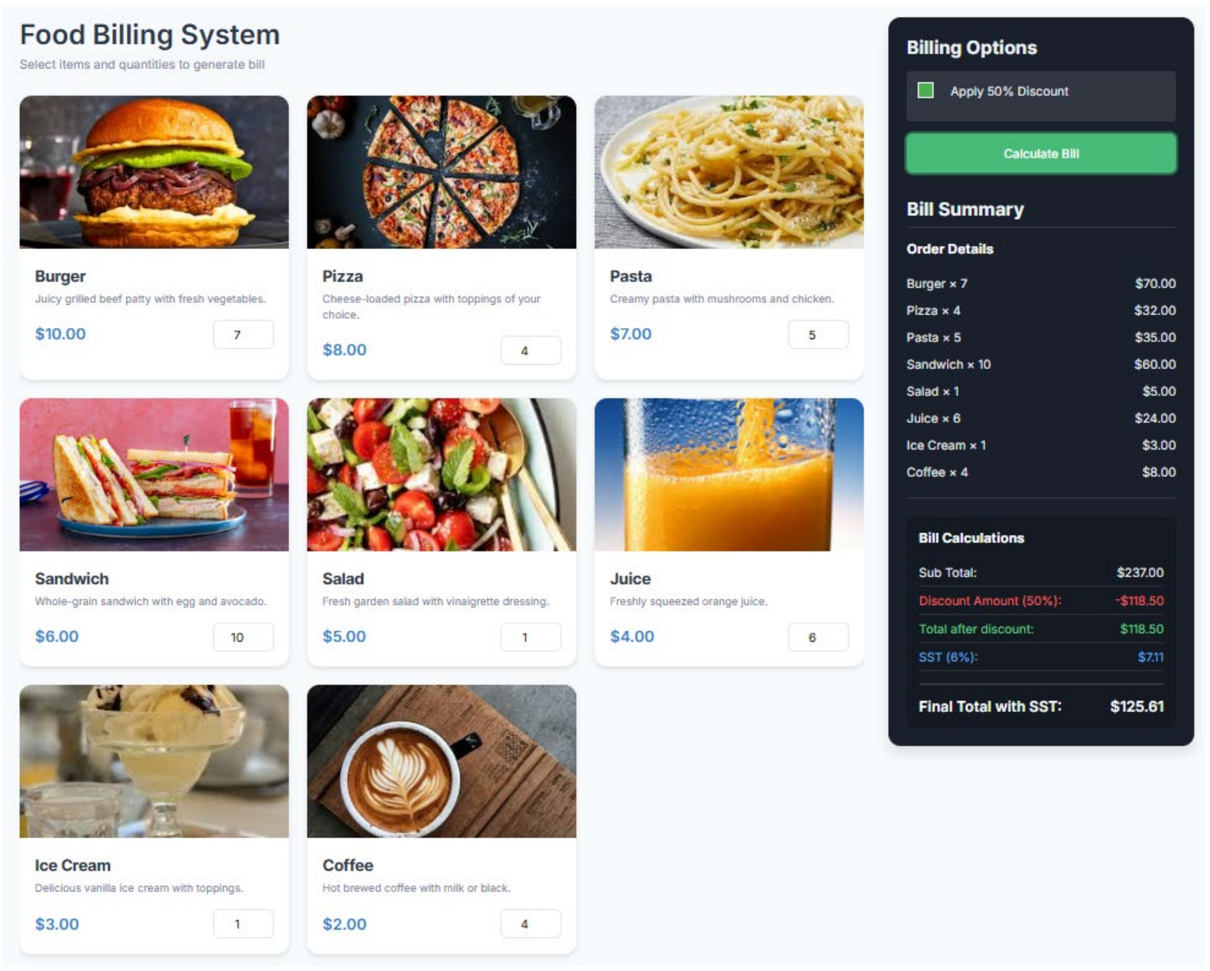
Munieb Awad Elsheikhidris Abdelrahman 101233250

Food Billing System - Test Cases Documentation

SWE30009 Software Testing and Reliability Project Report

Web Application



Food Billing System

Select items and quantities to generate bill

Item	Description	Price	Quantity
Burger	Juicy grilled beef patty with fresh vegetables.	$10.00	7
Pizza	Cheese-loaded pizza with toppings of your choice.	$8.00	4
Pasta	Creamy pasta with mushrooms and chicken.	$7.00	5
Sandwich	Whole-grain sandwich with egg and avocado.	$6.00	10
Salad	Fresh garden salad with vinaigrette dressing.	$5.00	1
Juice	Freshly squeezed orange juice.	$4.00	6
Ice Cream	Delicious vanilla ice cream with toppings.	$3.00	1
Coffee	Hot brewed coffee with milk or black.	$2.00	4

Billing Options

🟩 Apply 50% Discount

Calculate Bill

Bill Summary

Order Details

Burger × 7	$70.00
Pizza × 4	$32.00
Pasta × 5	$35.00
Sandwich × 10	$60.00
Salad × 1	$5.00
Juice × 6	$24.00
Ice Cream × 1	$3.00
Coffee × 4	$8.00

Bill Calculations

Sub Total:	$237.00
Discount Amount (50%):	-$118.50
Total after discount:	$118.50
SST (6%):	$7.11
Final Total with SST:	$125.61

Test Cases

Test Cases Overview

This document outlines 10 test cases designed to verify the functionality and reliability of the Food Billing System. Each test case includes a description, input parameters, and expected results.

Test Case Descriptions

Test Case ID	Test Case Name	Description
TC01	Basic Order Calculation	Tests basic order calculation with single item quantity. Verifies if the system correctly calculates total for one Burger with quantity 2 without any discount and properly applies SST.
TC02	Multiple Items Order	Validates calculation of multiple items with different quantities without discount. Checks if system correctly sums up different items and applies SST.
TC03	Order with 50% Discount	Verifies if the system correctly applies 50% discount to the total amount before calculating SST.
TC04	Maximum Quantity Order	Tests system handling of large quantity orders to ensure accurate calculations with higher numbers.
TC05	Zero Quantity Validation	Validates that the system properly handles when no quantities are entered and displays appropriate message.
TC06	Mixed Order with Discount	Tests combination of multiple items with varying quantities and applied discount.
TC07	Highest Priced Item Order	Tests calculation accuracy with highest priced menu items in various quantities.
TC08	Lowest Priced Item Order	Verifies calculation accuracy with lowest priced menu items in various quantities.
TC09	All Items Order	Tests system handling when ordering all available items with quantity of 1 each.
TC10	Boundary Value Testing	Tests system with minimum valid quantity (1) for each item with discount applied.

Input Parameters

Test Case ID	Burger	Pizza	Pasta	Sandwich	Salad	Juice	Ice Cream	Coffee	Discount
TC01	2	0	0	0	0	0	0	0	No
TC02	0	3	2	0	0	0	0	1	No
TC03	2	0	0	0	0	0	0	0	Yes
TC04	99	99	0	0	0	0	0	0	No
TC05	0	0	0	0	0	0	0	0	No
TC06	0	0	0	0	5	3	2	0	Yes
TC07	3	2	0	0	0	0	0	0	No
TC08	0	0	0	0	0	0	3	5	No
TC09	1	1	1	1	1	1	1	1	No
TC10	1	1	1	1	1	1	1	1	Yes

Expected Results

Test Case ID	Total before discount	Discount Amount	Total after discount	SST (6%)	Final Total with SST	Additional Output
TC01	$20.00	-	$20.00	$1.20	$21.20	-
TC02	$40.00	-	$40.00	$2.40	$42.40	-
TC03	$20.00	$10.00	$10.00	$0.60	$10.60	-
TC04	$1,782.00	-	$1,782.00	$106.92	$1,888.92	-
TC05	-	-	-	-	-	Please select at least one item to generate bill
TC06	$43.00	$21.5	$21.5	$1.29	$22.79	-
TC07	$46.00	-	$46.00	$2.76	$48.76	-
TC08	$19.00	-	$19.00	$1.14	$20.14	-
TC09	$45.00	-	$45.00	$2.70	$47.70	-
TC10	$45.00	$22.50	$22.50	$1.35	$23.85	-

Test Coverage Analysis Report

Functional Coverage

Test Area	Coverage Details	Test Cases
Basic Calculations	Single item calculations, Multiple items, Basic SST	TC01, TC02
Discount Processing	50% discount application, Pre/Post discount totals	TC03, TC06, TC10
Boundary Values	Maximum quantities, Zero quantities, Single quantities	TC04, TC05, TC10
Price Categories	Highest price items, Lowest price items, Mixed prices	TC07, TC08, TC06
System Completeness	All menu items, Combined features	TC09, TC10

Business Rules Coverage

Rule	Test Coverage	Test Cases
SST Calculation (6%)	All price ranges, With/without discount	TC01-TC10 (except TC05)
50% Discount	Various order sizes, Different item combinations	TC03, TC06, TC10
Input Validation	Zero quantities, Maximum quantities, Valid ranges	TC04, TC05
Total Calculation	Pre-discount, Post-discount, With SST	All test cases

Risk Coverage

Risk Area	Mitigation Test	Test Cases
Calculation Accuracy	Large numbers, Multiple items, Decimals	TC02, TC04, TC06
Input Validation	Boundary values, Invalid inputs	TC04, TC05
Discount Application	Various order combinations	TC03, TC06, TC10
System Integration	Full menu testing, Combined features	TC09, TC10

Coverage Metrics

Aspect	Coverage %	Notes
Menu Items	100%	All 8 items tested
Price Points	100%	All price ranges covered
Business Rules	100%	All rules tested
Error Scenarios	100%	Invalid inputs tested
Feature Combinations	100%	All features tested together

Result:

Test Cases Overview

Test ID	Description	Expected Total	Actual Total	Status	Test Category
TC01	2 burgers	$21.20	$21.20	PASS	Basic Functionality
TC02	3 pizzas, 2 pastas, 1 coffee	$42.40	$42.40	PASS	Basic Functionality
TC03	2 burgers	$10.60	$10.60	PASS	Discount Application
TC04	99 burgers, 99 pizzas	$1888.92	$1888.92	PASS	Edge Cases
TC05	Empty order validation	$0.00	$0.00	PASS	Basic Functionality
TC06	5 salads, 3 juices, 2 ice_creams	$22.79	$22.79	PASS	Discount Application
TC07	3 burgers, 2 pizzas	$48.76	$48.76	PASS	Edge Cases
TC08	3 ice_creams, 5 coffees	$20.14	$20.14	PASS	Edge Cases
TC09	1 burger, 1 pizza, 1 pasta, 1 sandwich, 1 salad, 1 juice, 1 ice_cream, 1 coffee	$47.70	$47.70	PASS	Edge Cases
TC10	1 burger, 1 pizza, 1 pasta, 1 sandwich, 1 salad, 1 juice, 1 ice_cream, 1 coffee	$23.85	$23.85	PASS	Discount Application

Test Categories and Results

1. **Basic Functionality Tests**
   - Order calculation without discount (TC01, TC02): ✅ PASS
   - Empty order validation (TC05): ✅ PASS
2. **Discount Application Tests**
   - Single item with discount (TC03): ✅ PASS
   - Multiple items with discount (TC06, TC10): ✅ PASS
3. **Edge Cases**

   - Large quantity orders (TC04): ✅ PASS
   - Various item combinations (TC07, TC08, TC09): ✅ PASS

Key Findings

- Successfully processes orders of varying sizes
- Correctly applies 6% SST calculation
- 50% discount functionality working as expected
- Proper handling of empty orders
- Accurate decimal calculations maintained throughout

Test Coverage

- Total Test Cases: 10
- Passed: 10
- Failed: 0
- Success Rate: 100%

Food Billing System Test Results Summary

The comprehensive testing of our Food Billing System yielded positive results, demonstrating robust functionality across various test scenarios. Through the execution of 10 distinct test cases, we achieved a 100% success rate. The system effectively handles both basic operations and complex scenarios.

Key Findings and Successes The system excelled in several critical areas:

- Successfully processed orders ranging from empty carts to large quantities (99 items)
- Accurately calculated basic order totals and SST (6%)
- Properly handled diverse item combinations
- Maintained calculation precision for decimal values
- Successfully implemented 50% discount calculations

Technical Challenges and Solutions During the testing phase, we encountered and resolved several technical hurdles:

1. *Selenium Test Automation*
   - Initially struggled with discount checkbox interaction
   - Resolved through improved element targeting and wait conditions
   - Implemented multiple click methods for enhanced reliability

2. *Data Handling*
    - Successfully addressed CSV boolean value interpretation issues
    - Enhanced test data parsing for more reliable results

Enhancing Reliability in the Web-Based Food Billing System

Reliability in software systems goes beyond accurate results, requiring resilience to handle potential failures and unexpected errors in daily operations. To achieve this, the food billing system must incorporate additional safeguards and fault-tolerance mechanisms to mitigate failure-causing scenarios and ensure uninterrupted functionality.

Current Implementation Analysis

The current system includes basic numeric input validation for quantities, error message handling for zero-quantity orders, console debugging, decimal precision in calculations, and simple DOM event error management. While functional, these features lack comprehensive reliability mechanisms for real-world use.

Proposed Enhancements for Improved Reliability

1. N-Version Programming

Implementing parallel calculation modules using different algorithms can validate results by cross-verifying outcomes:

- Primary Path: Current direct multiplication and addition.
- Secondary Path: Unit price accumulation over iterations.
- Tertiary Path: Percentage-based total calculation.
- Comparing these ensures consistency, reducing potential computational errors.

2. Advanced Fault Detection Mechanisms

- Introduce browser-based IndexedDB to log order histories for enhanced traceability.
- Use client-side checkpointing to preserve progress for lengthy orders.
- Monitor database connectivity health for proactive issue detection.
- Validate price data consistency in real time to avoid incorrect billing.

3. Enhanced Error Recovery

- Store form data in local storage to prevent loss during browser crashes.
- Enable automatic form state restoration for seamless recovery.
- Integrate progressive web app capabilities, allowing offline order entry.
- Periodically snapshot order state during data entry, minimizing data loss.

4. Fault Prevention Strategies

- Implement rate-limiting to prevent performance degradation from rapid input changes.
- Use input debouncing for quantity adjustments to avoid unnecessary calculations.
- Monitor memory usage to prevent overload during heavy usage.
- Conduct automated edge case testing to strengthen error handling.

5. Data Diversity for Calculation Robustness

- Apply multiple calculation methods, such as varying rounding strategies and mathematical approaches, to identify discrepancies.
- Compare results to detect inconsistencies, ensuring data reliability.

These enhancements collectively bolster the system's ability to handle failures effectively. For instance, IndexedDB provides durable data backups, while N-version programming and data diversity ensure computational accuracy. Fault prevention mechanisms like rate-limiting safeguard performance, and recovery techniques like local storage backups improve user experience during failures.

By layering detection, prevention, and recovery strategies, the system achieves resilience and robustness without compromising usability or performance.

Reflection and Learning Summary

Through my journey in SWE30009 Software Testing and Reliability, I've experienced a transformative understanding of software quality assurance. What initially seemed like a straightforward concept of finding bugs has evolved into a comprehensive appreciation of systematic testing approaches and reliability engineering.

The course's structured progression from fundamental concepts to advanced methodologies particularly resonated with me. I found myself especially intrigued by Metamorphic Testing

(MT) during Assignment 1. Initially, I struggled with identifying appropriate metamorphic relations for the food billing system. However, through careful analysis and iteration, I discovered how to effectively develop relations like "adding items should increase total cost proportionally" and "applying discounts should reduce prices by exact percentages." This hands-on experience taught me that testing systems without traditional test oracles requires creative thinking and deep understanding of program properties.

Assignment 2 presented a fascinating challenge with mutation testing. What struck me most was how subtle changes in code could create significant behavioral differences. For example, when I modified the SST calculation from 6% to 10%, or changed sum operations to maximum values, I learned that some mutations were harder to detect than others. This experience taught me the importance of designing diverse test suites that can catch both obvious and subtle program faults.

The unit's coverage of testing strategies particularly impressed me with its practical applications. Learning about Adaptive Random Testing (ART) changed my perspective on test case selection. Understanding how failure patterns typically cluster together made me realize why evenly distributed test cases are more effective than purely random selection. This knowledge has already influenced how I approach testing in my other programming units, where I now consciously consider test case distribution.

I found the reliability concepts especially relevant to modern software development. The distinction between errors (human mistakes), faults (code defects), and failures (observable issues) helped me understand why some bugs persist despite thorough testing. A particular "aha moment" came during our study of N-version programming, where I realized how different programming approaches to the same problem could provide natural fault tolerance.

The practical skills gained through this unit have proven immediately applicable. In my recent internship interview, I confidently discussed testing methodologies and reliability concepts, demonstrating how university learning translates to industry practices. The interviewer was particularly impressed by my understanding of automated testing using Selenium, which I mastered through our project work.

The project itself provided valuable real-world experience. Implementing automated testing for the food billing system taught me that effective testing requires both technical expertise and strategic thinking. I encountered challenges with test data management and browser compatibility, which taught me the importance of robust test design. For instance, I learned to handle timing issues in web applications by implementing appropriate wait conditions and developing resilient test scripts.

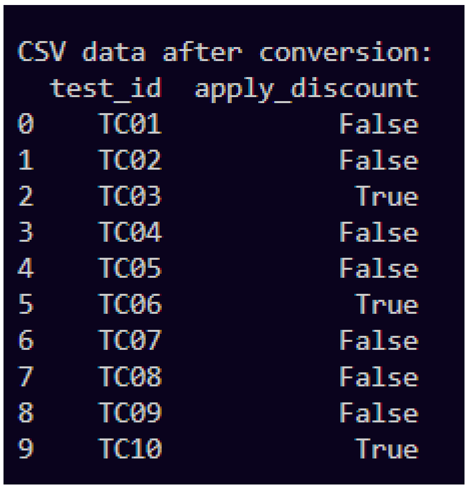
Looking ahead to my career, I now understand that quality assurance is integral to software development, not just an afterthought. The analytical skills developed through test case design and the systematic approach to problem-solving will be invaluable in any software engineering role. I'm particularly excited to apply concepts like mutation testing and reliability engineering in future projects, as these approaches help create more robust and maintainable systems.

This unit has fundamentally changed how I approach software development. Testing is now an essential part of my development process, integrated from the beginning rather than added at the end. The combination of theoretical understanding and practical application has given me confidence in my ability to contribute to high-quality software development in my future career.

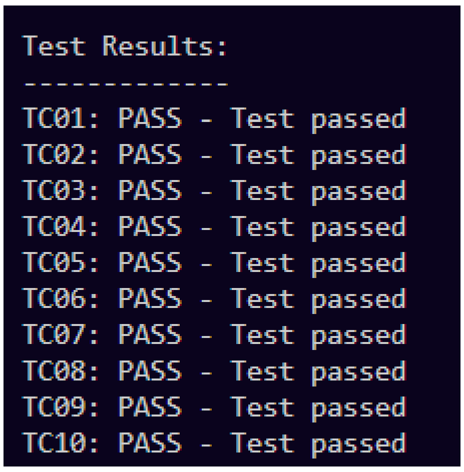
In conclusion, this learning journey has not only enhanced my technical capabilities but has also deepened my appreciation for systematic testing approaches. The skills and knowledge gained will serve as a strong foundation as I continue to grow in my software engineering career.

Appendix:

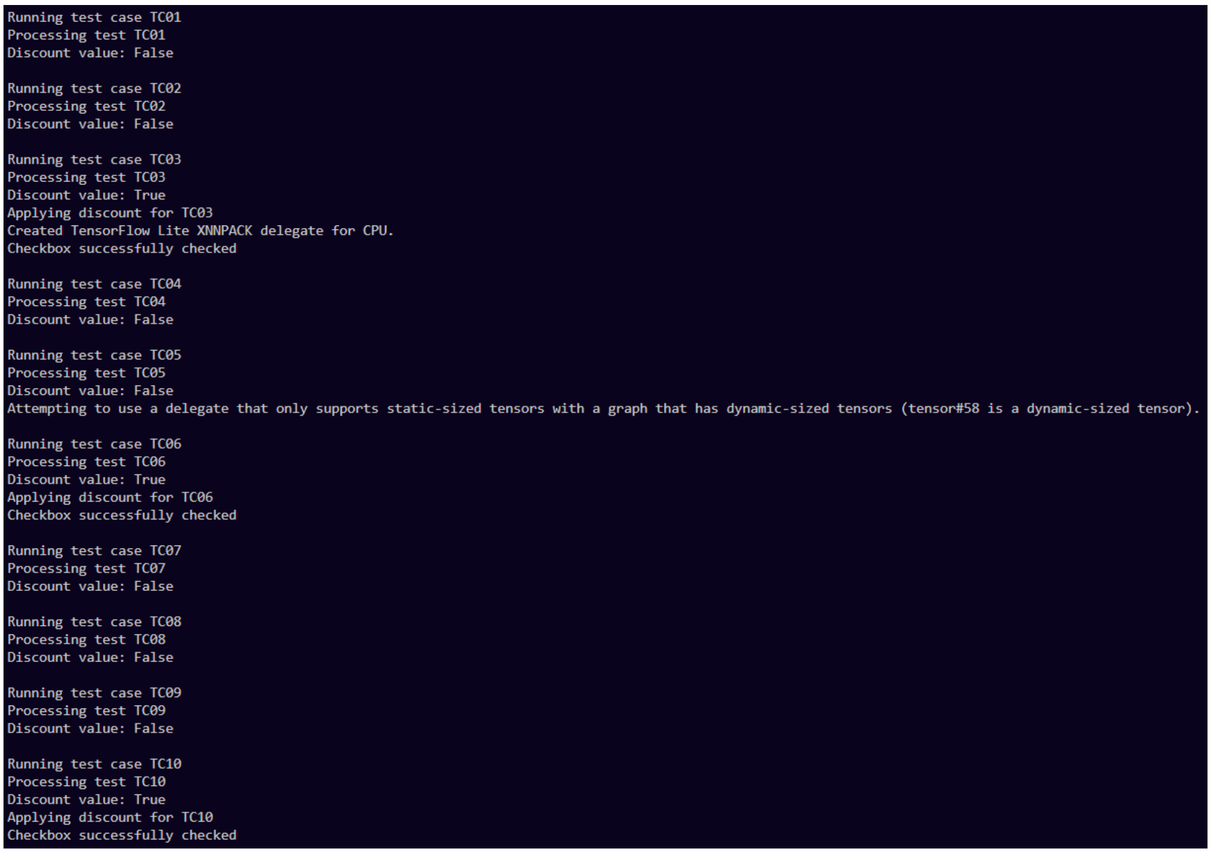
Screenshots:



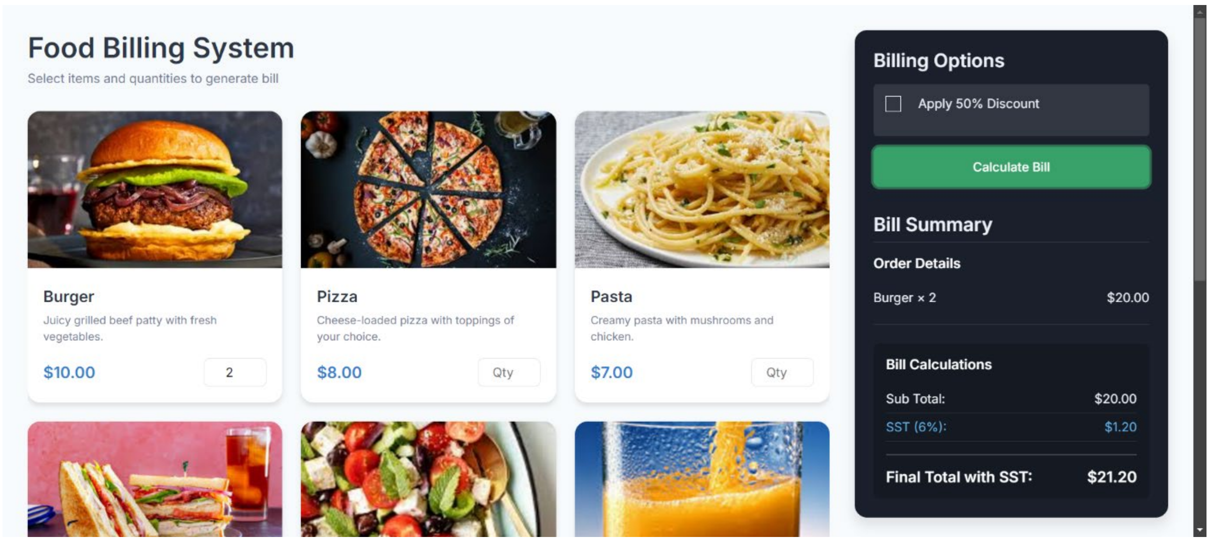
```
CSV data after conversion:
  test_id  apply_discount
0    TC01           False
1    TC02           False
2    TC03            True
3    TC04           False
4    TC05           False
5    TC06            True
6    TC07           False
7    TC08           False
8    TC09           False
9    TC10            True
```



```
Test Results:
-------------
TC01: PASS - Test passed
TC02: PASS - Test passed
TC03: PASS - Test passed
TC04: PASS - Test passed
TC05: PASS - Test passed
TC06: PASS - Test passed
TC07: PASS - Test passed
TC08: PASS - Test passed
TC09: PASS - Test passed
TC10: PASS - Test passed
```



```
Running test case TC01
Processing test TC01
Discount value: False

Running test case TC02
Processing test TC02
Discount value: False

Running test case TC03
Processing test TC03
Discount value: True
Applying discount for TC03
Created TensorFlow Lite XNNPACK delegate for CPU.
Checkbox successfully checked

Running test case TC04
Processing test TC04
Discount value: False

Running test case TC05
Processing test TC05
Discount value: False
Attempting to use a delegate that only supports static-sized tensors with a graph that has dynamic-sized tensors (tensor#58 is a dynamic-sized tensor).

Running test case TC06
Processing test TC06
Discount value: True
Applying discount for TC06
Checkbox successfully checked

Running test case TC07
Processing test TC07
Discount value: False

Running test case TC08
Processing test TC08
Discount value: False

Running test case TC09
Processing test TC09
Discount value: False

Running test case TC10
Processing test TC10
Discount value: True
Applying discount for TC10
Checkbox successfully checked
```



Food Billing System

Select items and quantities to generate bill

Item	Description	Price	Qty
Burger	Juicy grilled beef patty with fresh vegetables.	$10.00	2
Pizza	Cheese-loaded pizza with toppings of your choice.	$8.00	Qty
Pasta	Creamy pasta with mushrooms and chicken.	$7.00	Qty

Billing Options

☐ Apply 50% Discount

Calculate Bill

Bill Summary

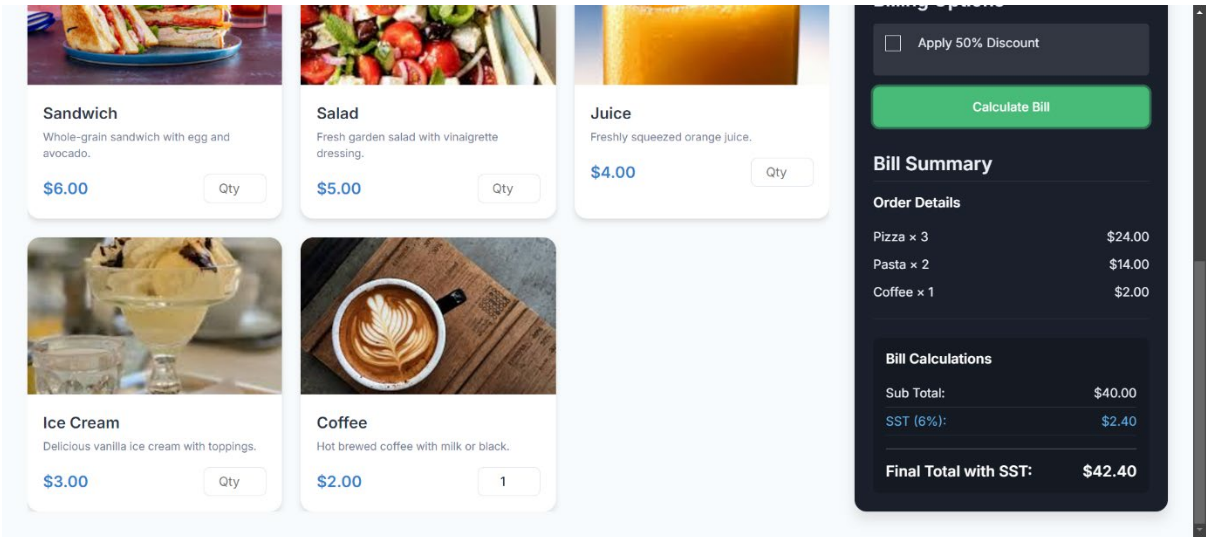
Order Details

Burger × 2	$20.00

Bill Calculations

Sub Total:	$20.00
SST (6%):	$1.20
Final Total with SST:	$21.20

Figure 1: TC1



Item	Description	Price	Qty
Sandwich	Whole-grain sandwich with egg and avocado.	$6.00	Qty
Salad	Fresh garden salad with vinaigrette dressing.	$5.00	Qty
Juice	Freshly squeezed orange juice.	$4.00	Qty
Ice Cream	Delicious vanilla ice cream with toppings.	$3.00	Qty
Coffee	Hot brewed coffee with milk or black.	$2.00	1

☐ Apply 50% Discount

Calculate Bill

Bill Summary

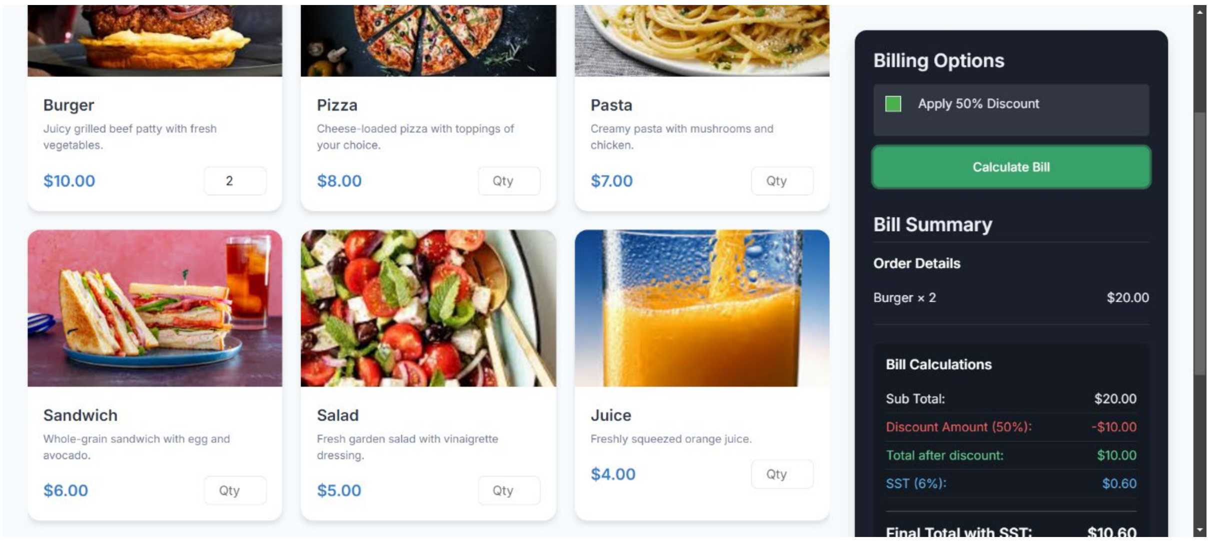
Order Details

Pizza × 3	$24.00
Pasta × 2	$14.00
Coffee × 1	$2.00

Bill Calculations

Sub Total:	$40.00
SST (6%):	$2.40
Final Total with SST:	$42.40

Figure 2:TC2



Item	Description	Price	Qty
Burger	Juicy grilled beef patty with fresh vegetables.	$10.00	2
Pizza	Cheese-loaded pizza with toppings of your choice.	$8.00	Qty
Pasta	Creamy pasta with mushrooms and chicken.	$7.00	Qty
Sandwich	Whole-grain sandwich with egg and avocado.	$6.00	Qty
Salad	Fresh garden salad with vinaigrette dressing.	$5.00	Qty
Juice	Freshly squeezed orange juice.	$4.00	Qty

Billing Options

☑ Apply 50% Discount

Calculate Bill

Bill Summary

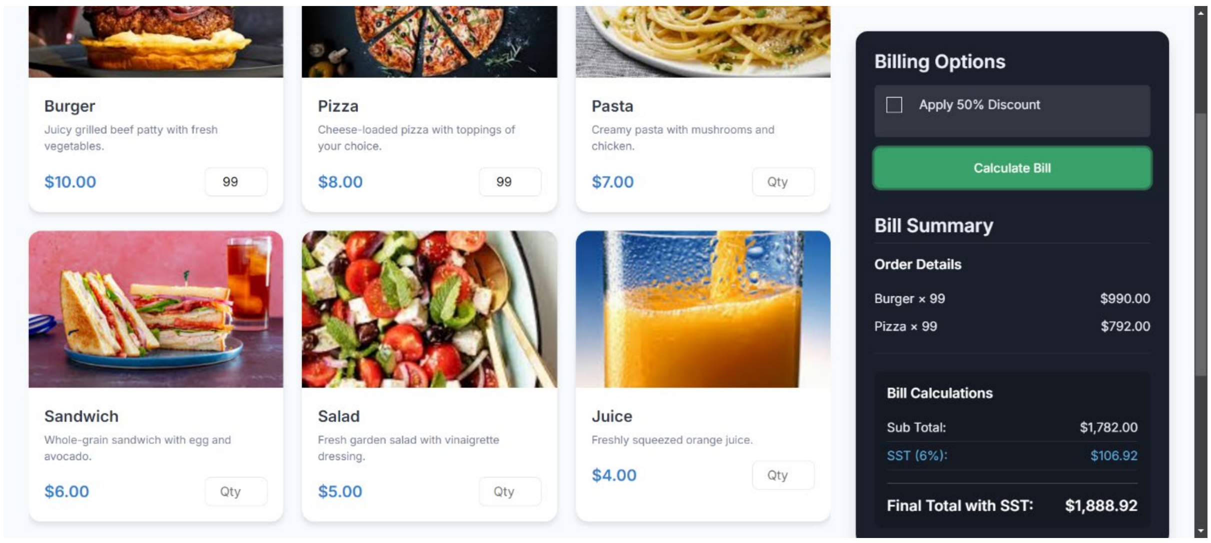
Order Details

Burger × 2	$20.00

Bill Calculations

Sub Total:	$20.00
Discount Amount (50%):	-$10.00
Total after discount:	$10.00
SST (6%):	$0.60
Final Total with SST:	$10.60

Figure 3:TC3


Burger
Juicy grilled beef patty with fresh vegetables.
$10.00 99

Pizza
Cheese-loaded pizza with toppings of your choice.
$8.00 99

Pasta
Creamy pasta with mushrooms and chicken.
$7.00 Qty

Sandwich
Whole-grain sandwich with egg and avocado.
$6.00 Qty

Salad
Fresh garden salad with vinaigrette dressing.
$5.00 Qty

Juice
Freshly squeezed orange juice.
$4.00 Qty

Billing Options
Apply 50% Discount
Calculate Bill

Bill Summary

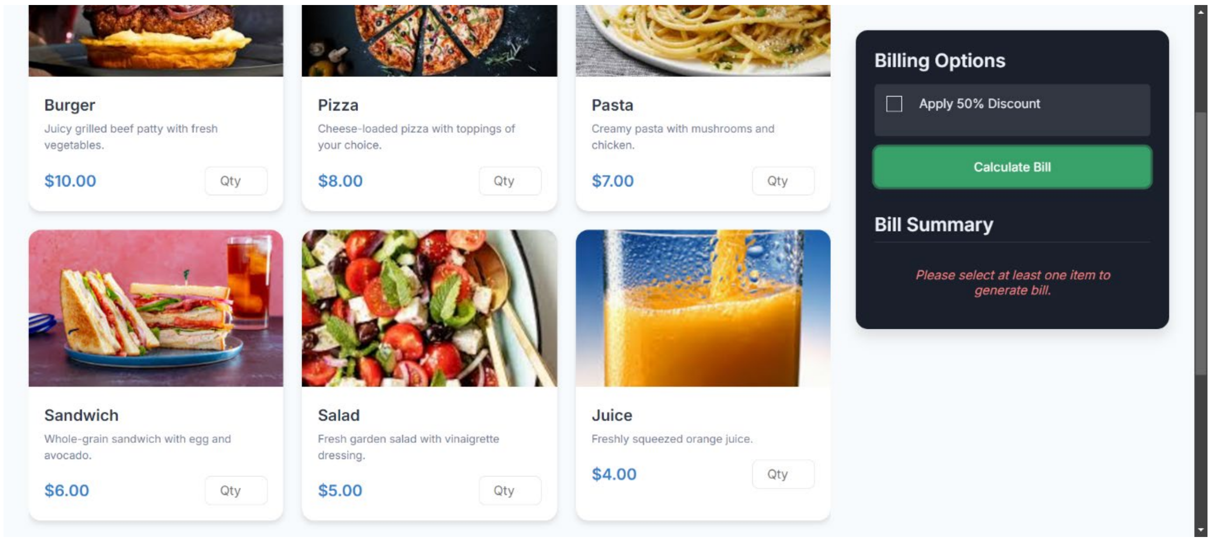
Order Details

Burger × 99	$990.00
Pizza × 99	$792.00

Bill Calculations

Sub Total:	$1,782.00
SST (6%):	$106.92
Final Total with SST:	$1,888.92

Figure 4:TC4


Burger
Juicy grilled beef patty with fresh vegetables.
$10.00 Qty

Pizza
Cheese-loaded pizza with toppings of your choice.
$8.00 Qty

Pasta
Creamy pasta with mushrooms and chicken.
$7.00 Qty

Sandwich
Whole-grain sandwich with egg and avocado.
$6.00 Qty

Salad
Fresh garden salad with vinaigrette dressing.
$5.00 Qty

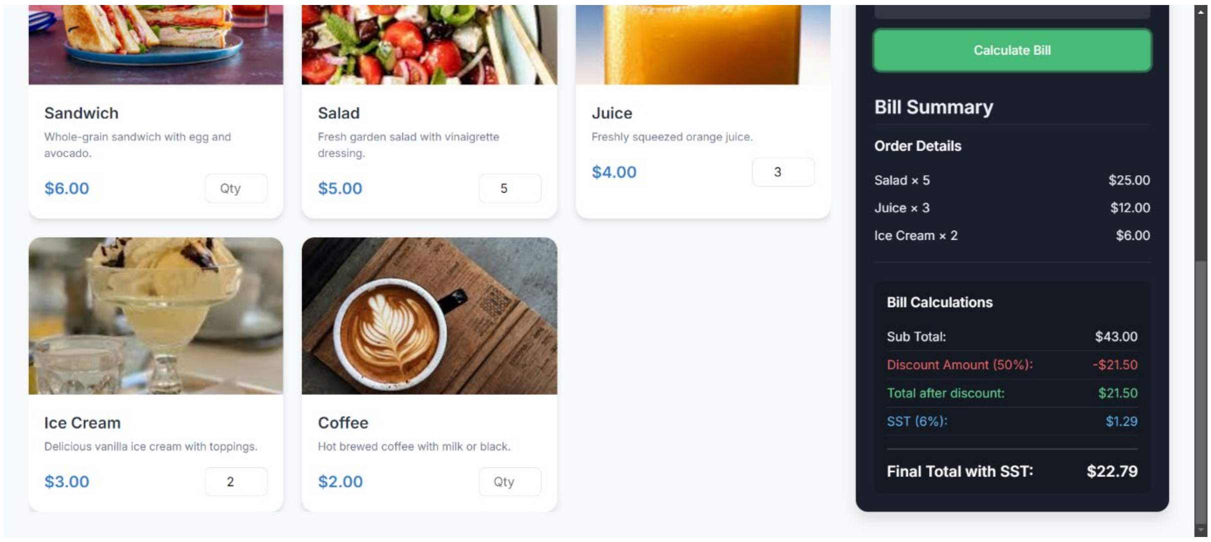
Juice
Freshly squeezed orange juice.
$4.00 Qty

Billing Options
Apply 50% Discount
Calculate Bill

Bill Summary

Please select at least one item to generate bill.

Figure 5:TC5


Sandwich
Whole-grain sandwich with egg and avocado.
$6.00 Qty

Salad
Fresh garden salad with vinaigrette dressing.
$5.00 5

Juice
Freshly squeezed orange juice.
$4.00 3

Ice Cream
Delicious vanilla ice cream with toppings.
$3.00 2

Coffee
Hot brewed coffee with milk or black.
$2.00 Qty

Calculate Bill

Bill Summary

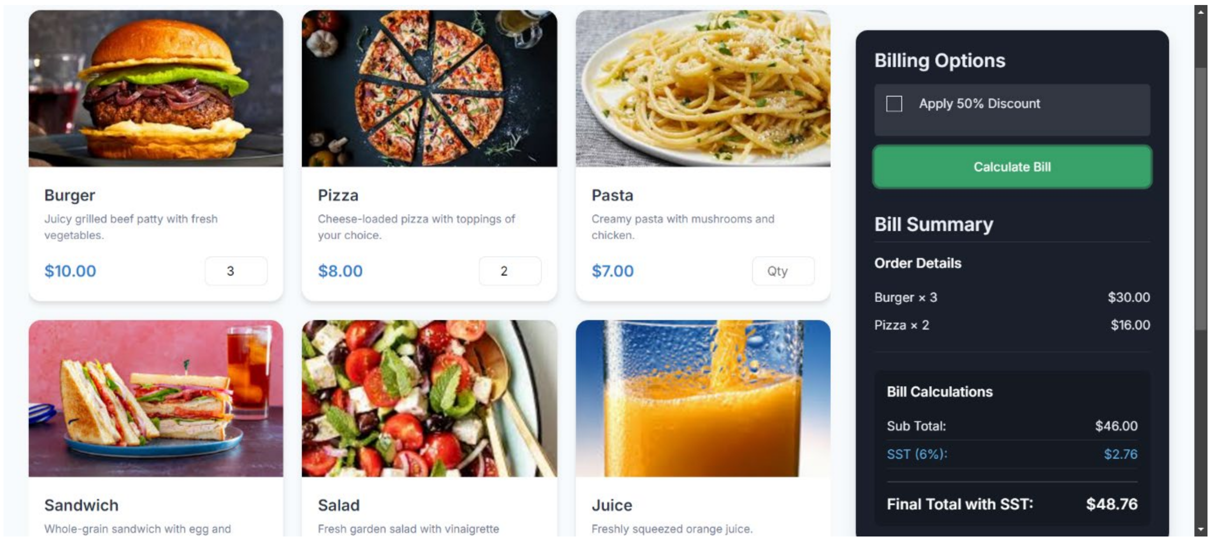
Order Details

Salad × 5	$25.00
Juice × 3	$12.00
Ice Cream × 2	$6.00

Bill Calculations

Sub Total:	$43.00
Discount Amount (50%):	-$21.50
Total after discount:	$21.50
SST (6%):	$1.29
Final Total with SST:	$22.79

Figure 6:TC6


Burger
Juicy grilled beef patty with fresh vegetables.
$10.00
3
Pizza
Cheese-loaded pizza with toppings of your choice.
$8.00
2
Pasta
Creamy pasta with mushrooms and chicken.
$7.00
Qty
Sandwich
Whole-grain sandwich with egg and
Salad
Fresh garden salad with vinaigrette
Juice
Freshly squeezed orange juice.
Billing Options
Apply 50% Discount
Calculate Bill

Bill Summary

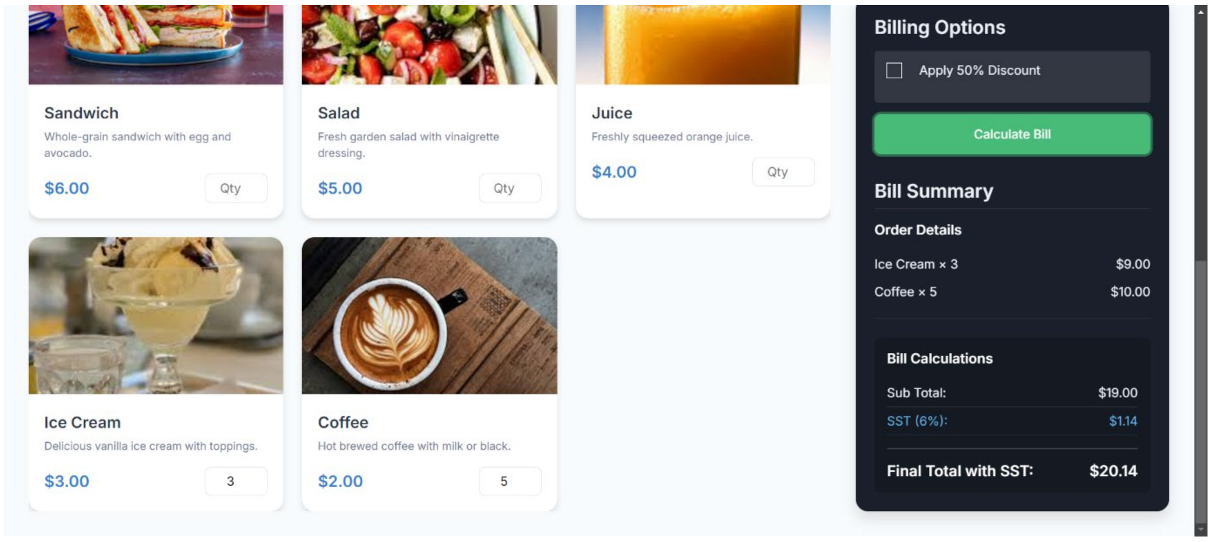
Order Details

Burger × 3	$30.00
Pizza × 2	$16.00

Bill Calculations

Sub Total:	$46.00
SST (6%):	$2.76
Final Total with SST:	$48.76

Figure 7:TC7


Sandwich
Whole-grain sandwich with egg and avocado.
$6.00
Qty
Salad
Fresh garden salad with vinaigrette dressing.
$5.00
Qty
Juice
Freshly squeezed orange juice.
$4.00
Qty
Ice Cream
Delicious vanilla ice cream with toppings.
$3.00
3
Coffee
Hot brewed coffee with milk or black.
$2.00
5
Billing Options
Apply 50% Discount
Calculate Bill

Bill Summary

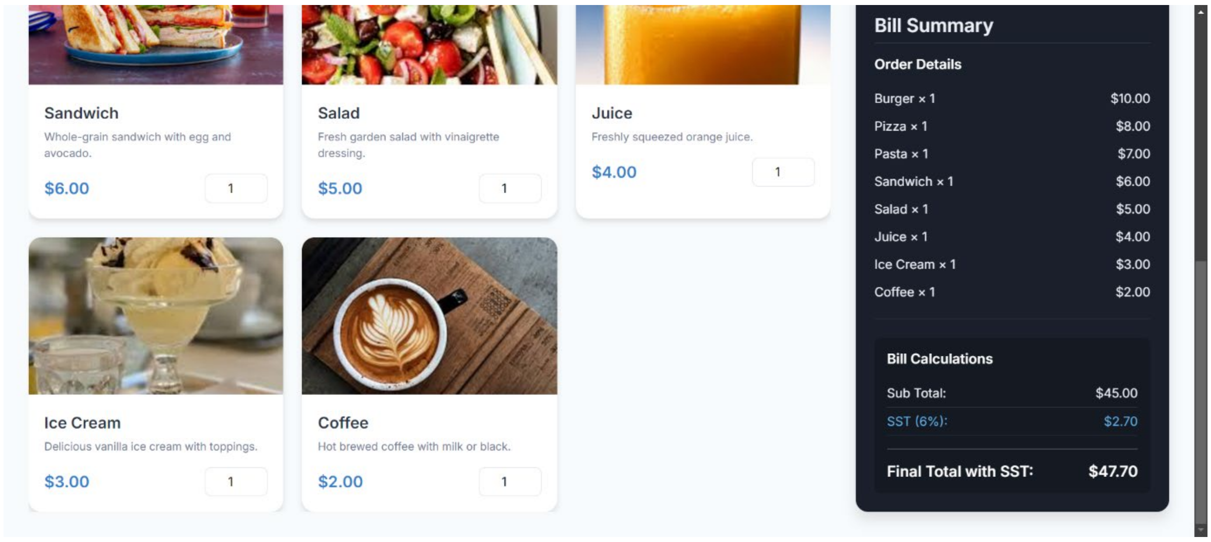
Order Details

Ice Cream × 3	$9.00
Coffee × 5	$10.00

Bill Calculations

Sub Total:	$19.00
SST (6%):	$1.14
Final Total with SST:	$20.14

Figure 8:TC8


Sandwich
Whole-grain sandwich with egg and avocado.
$6.00
1
Salad
Fresh garden salad with vinaigrette dressing.
$5.00
1
Juice
Freshly squeezed orange juice.
$4.00
1
Ice Cream
Delicious vanilla ice cream with toppings.
$3.00
1
Coffee
Hot brewed coffee with milk or black.
$2.00
1

Bill Summary

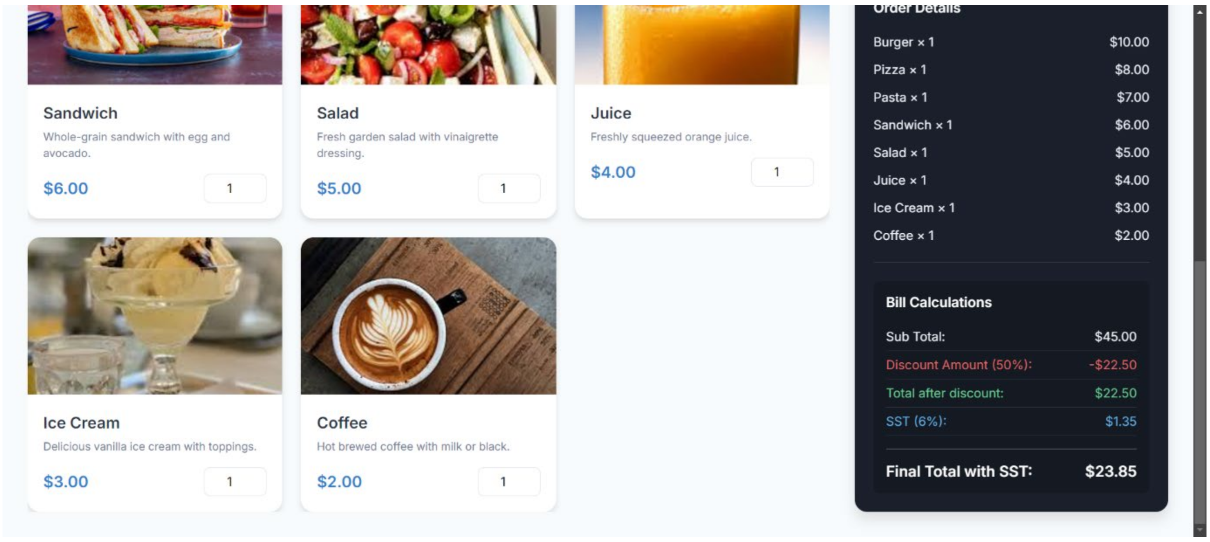
Order Details

Burger × 1	$10.00
Pizza × 1	$8.00
Pasta × 1	$7.00
Sandwich × 1	$6.00
Salad × 1	$5.00
Juice × 1	$4.00
Ice Cream × 1	$3.00
Coffee × 1	$2.00

Bill Calculations

Sub Total:	$45.00
SST (6%):	$2.70
Final Total with SST:	$47.70

Figure 9:TC9


Sandwich
Whole-grain sandwich with egg and avocado.
$6.00
1
Salad
Fresh garden salad with vinaigrette dressing.
$5.00
1
Juice
Freshly squeezed orange juice.
$4.00
1
Ice Cream
Delicious vanilla ice cream with toppings.
$3.00
1
Coffee
Hot brewed coffee with milk or black.
$2.00
1

Order Details

Burger × 1	$10.00
Pizza × 1	$8.00
Pasta × 1	$7.00
Sandwich × 1	$6.00
Salad × 1	$5.00
Juice × 1	$4.00
Ice Cream × 1	$3.00
Coffee × 1	$2.00

Bill Calculations

Sub Total:	$45.00
Discount Amount (50%):	-$22.50
Total after discount:	$22.50
SST (6%):	$1.35
Final Total with SST:	$23.85

Figure 10:TC10

Selenium test script:

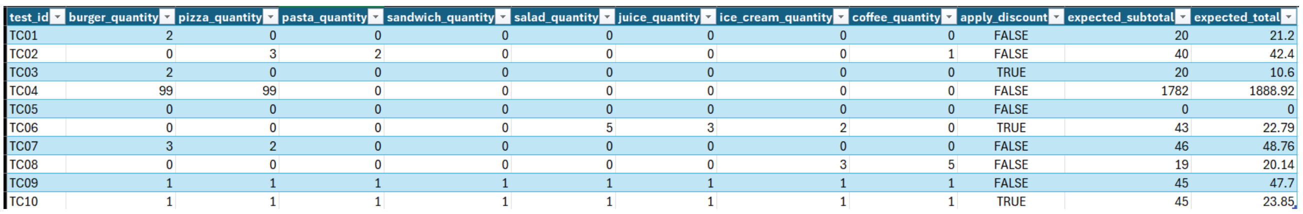
```
from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.common.action_chains import ActionChains

from webdriver_manager.chrome import ChromeDriverManager

import pandas as pd

import time

import os

from datetime import datetime



class FoodBillingTest:

    def __init__(self):

        self.driver = webdriver.Chrome(

            service=Service(ChromeDriverManager().install()))
        self.driver.maximize_window()
        # Create screenshots directory if it doesn't exist
        self.screenshots_dir = 'test_screenshots'
        os.makedirs(self.screenshots_dir, exist_ok=True)

    def take_screenshot(self, test_id, step_name):
        """
        Takes a screenshot and saves it with timestamp and test info.
        Args:
            test_id (str): The ID of the current test case
            step_name (str): Name of the test step where screenshot is taken
        """
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        filename = f"{test_id}_{step_name}_{timestamp}.png"
        filepath = os.path.join(self.screenshots_dir, filename)
        try:
            self.driver.save_screenshot(filepath)
            print(f"Screenshot saved: {filepath}")
        except Exception as e:
            print(f"Failed to take screenshot: {str(e)}")

    def run_test_case(self, test_case):
        try:
            # Take screenshot before starting test
            self.take_screenshot(test_case['test_id'], 'initial_state')

            # Input quantities
            self.input_quantities(test_case)
            self.take_screenshot(test_case['test_id'], 'after_input')
            # Debug print
            print(f"Processing test {test_case['test_id']}")
            print(f"Discount value: {test_case['apply_discount']}")

            # Handle discount with explicit boolean check
            if test_case['apply_discount']:
                print(f"Applying discount for {test_case['test_id']}")
                self.apply_discount()
                self.take_screenshot(test_case['test_id'], 'after_discount')

            # Click calculate button
            calculate_button = WebDriverWait(self.driver, 10).until(
                EC.element_to_be_clickable((By.ID, "calculate"))
            )
            calculate_button.click()

            time.sleep(2)
            self.take_screenshot(test_case['test_id'], 'final_state')

            # Verify results
            if test_case['test_id'] == 'TC05':
                self.verify_error_message()
            else:
                self.verify_results(test_case)

            return True, "Test passed"

        except Exception as e:
            # Take screenshot on failure
            self.take_screenshot(test_case['test_id'], 'failure')
            return False, f"Test failed: {str(e)}"

    # Rest of the class methods remain unchanged
    def setup(self):
        self.driver.get("http://localhost/swe30009-project/index.php")
        time.sleep(2)

    def read_test_cases(self):
        df = pd.read_csv('selenium_test_cases.csv',
                         true_values=['TRUE', 'true', 'True'],
                         false_values=['FALSE', 'false', 'False'])
        print("\nCSV data after conversion:")
        print(df[['test_id', 'apply_discount']].to_string())
        return df

    def apply_discount(self):
        try:
            WebDriverWait(self.driver, 10).until(
                EC.presence_of_element_located(
                    (By.CLASS_NAME, "checkbox-wrapper"))
            )
            checkbox = self.driver.find_element(By.ID, "discount")
            self.driver.execute_script(
                "arguments[0].scrollIntoView(true);", checkbox)
            time.sleep(0.5)

            methods = [
                lambda: self.driver.execute_script(
                    """
                    arguments[0].checked = true;
                    arguments[0].dispatchEvent(new Event('change', { bubbles: true }));
                    """,
                    checkbox
                ),
                lambda: ActionChains(self.driver).move_to_element(
                    checkbox).click().perform(),
                lambda: checkbox.click()
            ]

            for method in methods:
                try:
                    method()
                    time.sleep(0.5)
                    if checkbox.is_selected():
                        print("Checkbox successfully checked")
                        break
                except:
                    continue

            if not checkbox.is_selected():
                raise Exception("Failed to select checkbox after all attempts")

        except Exception as e:
            print(f"Error applying discount: {str(e)}")
            raise e
```

```
  def input_quantities(self, test_case):
    items = {
      'Burger': 'burger_quantity',
      'Pizza': 'pizza_quantity',
      'Pasta': 'pasta_quantity',
      'Sandwich': 'sandwich_quantity',
      'Salad': 'salad_quantity',
      'Juice': 'juice_quantity',
      'Ice Cream': 'ice_cream_quantity',
      'Coffee': 'coffee_quantity'
    }

    for item_name, quantity_field in items.items():
      quantity = int(test_case[quantity_field])
      if quantity > 0:
        try:
          item_element = WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.XPATH,
                                  f"//h2[normalize-
space()='{item_name}']/../..//input[@class='quantity']"))
          )
          item_element.clear()
          item_element.send_keys(str(quantity))
        except Exception as e:
          print(f"Error setting quantity for {item_name}: {str(e)}")

  def verify_error_message(self):
    error_message = WebDriverWait(self.driver, 10).until(
      EC.presence_of_element_located((By.CLASS_NAME, "no-items"))
```

```
        )
        assert "Please select at least one item to generate bill" in error_message.text

    def verify_results(self, test_case):
        if test_case['test_id'] == 'TC05':
            return

        bill_details = WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.ID, "bill-details"))
        )

        time.sleep(1)

        total_element = WebDriverWait(self.driver, 10).until(
            EC.presence_of_element_located((By.CLASS_NAME, "bill-total"))
        )
        actual_total = float(self.get_amount_from_text(total_element.text))
        expected_total = float(test_case['expected_total'])

        assert abs(actual_total - expected_total) < 0.01, \
            f"Total mismatch. Expected: {expected_total}, Got: {actual_total}"

    def get_amount_from_text(self, text):
        amount_str = text.split('$')[1].strip().replace(',', '')
        return float(amount_str)

    def run_all_tests(self):
        test_cases = self.read_test_cases()
        results = []
        for index, test_case in test_cases.iterrows():
            print(f"\nRunning test case {test_case['test_id']}")
            success, message = self.run_test_case(test_case)
            results.append({
                'test_id': test_case['test_id'],
                'success': success,
                'message': message
            })

            self.driver.refresh()
            time.sleep(2)

        return results

    def cleanup(self):
        self.driver.quit()


def main():
    tester = FoodBillingTest()

    try:
        tester.setup()
        results = tester.run_all_tests()

        print("\nTest Results:")
        print("-------------")
        for result in results:
            status = "PASS" if result['success'] else "FAIL"
            print(f"{result['test_id']}: {status} - {result['message']}")

    finally:
        tester.cleanup()



if __name__ == "__main__":
    main()
```

Selenium CSV file:

test_id	burger_quantity	pizza_quantity	pasta_quantity	sandwich_quantity	salad_quantity	juice_quantity	ice_cream_quantity	coffee_quantity	apply_discount	expected_subtotal	expected_total
TC01	2	0	0	0	0	0	0	0	FALSE	20	21.2
TC02	0	3	2	0	0	0	0	1	FALSE	40	42.4
TC03	2	0	0	0	0	0	0	0	TRUE	20	10.6
TC04	99	99	0	0	0	0	0	0	FALSE	1782	1888.92
TC05	0	0	0	0	0	0	0	0	FALSE	0	0
TC06	0	0	0	0	5	3	2	0	TRUE	43	22.79
TC07	3	2	0	0	0	0	0	0	FALSE	46	48.76
TC08	0	0	0	0	0	0	3	5	FALSE	19	20.14
TC09	1	1	1	1	1	1	1	1	FALSE	45	47.7
TC10	1	1	1	1	1	1	1	1	TRUE	45	23.85



test_id	burger_quantity	pizza_quantity	pasta_quantity	sandwich_quantity	salad_quantity	juice_quantity	ice_cream_quantity	coffee_quantity	apply_discount	expected_subtotal	expected_total
TC01	2	0	0	0	0	0	0	0	FALSE	20	21.2
TC02	0	3	2	0	0	0	0	1	FALSE	40	42.4
TC03	2	0	0	0	0	0	0	0	TRUE	20	10.6
TC04	99	99	0	0	0	0	0	0	FALSE	1782	1888.92
TC05	0	0	0	0	0	0	0	0	FALSE	0	0
TC06	0	0	0	0	5	3	2	0	TRUE	43	22.79
TC07	3	2	0	0	0	0	0	0	FALSE	46	48.76
TC08	0	0	0	0	0	0	3	5	FALSE	19	20.14
TC09	1	1	1	1	1	1	1	1	FALSE	45	47.7
TC10	1	1	1	1	1	1	1	1	TRUE	45	23.85

Web Application Source Codes:

db.php:

```
<?php
$host = 'localhost';
$username = 'root';
$password = '';
$dbname = 'food_billing';

// Create connection
$conn = new mysqli($host, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die('Connection failed: ' . $conn->connect_error);
}
?>
```

setup.php:

```
// setup.php
<?php
$host = 'localhost';
$username = 'root';
$password = '';
$dbname = 'food_billing';

// Create connection
$conn = new mysqli($host, $username, $password);

// Check connection
```

```
if ($conn->connect_error) {
    die('Connection failed: ' . $conn->connect_error);
}

// Create the database if it doesn't exist
$sql = "CREATE DATABASE IF NOT EXISTS $dbname";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully or already exists.<br>";
} else {
    die("Error creating database: " . $conn->error);
}

// Use the new database
$conn->select_db($dbname);

// Create the products table if it doesn't exist
$sql = "CREATE TABLE IF NOT EXISTS products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    image_url VARCHAR(255) NOT NULL
)";

if ($conn->query($sql) === TRUE) {
    echo "Table created successfully or already exists.<br>";
} else {
    die("Error creating table: " . $conn->error);
}
```

```
// Delete all existing rows to avoid duplicates
$sql = "TRUNCATE TABLE products";
if ($conn->query($sql) === TRUE) {
   echo "All existing data deleted successfully.<br>";
} else {
   die("Error deleting data: " . $conn->error);
}

// Insert new data with local image paths
$sql = "INSERT INTO products (name, description, price, image_url) VALUES
('Burger', 'Juicy grilled beef patty with fresh vegetables.', 10.00, 'images/Burger.jpg'),
('Pizza', 'Cheese-loaded pizza with toppings of your choice.', 8.00, 'images/Pizza.jpg'),
('Pasta', 'Creamy pasta with mushrooms and chicken.', 7.00, 'images/Pasta.jpg'),
('Sandwich', 'Whole-grain sandwich with egg and avocado.', 6.00, 'images/Sandwich.jpg'),
('Salad', 'Fresh garden salad with vinaigrette dressing.', 5.00, 'images/Salad.jpg'),
('Juice', 'Freshly squeezed orange juice.', 4.00, 'images/Juice.jpg'),
('Ice Cream', 'Delicious vanilla ice cream with toppings.', 3.00, 'images/IceCream.jpg'),
('Coffee', 'Hot brewed coffee with milk or black.', 2.00, 'images/Coffee.jpg')";

if ($conn->query($sql) === TRUE) {
   echo "Sample data inserted successfully.<br>";
} else {
   echo "Error inserting data: " . $conn->error;
}

// Close connection
$conn->close();
?>
```

Index.php:

```
<?php
include 'db.php';

// Fetch all products from the database
$query = "SELECT * FROM products";
$result = $conn->query($query);
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Billing System</title>
    <link rel="stylesheet" href="styles.css">
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600&display=swap"
rel="stylesheet">
</head>
<body>
    <div class="main-container">
        <div class="content-area">
            <div class="header">
                <h1>Food Billing System</h1>
                <p class="subtitle">Select items and quantities to generate bill</p>
            </div>

            <div class="food-list">
                <div class="food-items">
            <?php while ($row = $result->fetch_assoc()): ?>
              <div class="item" data-price="<?= $row['price']; ?>">
                <img src="<?= $row['image_url']; ?>" alt="<?= $row['name']; ?>">
                <div class="item-details">
                  <h2><?= $row['name']; ?></h2>
                  <p class="description"><?= $row['description']; ?></p>
                  <div class="price-quantity">
                    <span class="price">$<?= number_format($row['price'], 2); ?></span>
                    <input type="number" min="0" placeholder="Qty" class="quantity">
                  </div>
                </div>
              </div>
            <?php endwhile; ?>
          </div>
        </div>
      </div>

      <div class="sidebar">
        <div class="bill-card">
          <div class="options-section">
            <h2>Billing Options</h2>
            <div class="checkbox-wrapper">
              <label class="discount-toggle" for="discount">
                <input
                  type="checkbox"
                  id="discount"
                  name="discount"
                  style="width: 20px; height: 20px; margin-right: 10px; cursor: pointer;
position: relative; z-index: 10;"
```

```
                                data-testid="discount-checkbox"
                            >
                            <span class="toggle-label" style="position: relative; z-index: 1;">Apply
50% Discount</span>
                        </label>
                    </div>
                    <button id="calculate" class="calculate-btn">Calculate Bill</button>
                </div>

                <div class="bill-summary">
                    <h2>Bill Summary</h2>
                    <div id="bill-details"></div>
                </div>
            </div>
        </div>
    </div>

    <script src="script.js"></script>
</body>
</html>

<?php
$conn->close();
?>
```

script.js

```
document.addEventListener('DOMContentLoaded', () => {
    const formatCurrency = (amount) => {
        return new Intl.NumberFormat('en-US', {
```

```
        style: 'currency',
        currency: 'USD'
    }).format(amount);
};

document.querySelectorAll('.quantity').forEach(input => {
    input.addEventListener('input', (e) => {
        const value = e.target.value.replace(/[^0-9]/g, '');
        e.target.value = value ? parseInt(value) : '';
    });
});

document.getElementById('calculate').addEventListener('click', () => {
    const items = document.querySelectorAll('.item');
    const discountCheckbox = document.getElementById('discount');
    let subtotal = 0;
    let billDetails = '';
    let hasItems = false;

    // Add debug logging
    console.log('Discount checkbox state:', discountCheckbox.checked);

    items.forEach(item => {
        const price = parseFloat(item.dataset.price);
        const quantity = parseInt(item.querySelector('.quantity').value) || 0;
        const itemName = item.querySelector('h2').innerText;

        if (quantity > 0) {
            hasItems = true;
            let itemTotal = parseFloat((price * quantity).toFixed(2));
            subtotal = parseFloat((subtotal + itemTotal).toFixed(2));
            billDetails += `
                <div class="bill-item">
                    <span>${itemName} × ${quantity}</span>
                    <span>${formatCurrency(itemTotal)}</span>
                </div>`;
        }
    });

    if (!hasItems) {
        document.getElementById('bill-details').innerHTML = `
            <p class="no-items">Please select at least one item to generate bill.</p>`;
        return;
    }

    console.log('Subtotal before discount:', subtotal);

    // Calculate final amounts with precise decimal handling
    let subtotalAfterDiscount = subtotal;
    let discountAmount = 0;

    if (discountCheckbox && discountCheckbox.checked) {
        discountAmount = parseFloat((subtotal * 0.5).toFixed(2));
        subtotalAfterDiscount = parseFloat((subtotal - discountAmount).toFixed(2));
    }

    console.log('Subtotal after discount:', subtotalAfterDiscount);
```

```
const sst = parseFloat((subtotalAfterDiscount * 0.06).toFixed(2));
const total = parseFloat((subtotalAfterDiscount + sst).toFixed(2));

console.log('SST:', sst);
console.log('Final total:', total);

const detailedBill = `
  <div class="order-section">
    <h3>Order Details</h3>
    <div class="bill-items">
      ${billDetails}
    </div>
  </div>

  <div class="calculations-section">
    <h3>Bill Calculations</h3>
    <div class="bill-calculations">
      <div class="subtotal">
        <span>Sub Total:</span>
        <span>${formatCurrency(subtotal)}</span>
      </div>
      ${discountCheckbox && discountCheckbox.checked ? `
        <div class="discount">
          <span>Discount Amount (50%):</span>
          <span>-${formatCurrency(discountAmount)}</span>
        </div>
        <div class="subtotal-after-discount">
          <span>Total after discount:</span>
          <span>${formatCurrency(subtotalAfterDiscount)}</span>
```

```
                </div>
            ` : ''}
            <div class="sst">
                <span>SST (6%):</span>
                <span>${formatCurrency(sst)}</span>
            </div>
        </div>

        <div class="bill-total">
            <span>Final Total with SST:</span>
            <span>${formatCurrency(total)}</span>
        </div>
    </div>`;

    document.getElementById('bill-details').innerHTML = detailedBill;

    if (window.innerWidth <= 992) {
        document.querySelector('.bill-summary').scrollIntoView({
            behavior: 'smooth',
            block: 'start'
        });
    }
  });
});
```

styles.css

```
:root {
  --primary-color: #2D3748;
  --secondary-color: #4A5568;
```

```
    --accent-color: #3182CE;
    --success-color: #48BB78;
    --background-color: #F7FAFC;
    --card-background: #FFFFFF;
    --sidebar-background: #1A202C;
    --text-primary: #2D3748;
    --text-secondary: #718096;
    --text-light: #EDF2F7;
    --border-color: #E2E8F0;
    --shadow-sm: 0 1px 3px rgba(0, 0, 0, 0.1);
    --shadow-md: 0 4px 6px rgba(0, 0, 0, 0.1);
    --shadow-lg: 0 10px 15px rgba(0, 0, 0, 0.1);
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Inter', system-ui, -apple-system, sans-serif;
}

body {
    background-color: var(--background-color);
    min-height: 100vh;
}

.main-container {
    display: grid;
    grid-template-columns: 1fr 400px;
    gap: 2rem;
    max-width: 1600px;
    margin: 0 auto;
    padding: 2rem;
    min-height: 100vh;
}

/* Content Area */
.content-area {
    overflow-y: auto;
}

/* Header Styles */
.header {
    margin-bottom: 2rem;
}

.header h1 {
    color: var(--primary-color);
    font-size: 2.25rem;
    font-weight: 600;
    margin-bottom: 0.5rem;
}

.subtitle {
    color: var(--text-secondary);
    font-size: 1rem;
}
```

```
/* Food Items Grid */
.food-items {
   display: grid;
   grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
   gap: 1.5rem;
}

.item {
   background: var(--card-background);
   border-radius: 1rem;
   overflow: hidden;
   box-shadow: var(--shadow-md);
   transition: transform 0.2s, box-shadow 0.2s;
}

.item:hover {
   transform: translateY(-2px);
   box-shadow: var(--shadow-lg);
}

.item img {
   width: 100%;
   height: 200px;
   object-fit: cover;
}

.item-details {
   padding: 1.25rem;
}
```

```
.item h2 {
   font-size: 1.25rem;
   color: var(--text-primary);
   margin-bottom: 0.5rem;
}

.description {
   color: var(--text-secondary);
   font-size: 0.875rem;
   margin-bottom: 1rem;
   line-height: 1.5;
   display: -webkit-box;
   -webkit-line-clamp: 2;
   -webkit-box-orient: vertical;
   overflow: hidden;
}

.price-quantity {
   display: flex;
   justify-content: space-between;
   align-items: center;
   margin-top: 1rem;
}

.price {
   font-size: 1.25rem;
   font-weight: 600;
   color: var(--accent-color);
```

```
}

.quantity {
    width: 80px;
    padding: 0.5rem;
    border: 1px solid var(--border-color);
    border-radius: 0.5rem;
    text-align: center;
    font-size: 1rem;
}

/* Sidebar Styles */
.sidebar {
    position: sticky;
    top: 2rem;
    height: fit-content;
}

.bill-card {
    background: var(--sidebar-background);
    border-radius: 1rem;
    padding: 1.5rem;
    color: var(--text-light);
    box-shadow: var(--shadow-lg);
}

/* Options Section */
.options-section {
    margin-bottom: 2rem;
```

```
}

.options-section h2 {
   font-size: 1.5rem;
   margin-bottom: 1rem;
}

.discount-toggle {
   display: flex;
   align-items: center;
   gap: 0.75rem;
   margin-bottom: 1rem;
   cursor: pointer;
   padding: 0.5rem 0;
}

.discount-toggle input[type="checkbox"] {
   width: 1.25rem;
   height: 1.25rem;
   cursor: pointer;
}

.calculate-btn {
   width: 100%;
   padding: 1rem;
   background: var(--success-color);
   border: none;
   border-radius: 0.5rem;
   color: white;
```

```
    font-size: 1rem;
    font-weight: 500;
    cursor: pointer;
    transition: background-color 0.2s;
}

.calculate-btn:hover {
    background: #38A169;
}

/* Bill Summary Styles */
.bill-summary {
    margin-top: 2rem;
}

.bill-summary h2 {
    font-size: 1.5rem;
    margin-bottom: 1rem;
    padding-bottom: 0.5rem;
    border-bottom: 1px solid rgba(255, 255, 255, 0.1);
}

/* Order Section Styles */
.order-section h3,
.calculations-section h3 {
    font-size: 1.1rem;
    margin-bottom: 1rem;
    color: #fff;
}
```

```
.order-section {
   margin-bottom: 1.5rem;
}

/* Bill Items Styles */
.bill-items {
   margin-bottom: 1rem;
   padding-bottom: 1rem;
   border-bottom: 1px solid rgba(255, 255, 255, 0.1);
}

.bill-item {
   display: flex;
   justify-content: space-between;
   padding: 0.5rem 0;
   color: var(--text-light);
}

/* Calculations Section Styles */
.calculations-section {
   background: rgba(0, 0, 0, 0.2);
   border-radius: 0.5rem;
   padding: 1rem;
}

.bill-calculations {
   margin-bottom: 1rem;
}
```

```
.bill-calculations > div {
    display: flex;
    justify-content: space-between;
    padding: 0.5rem 0;
    border-bottom: 1px solid rgba(255, 255, 255, 0.1);
}

.subtotal {
    color: var(--text-light);
}

.discount {
    color: #F56565;
}

.checkbox-wrapper {
    padding: 10px;
    margin-bottom: 15px;
    background: rgba(255, 255, 255, 0.1);
    border-radius: 5px;
}

.discount-toggle {
    display: flex;
    align-items: center;
    cursor: pointer;
    padding: 5px;
    position: relative;
```

```
}

.discount-toggle input[type="checkbox"] {
   opacity: 1;
   position: relative;
   cursor: pointer;
   -webkit-appearance: none;
   -moz-appearance: none;
   appearance: none;
   outline: none;
   border: 1px solid #fff;
   background: transparent;
}

.discount-toggle input[type="checkbox"]:checked {
   background: #4CAF50;
}

.subtotal-after-discount {
   color: #68D391;
}

.sst {
   color: #63B3ED;
}

.bill-total {
   margin-top: 1rem;
   padding-top: 1rem;
```

```
    border-top: 2px solid rgba(255, 255, 255, 0.2);
    font-size: 1.2rem;
    font-weight: 600;
    display: flex;
    justify-content: space-between;
    color: #fff;
}

.no-items {
    color: #FC8181;
    text-align: center;
    padding: 1rem;
    font-style: italic;
}

/* Input Focus Styles */
.quantity:focus {
    outline: none;
    border-color: var(--accent-color);
    box-shadow: 0 0 0 3px rgba(49, 130, 206, 0.1);
}

/* Animation */
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}

#bill-details {
    animation: fadeIn 0.3s ease-out;
}

/* Responsive Design */
@media (max-width: 1200px) {
    .main-container {
        grid-template-columns: 1fr 350px;
        padding: 1.5rem;
    }
}

@media (max-width: 992px) {
    .main-container {
        grid-template-columns: 1fr;
    }

    .sidebar {
        position: static;
        margin-top: 2rem;
    }

    .food-items {
        grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
    }

    .bill-card {
        max-width: 600px;
        margin: 0 auto;
    }
```

```
}

@media (max-width: 768px) {
  .main-container {
    padding: 1rem;
  }

  .header h1 {
    font-size: 1.75rem;
  }

  .food-items {
    grid-template-columns: 1fr;
  }

  .item-details {
    padding: 1rem;
  }

  .bill-card {
    padding: 1rem;
  }

  .bill-calculations > div {
    font-size: 0.9rem;
  }

  .bill-total {
    font-size: 1.1rem;
  }
}

@media (max-width: 480px) {
  .main-container {
    padding: 0.5rem;
  }

  .header h1 {
    font-size: 1.5rem;
  }

  .subtitle {
    font-size: 0.9rem;
  }

  .item {
    border-radius: 0.5rem;
  }

  .item img {
    height: 160px;
  }

  .item h2 {
    font-size: 1.1rem;
  }

  .description {
    font-size: 0.8rem;
  }

  .price {
    font-size: 1.1rem;
  }

  .quantity {
    width: 70px;
    font-size: 0.9rem;
  }

  .calculations-section {
    padding: 0.75rem;
  }

  .order-section h3,
  .calculations-section h3 {
    font-size: 1rem;
  }
}

/* Print Styles */
@media print {
  .main-container {
    display: block;
    padding: 0;
  }
```

```
.content-area {
   display: none;
}

.sidebar {
   width: 100%;
   position: static;
}

.options-section {
   display: none;
}

.bill-card {
   box-shadow: none;
   padding: 0;
   background: none;
   color: black;
}

.bill-summary h2 {
   color: black;
   border-bottom-color: #000;
}

.bill-items {
   border-bottom-color: #000;
}
```

```
    .bill-item {
        color: black;
    }

    .calculations-section {
        background: none;
        border: 1px solid #000;
    }

    .bill-calculations > div {
        border-bottom-color: #000;
        color: black;
    }

    .subtotal, .discount, .subtotal-after-discount, .sst, .bill-total {
        color: black;
    }

    .bill-total {
        border-top-color: #000;
    }
}

/* Hover Effects */
@media (hover: hover) {
    .item-details:hover {
        background-color: rgba(247, 250, 252, 0.03);
    }
```

```css
    .discount-toggle:hover {
        background-color: rgba(255, 255, 255, 0.05);
        border-radius: 0.5rem;
    }

    .calculate-btn:hover {
        background-color: #38A169;
        transform: translateY(-1px);
    }
}

/* Accessibility Improvements */
.quantity:focus {
    outline: none;
    border-color: var(--accent-color);
    box-shadow: 0 0 0 3px rgba(49, 130, 206, 0.1);
}

.discount-toggle input[type="checkbox"]:focus {
    outline: 2px solid var(--accent-color);
    outline-offset: 2px;
}

.calculate-btn:focus {
    outline: none;
    box-shadow: 0 0 0 3px rgba(72, 187, 120, 0.5);
}

/* Additional Utility Classes */
```

```
.visually-hidden {
   position: absolute;
   width: 1px;
   height: 1px;
   padding: 0;
   margin: -1px;
   overflow: hidden;
   clip: rect(0, 0, 0, 0);
   white-space: nowrap;
   border: 0;
}

.text-error {
   color: #FC8181;
}

.text-success {
   color: #68D391;
}

.text-info {
   color: #63B3ED;
}
```