orders without slowing down or crashing. Also, the system's usability was examined, and it was discovered to be simple to operate. Customers and staff could easily use the system because of its user-friendly UI. Overall, the evaluation of the pizza billing system shows the development team has to set a boundary over pizza quantity and fix some logical code in the billing system, other than that the system is user-friendly and dependable.