Test Cases

1. Black Box Testing

1.1 Important Controllers and tests to be done

Store Owner Controller

Control class to test: **Store Owner Controller**

The *Store Owner Controller* class manages the functionality for store owners to add and update food listings within the NTUFoodie application. Key functions in this class, such as *Add Food* and *Add Discount*, are responsible for enabling store owners to input accurate food details, including *Name*, *Price*, *Type*, *Cuisine*, *Description*, and *Discount*.

User Controller

Control class to test: **User Controller**

The *User Controller* class manages the core functionalities for user account management within the NTUFoodie application. This includes operations for user sign-up and login. The class ensures that users input accurate details such as *First Name, Last Name, Username, Email Address, Password, Confirm Password, Role, Canteen Name (if the input role is Store Owner)* and *Store Name (if the input role is Store Owner)* with validation checks to confirm that input fields meet format and security requirements.

Equivalence Class Partitioning

This approach divides input data into valid and invalid classes to cover as many variations with minimal test cases.

Boundary Value Testing

This approach targets the boundaries around valid input ranges, testing edge cases to see if the application handles them correctly.

1.2 Parameters

1.2.1 Add Food Function Fields Parameters(Store Owner Controller)

For *Name* Field:

- **Valid Classes:**
  - Any valid strings with no more than 200 characters
- **Invalid Classes:**
  - No input
  - More than 200 characters (automatically prevented by system)

For *Price* Field:

- **Valid Classes:**
  - Numerical values above -1
- **Invalid Classes:**
  - Negative value
  - Non-numeric values (e.g., letters or special characters).

For *Type* Field:

- **Valid Classes:**
  - Any drop down option except “-”
- **Invalid Classes:**
  - “-” option

For *Cuisine* Field:

- **Valid Classes:**
    - Any drop down option except “-”
- **Invalid Classes:**
    - “-” option

For *Description* Field:

- **Valid Classes:**
    - Any valid strings with no more than 1000 characters
- **Invalid Classes:**
    - No input
    - More than 1000 characters (automatically prevented by system)

1.2.2 Add Discount Function Parameters(Store Owner Controller)

For *Price* Field:

- Valid values: 0 (lower bound), 99(upper bound)
- Invalid values: -1(lower bound), 100(upper bound), Non-numeric values (e.g., letters or special characters)

1.2.3 Sign Up Function Fields Parameters(User Controller)

For *First Name* Field:

- **Valid Classes:**
    - Any valid strings
- **Invalid Classes:**

    - No input

For *Last Name* Field:

- **Valid Classes:**
    - Any valid strings
- **Invalid Classes:**
    - No input

For *Username* Field:

- **Valid Classes:**
    - Any valid strings with no more than 150 characters
- **Invalid Classes:**
    - No input
    - An existing user’s username
    - More than 150 characters (automatically prevented by system)

For *Email Address* Field:

- **Valid Classes:**
    - Any valid strings with no more than 254 characters
    - No input (System allows account creation with no email)
- **Invalid Classes:**
    - Invalid email address
    - More than 254 characters (automatically prevented by system)

For *Password* Field:

- **Valid Classes:**

    - Any valid strings matching with input for Confirm Password Field
- **Invalid Classes:**
    - No input
    - Strings not matching input for Confirm Password Field

For *Confirm Password* Field:

- **Valid Classes:**
    - Any valid strings matching with input for Password Field
- **Invalid Classes:**
    - No input
    - Strings not matching input for Password Field

For *Role* Field:

- **Valid Classes:**
    - Any drop down
- **Invalid Classes (None)**

For *Canteen Name* Field:

- **Valid Classes:**
    - Any drop down option except “-”
- **Invalid Classes(None)**

For *Store Name* Field:

- **Valid Classes:**
    - Any valid strings
- **Invalid Classes:**

- No input

1.2.4 Login Function Parameters(User Controller)

For *Username* Field:

- **Valid Classes:**
  - A registered user's username
- **Invalid Classes:**
  - Empty input,
  - A username that is not registered in the system

For *Password* Field:

- **Valid Classes:**
  - A valid password
- **Invalid Classes:**
  - Empty input,
  - Wrong password

1.3 Test Cases

1.3.1 Add Food Function Tests(Store Owner Controller)

Input parameter: Name, Price, Type, Cuisine, Description (**Fields in red belongs to invalid class**)

Test Case	Descripti	Test Type	Priority	Input	Expected Output	Actual Output

ID	on					
TC_AF_001	Valid food item addition	Equivalence Partitioning	High	**(All valid inputs)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice”	System outputs “Listing Added Successfully!”	System outputs “Listing Added Successfully!”
TC_AF_002	Empty name field	Negative Testing	High	**(Invalid Name, others valid)** Name: “” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice”	System outputs “Please fill out this field” for name section	System outputs “Please fill out this field” for name section
TC_AF_003	Negative price value	Boundary Value Testing	High	**(Invalid Price, others valid)** Name: “Chicken Rice” Price: -1 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice”	System outputs “Error in price: * Price cannot be negative.”	System outputs “Error in price: * Price cannot be negative.”
TC_AF_004	Non-numeric place	Data Type Testing	Medium	**(Invalid Price, others valid)** Name: “Chicken Rice” Price: “e” Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese	System outputs “Please enter a number.”	System outputs “Please enter a number.”

				Chicken Rice”		
TC_AF_005	Invalid type selection	Negative Testing	High	**(Invalid Type, others valid)** Name: “Chicken Rice” Price: 0 Type: - Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice”	System outputs “Error in type: * This field is required.”	System outputs “Error in type: * This field is required.”
TC_AF_006	Invalid cuisine selection	Negative Testing	High	**(Invalid Cuisine, others valid)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: - Description: “Tasty Hainanese Chicken Rice”	System outputs “Error in type: * This field is required.”	System outputs “Error in type: * This field is required.”
TC_AF_007	Empty description	Negative Testing	Medium	**(Invalid Description, others valid)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “”	System outputs “Please fill out this field” for description section	System outputs “Please fill out this field” for description section

1.3.2 Add Discount Function (Store Owner Controller)

Input parameter: Discount (**Fields in red belongs to invalid class**)

Test Case ID	Description	Test Type	Priority	Input	Expected	Actual Output

					Output	
TC_AD_001	Valid discount (lower bound)	Boundary Value Testing	High	**(Lower bound valid input)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice” Discount: 0	System outputs “Listing Updated Successfully!”	System outputs “Listing Updated Successfully!”
TC_AD_002	Valid discount (upper bound)	Boundary Value Testing	High	**(Upper bound valid input)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice” Discount: 99	System outputs “Listing Updated Successfully!”	System outputs “Listing Updated Successfully!”
TC_AD_003	Invalid discount (below lower bound)	Boundary Value Testing	High	**(Lower bound invalid input)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice” Discount: -1	System outputs “An error occurred. Please check the form and try again.” and “Discount rate cannot be negative.“ message underneath	System outputs “An error occurred. Please check the form and try again.” and “Discount rate cannot be negative.“ message underneath

					discount section	discount section
TC_AD_004	Invalid discount (above upper bound)	Boundary Value Testing	High	**(Upper bound invalid input)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice” Discount: 100	System outputs “An error occurred. Please check the form and try again.” and “Discount rate must be less than 100%.“ message underneath discount section	System outputs “An error occurred. Please check the form and try again.” and “Discount rate must be less than 100%.“ message underneath discount section
TC_AD_005	Non-numeric discount value	Data Type Testing	Medium	**(String input)** Name: “Chicken Rice” Price: 0 Type: Non-halal Cuisine: Chinese Description: “Tasty Hainanese Chicken Rice” Discount: “e”	System outputs “Please enter a number.”	System outputs “Please enter a number.”

1.3.3 Sign Up Function (User Controller)

Input parameter: First Name, Last Name, Username, Email Address, Password, Confirm Password, Role, Canteen Name, Store Name (**Fields in red belongs to invalid class, Fields in green belongs to valid class**)

Test Case ID	Description	Test Type	Priority	Input	Expected Output	Actual Output
TC_SU_001	Valid registration	Equivalence Partitioning	High	**(All valid inputs)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.com” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	Redirects to the login page. System outputs “Account was created for storeowner”	Redirects to the login page. System outputs “Account was created for storeowner”
TC_SU_002	Empty first name	Negative Testing	High	**(Invalid First Name, others valid)** First Name: “” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.com” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “Please fill out this field” for first name section	System outputs “Please fill out this field” for first name section
TC_SU_003	Empty last	Negative	High	**(Invalid Last Name, others valid)**	System	System outputs

	name	Testing		First Name: “store” Last Name: “” Username: “storeowner” Email Address: “storeowner@gmail.com” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	outputs “Please fill out this field” for last name section	“Please fill out this field” for last name section
TC_SU_004	Empty username	Negative Testing	High	**(Invalid Username, others valid)** First Name: “store” Last Name: “owner” Username: “” Email Address: “storeowner@gmail.com” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “Please fill out this field” for username section	System outputs “Please fill out this field” for username section
TC_SU_005	Duplicate username	Negative Testing	High	**(Invalid Username, others valid)** First Name: “store” Last Name: “owner” Username: “teststoreowner1” Email Address: “storeowner@gmail.com” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner	System outputs “username: A user with that username already exists.”	System outputs “username: A user with that username already exists.”

				Canteen Name: Quad Cafe Store Name: “Store”		
TC_SU_006	Invalid email (missing @)	Data Format Testing	High	**(Invalid Email Address, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “Please include an ‘@’ in the email address. ‘storeowner’ is missing an ‘@’.” for email address section	System outputs “Please include an ‘@’ in the email address. ‘storeowner’ is missing an ‘@’.” for email address section
TC_SU_007	Invalid email (incomplete domain)	Data Format Testing	High	**(Invalid Email Address, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “Please enter a part following ‘@’. ‘storeowner@’ is incomplete.” for email address section	System outputs “Please enter a part following ‘@’. ‘storeowner@’ is incomplete.” for email address section
TC_SU_008	Invalid email (missing TLD)	Data Format Testing	High	**(Invalid Email Address, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner”	System outputs “email: Enter a valid email address.”	System outputs “email: Enter a valid email address.”

				Email Address: “storeowner@gmail” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”		
TC_SU_009	Invalid email (incomplete TLD)	Data Format Testing	High	**(Invalid Email Address, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.” Password: “testpassword” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “‘.’ is used at a wrong position in ‘gmail.’.” for email address section	System outputs “‘.’ is used at a wrong position in ‘gmail.’.” for email address section
TC_SU_010	Empty password	Negative Testing	High	**(Invalid Password, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.” Password: “” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “password1: This field is required.”	System outputs “password1: This field is required.”
TC_SU_011	Password mismatch	Data Validation	High	**(Password and Confirm Password not matching, others valid)**	System outputs	System outputs “password2:

				First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.” Password: “testpass” Confirm Password: “testpassword” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	“password2: The two password fields didn’t match.”	The two password fields didn’t match.”
TC_SU_012	Empty confirm password	Negative Testing	High	**(Invalid Confirm Password, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.” Password: “testpassword” Confirm Password: “” Role: Store Owner Canteen Name: Quad Cafe Store Name: “Store”	System outputs “password2: This field is required.”	System outputs “password2: This field is required.”
TC_SU_013	Empty store name	Negative Testing	Mediu m	**(Invalid Store Name, others valid)** First Name: “store” Last Name: “owner” Username: “storeowner” Email Address: “storeowner@gmail.” Password: “testpassword” Confirm Password: “” Role: Store Owner Canteen Name: Quad Cafe Store Name: “”	System outputs “Please fill out this field” for store section	System outputs “Please fill out this field” for store section

1.3.4 Sign In Function (User Controller)

Input parameter: Username, Password (**Fields in red belongs to invalid class**)

Test Case ID	Description	Test Type	Priority	Input	Expected Output	Actual Output
TC_L_001	Valid login	Equivalence Partitioning	High	**(All valid inputs)** Name: “teststoreowner” Password: “testpassword”	Redirects to home page for store owners	Redirects to home page for store owners
TC_L_002	Empty username	Negative Testing	High	**(Invalid Name, Valid Password)** Name: “” Password: “testpassword”	System outputs “Invalid username or password.”	System outputs “Invalid username or password.”
TC_L_003	Invalid username	Negative Testing	High	**(Invalid Name, Valid Password)** Name: “teststoreowners” Password: “testpassword”	System outputs “Invalid username or password.”	System outputs “Invalid username or password.”
TC_L_004	Empty password	Negative Testing	High	**(Invalid Password, Valid Name)** Name: “teststoreowner” Password: “”	System outputs “Invalid username or password.”	System outputs “Invalid username or password.”

TC_L_005	Wrong password	Negative Testing	High	**(Invalid Password, Valid Name)** Name: “teststoreowner” Password: “wrongpass”	System outputs “Invalid username or password.”	System outputs “Invalid username or password.”

2. White Box Testing

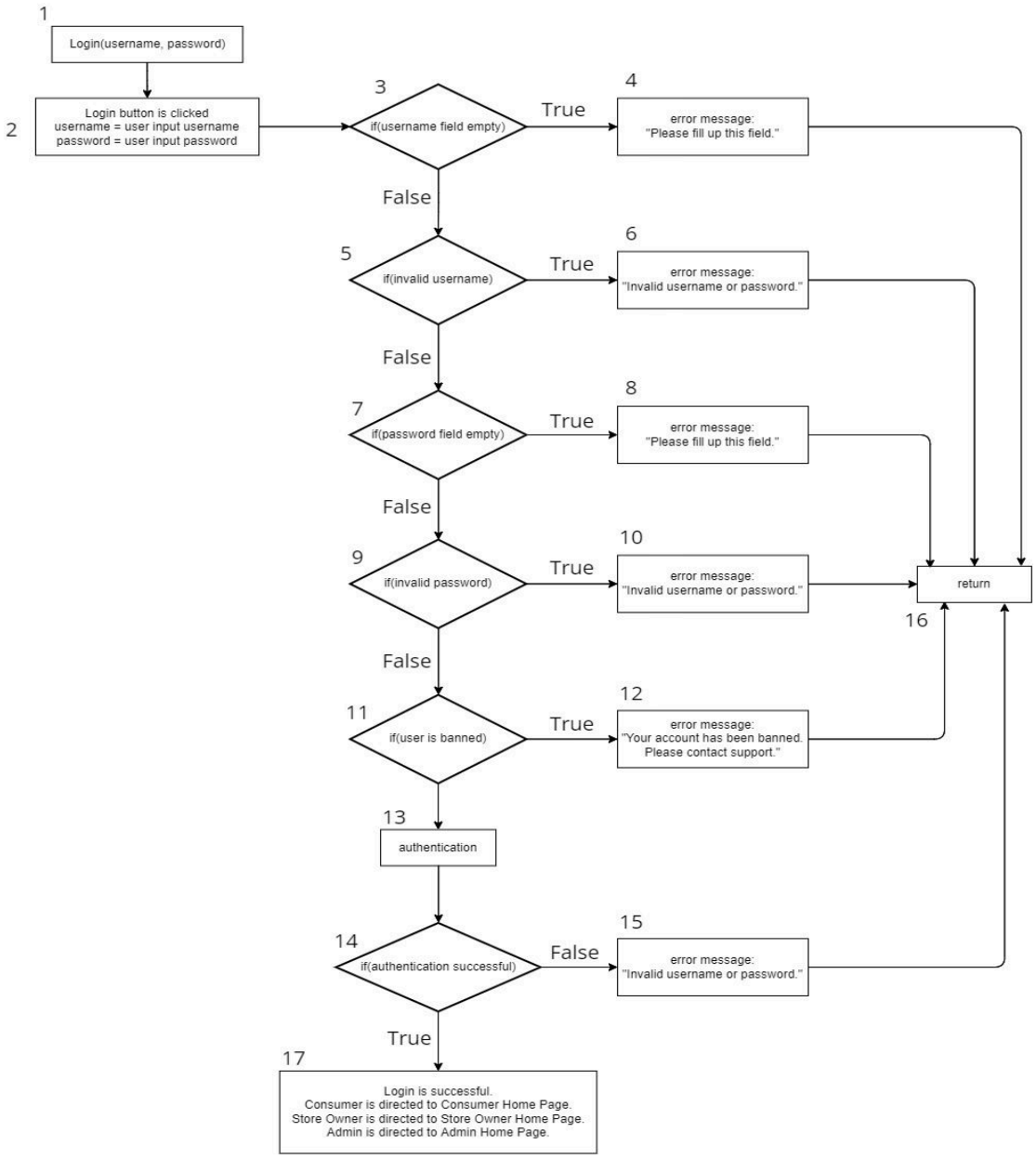
- Determining the expected outputs for chosen inputs.
- Constructs tests with chosen inputs (Pytest).
- Compares actual outputs with chosen inputs.
- Code snippets are either written in python or HTML

2.1 Control Flow for Login

Basic Path Testing

Cyclomatic Complexity = |Decision points| + 1 = 6 + 1 = 7
Basis Paths:

1. Baseline path: 1,2,3,5,7,9,11,13,14,17
2. Basis path 2: 1,2,3,4,16
3. Basis path 3: 1,2,3,5,6,16
4. Basis path 4: 1,2,3,5,7,8,16
5. Basis path 5: 1,2,3,5,7,9,10,16
6. Basis path 6: 1,2,3,5,7,9,11,12,16
7. Basis path 7: 1,2,3,5,7,9,11,13,14,15,16

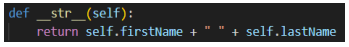

1
Login(username, password)
2
Login button is clicked
username = user input username
password = user input password
3
if(username field empty)
True
4
error message:
"Please fill up this field."
False
5
if(invalid username)
True
6
error message:
"Invalid username or password."
False
7
if(password field empty)
True
8
error message:
"Please fill up this field."
False
9
if(invalid password)
True
10
error message:
"Invalid username or password."
return
16
False
11
if(user is banned)
True
12
error message:
"Your account has been banned.
Please contact support."
13
authentication
14
if(authentication successful)
False
15
error message:
"Invalid username or password."
True
17
Login is successful.
Consumer is directed to Consumer Home Page.
Store Owner is directed to Store Owner Home Page.
Admin is directed to Admin Home Page.

2.2 Models Test:

2.2.1 Consumer Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_CM_001	Create consumer object	High	
			Input particulars needed:
			Username = “testuser”
			firstName = ‘Jane’
			lastName = ‘Tan’
			Email = “jane.tan@gmail.com”
			profilePic = “default_profile.png”
				def test_consumer_creation(self):
				
				self.assertIsInstance(self.consumer, Consumer) # Test if Consumer instance is created
				print("\nConsumer instance created successfully.")
					Consumer instance created successfully.
					---------- test_consumer_creation passed successfully ----------

TC_CM_002	Consumer string display	High	firstName + lastName = Jane Tan	 def test_consumer_str_display(self): expected_output = "Jane Tan" self.assertEqual(str(self.consumer), expected_output) print(f"Consumer string representation: {str(self.consumer)}")	Consumer string representation: Jane Tan ---------- test_consumer_str_display passed successfully ----------

2.2.2 Location Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_LM_001	Create location object	High	`cls.location = Location.objects.create(`
			`  canteenName="Canteen 20",`
			`  address="Opp Hall 19 Bus Stop",`
			`)`
			Input needed:
			canteenName = “Canteen 20”
			address = “Opp Hall 19 Bus Stop”
				def test_location_creation(self):
				
				self.assertIsInstance(self.location, Location)
				
				self.assertEqual(self.location.canteenName, "Canteen 20")
				
				self.assertEqual(self.location.address, "Opp Hall 19 Bus Stop")
				print("Location instance created successfully with canteenName and address.")
				
				location.canteenName = “Canteen 20” & location.address = “Opp Hall 19 Bus Stop”
					Location instance created successfully with canteenName and address.
					---------- test_location_creation passed successfully ----------

TC_LM_002	Location String display	High	Address = “Opp Hall 19 Bus Stop”	`def __str__(self):`
				`  return str(self.address)`
				
				def test_location_address_display(self):
				expected_output = "Opp Hall 19 Bus Stop"
				
				self.assertEqual(str(self.location), expected_output)
				print(f"Location string representation:
					Location string representation: Opp Hall 19 Bus Stop
					---------- 	
					test_location_address_display passed successfully ----------

				{str(self.location)}")	

2.2.3 Canteen Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_CAM_001	Create canteen object	High	`cls.location = Location.objects.create(`
			`  canteenName="Canteen 20",`
			`  address="Opp Hall 19 Bus Stop",`
			`)`
			`cls.canteen = Canteen.objects.create(`
			`  location=cls.location`
			`)`
			Canteen uses the location object.
				def test_canteen_creation(self):
				
				self.assertIsInstance(self.canteen , Canteen) # Test if Canteen instance is created
				print("Canteen instance created successfully.")
				
				Canteen object should be created successfully with the required inputs.
					Canteen instance created successfully.
					---------- test_canteen_creation passed successfully ----------

TC_CAM_002	Canteen String display	High	canteenName = “Canteen 20”	`def __str__(self):`
				`  return self.location.canteenName`
				
				def test_canteen_str_display(self):
				expected_output = "Canteen 20"
				
				self.assertEqual(str(self.canteen.location.canteenName), expected_output)
				print(f"Canteen string
					Canteen string representation: Canteen 20
					---------- test_canteen_str_display passed successfully ----------

				representation: {str(self.canteen.location.canteenName)}")	

2.2.4 Store Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_SM_001	Create store object	High	`cls.location = Location.objects.create(`
			`  canteenName="Canteen 20",`
			`  address="Opp Hall 19 Bus Stop",`
			`)`
			`cls.canteen = Canteen.objects.create(`
			`  location=cls.location`
			`)`
			`cls.store = Store.objects.create(`
			`  storeName="Korean Store",`
			`  canteen=cls.canteen,`
			`)`
			
			Store uses the canteen object.
				def test_store_creation(self):
				self.assertIsInstance(self.store, Store)
				
				self.assertEqual(self.store.storeName, "Korean Store")
				self.assertEqual(self.store.canteen, self.canteen)
				print("Store instance created successfully with storeName and canteen.")
				
				Store object should be created successfully with the required inputs.
					Store instance created successfully with storeName and canteen.
					---------- test_store_creation passed successfully ----------

TC_SM_002	Store String display	High	Korean Store @ Canteen 20	`def __str__(self):`
				`  return self.storeName + " @ " + self.canteen.location.canteenName`
				
				def test_store_str_display(self):
				expected_output = "Korean Store @ Canteen 20"
				self.assertEqual(str(self.store), expected_output)
				print(f"Store string representation:
					Store string representation: Korean Store @ Canteen 20
					---------- 	
					test_store_str_display passed successfully ----------

				{str(self.store)}")	

2.2.5 Store Owner Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output
TC_SOM_001	Create store owner object	High	cls.user = User.objects.create_user(username='storeownertest', password="password") cls.storeOwner = StoreOwner.objects.create(user=cls.user, store=cls.store, firstName="store", lastName="owner", email="storeownertest@gmail.com", profilePic="default_profile.png",) Input particulars needed to create a store owner object. Store uses the store object created in the Store model.	def test_storeOwner_creation(self): self.assertIsInstance(self.storeOwner, StoreOwner) self.assertEqual(self.storeOwner.user, self.user) self.assertEqual(self.storeOwner.store, self.store) self.assertEqual(self.storeOwner.firstName, "store") self.assertEqual(self.storeOwner.lastName, "owner") print("StoreOwner instance created successfully with user, store, firstName, and lastName.") New store owner object should have the same particulars declared in the input.	StoreOwner instance created successfully with user, store, firstName, and lastName. ---------- test_storeOwner_creation passed successfully ----------

TC_SOM_002	Store owner String display	High	firstName + lastName = “store owner”	`def __str__(self):`
				`  return self.firstName + " " + self.lastName`
				
				def test_storeOwner_str_display(self):
				  expected_output = "store owner"
				
				self.assertEqual(str(self.storeOwner), expected_output)
				  print(f"StoreOwner string representation: {str(self.storeOwner)}")
					StoreOwner string representation: store owner
					---------- 	
					test_storeOwner_str_display passed successfully ----------

2.2.6 Food Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_FM_001	Create food object	High	def setUpTestData(cls):
			  super().setUpTestData()
			  cls.food = Food.objects.create(
			  store=cls.store,
			  foodName="Japchae",
			  price=5.50,
			  type="Non-Halal",
			  cuisine="Korean",
			  description="Delicious Japchae",
			  rating=4.5,
			  isDiscounted=True,
			  discountRate=10,
			  )
			
			Input particulars needed to create a food object. Store uses the store
				def test_food_creation(self):
				  self.assertIsInstance(self.food, Food)
				
				self.assertEqual(self.food.store.storeName, self.store.storeName)
				
				self.assertEqual(self.food.store.canteen.location.canteenName, self.store.canteen.location.canteenName)
				
				self.assertEqual(self.food.foodName, "Japchae")
				  print("Store name: ", self.food.store.storeName)
				  print("Canteen name: ", self.food.store.canteen.location.canteenName)
					Store name: Korean Store
					Canteen name: Canteen 20
					Food instance created successfully with foodName, store, and canteen information.
					---------- test_food_creation passed successfully ----------

			object created in the Store model.	print("Food instance created successfully with foodName, store, and canteen information.") New food object should also have the same storeName from store object and canteenName from location object.	

TC_FM_002	Food String display	High	foodName = “Japchae”
			storeName = “Korean Store”
			canteenName = “Canteen 20”
				def __str__(self):
				return self.foodName + " from " + self.store.storeName + " @ " + self.store.canteen.location.canteenName
				
				def test_food_str_display(self):
				expected_output = "Japchae from Korean Store @ Canteen 20"
				self.assertEqual(str(self.food), expected_output)
				print(f"Food string representation: {str(self.food)}")
					Food string representation: Japchae from Korean Store @ Canteen 20
					---------- test_food_str_display passed successfully ----------

TC_FM_003	Discount price calculation	High	def setUpTestData(cls):
			super().setUpTestData()
			cls.food = Food.objects.create(
			store=cls.store,
			foodName="Japchae",
			price=5.50,
			type="Non-Halal",
			cuisine="Korean",
			description="Delicious Japchae",
			rating=4.5,
			isDiscounted=True,
			discountRate=10,
			)
				def save(self, *args, **kwargs):
				if self.isDiscounted and self.discountRate:
				self.discountedPrice = self.price * ((100 - self.discountRate) / 100)
				
				def test_discountedPrice_calculation(self):
				expected_discountedPrice = self.food.price * ((100 - self.food.discountRate) / 100)
				self.food.refresh_from_db() #
					Discounted price calculated correctly: 4.95
					---------- 	
					test_discountedPrice_calculation passed successfully
					---------- 	

			Set isDiscounted = True and give the discountRate = 10%	Refresh the instance from the database to ensure the latest data is loaded after save self.assertEqual(self.food.discountedPrice, expected_discountedPrice) print(f"Discounted price calculated correctly: {self.food.discountedPrice}") Test the discount formula, expected output = 90%*5.50 = 4.95	

2.2.7 Review Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output
TC_RM_001	Create review object	High	cls.user = User.objects.create_user(username='testuser', password="password") cls.food = Food.objects.create(foodName="Ddukbokki", store=cls.store) cls.review = Review.objects.create(user=cls.user, food=cls.food, rating=4.5, description="Great taste!",) Input particulars needed to create a	def test_review_creation(self): self.assertEqual(self.review.user, self.user) self.assertEqual(self.review.food, self.food) self.assertEqual(self.review.rating, 4.5) self.assertEqual(self.review.description, "Great taste!") print("Review instance created successfully with user, food, rating, and description.") New review object should be created with the same	Review instance created successfully with user, food, rating, and description. ---------- test_review_creation passed successfully ----------

			review object. Only consumers have access to reviews, so the user = “testuser” instead of “teststoreowner” or both.	information as the input.	

TC_RM_002	Review String display	High	firstName = “Jane”
			lastName = “Tan”
			foodName = “Ddukbokki”
			storeName = “Korean Store”
				def __str__(self):
				return self.user.consumer.firstName + " " + self.user.consumer.lastName + " , " + self.food.foodName + " from " + self.food.store.storeName
				
				def test_review_str_display(self):
				expected_output = "Jane Tan , Ddukbokki from Korean Store"
				self.assertEqual(str(self.review), expected_output)
				print(f"Review string representation: {str(self.review)}")
					Review string representation: Jane Tan , Ddukbokki from Korean Store
					---------- 	
					test_review_str_display passed successfully ----------

TC_RM_003	Valid rating creation	High	review = Review.objects.create(user=self.user, food=self.food, rating=5.0, description="Good") Review should be created with a rating of 5.	def test_valid_rating_creation(self): review = Review.objects.create(user=self.user, food=self.food, rating=5.0, description="Good") self.assertEqual(review.rating, 5.0) print("Review instance created with a valid rating of 5.0.") Rating = 5.0	Review instance created with a valid rating of 5.0. ---------- test_valid_rating_creation passed successfully ----------

TC_RM_004	Multiple reviews from same user	High	cls.user = User.objects.create_user(username='testuser', password="password") cls.review = Review.objects.create(user=cls.user, food=cls.food, rating=4.5, description="Great taste!",) Review.objects.create(user=self.user, food=self.food, rating=4.5) Review.objects.create(user=self.user, food=self.food, rating=3.0) 3 review objects created by the same user.	def test_same_name_reviews(self): Review.objects.create(user=self.user, food=self.food, rating=4.5) Review.objects.create(user=self.user, food=self.food, rating=3.0) self.assertEqual(self.food.reviews.count(), 3) print("Review count: ",self.food.reviews.count()) print("Multiple reviews with the same food and user names were created successfully.") The number of reviews by the same user should be 3.	Review count: 3 Multiple reviews with the same food and user names were created successfully. ---------- test_same_name_reviews passed successfully ----------

2.2.8 Flagged Review Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_FRM_001	Create flagged review object	High	cls.flaggingUser = User.objects.create_user(username='john', password="password") cls.food = Food.objects.create(foodName="Bulgogi", store=cls.store) cls.review = Review.objects.create(user=cls.user, food=cls.food, rating=2.0, description="Tasteless...",) cls.flaggedReview = FlaggedReview.objects.create(review=cls.review, flaggedBy=cls.flaggingUser, reason="Inappropriate Review",) Username “john” flags a review, with reason “Inappropriate Review”.	def test_flaggedReview_creation(self): self.assertIsInstance(self.flaggedReview, FlaggedReview) self.assertEqual(self.flaggedReview.review, self.review) self.assertEqual(self.flaggedReview.flaggedBy, self.flaggingUser) self.assertEqual(self.flaggedReview.reason, "Inappropriate Review") print("FlaggedReview instance created successfully with review, flaggedBy, and reason.") New flagged review object should be created with the same information as the input.	FlaggedReview instance created successfully with review, flaggedBy, and reason. ---------- test_flaggedReview_creation passed successfully ----------
TC_FRM_002	Default isProcessed status	Medium	def test_default_isProcessed(self): self.assertFalse(self.flaggedReview.isProcessed) print("isProcessed default value is False.") The default status of isProcessed, which checks whether a flagged review is processed by an admin, is false.	class FlaggedReview(models.Model): review = models.ForeignKey(Review, on_delete=models.CASCADE) flaggedBy = models.ForeignKey(User, on_delete=models.CASCADE) reason = models.TextField(max_length=500, null=True) isProcessed = models.BooleanField(default=False) dateFlagged = models.DateTimeField(auto_now_add=True)	isProcessed default value is False. ---------- test_default_isProcessed passed successfully ----------

TC_FRM_003	Flagged review String display	High	self.review= Jane Tan , Bulgogi from Korean Store
			
			self.flaggingUser = john
				def __str__(self):
				return f"Review {self.review} flagged by {self.flaggedBy}"
				
				def test_flaggedReview_str_display(self):
				expected_output = f"Review {self.review} flagged by {self.flaggingUser}"
				
				self.assertEqual(str(self.flaggedReview), expected_output)
				print(f"FlaggedReview string representation: {str(self.flaggedReview)}")
				
				Expected output= “Review Jane Tan , Bulgogi from Korean Store flagged by john”
					FlaggedReview string representation: Review Jane Tan , Bulgogi from Korean Store flagged by john
					--- 	
					test_flaggedReview_str_display passed successfully
					--- 	

2.2.9 Notification Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_NM_001	Create notification object	High	cls.user = User.objects.create_user(username='testuser', password="password") def setUpTestData(cls): super().setUpTestData() cls.notification = Notification.objects.create(title="Hello", description="Nice to see you", user=cls.user,) Input needed to create a notification object by admin. Username of “testuser” should receive this notification.	def test_notification_creation(self): self.assertIsInstance(self.notification, Notification) self.assertEqual(self.notification.title, "Hello") self.assertEqual(self.notification.description, "Nice to see you") self.assertEqual(self.notification.user, self.user) print("Notification instance created successfully with title, description, and user.") The notification object should be created with the same information as input.	Notification instance created successfully with title, description, and user. ---------- test_notification_creation passed successfully ----------

TC_NM_002	Default viewed status	Medium	def test_notification_defaultViewed(self):
			self.assertFalse(self.notification.viewed) # viewed should default to False
			print("Viewed default value is False.")
			
			The default status of viewed, to check whether the notification is viewed by the user, is false.
				class Notification(models.Model):
				title = models.TextField(max_length = 100, null = True)
				description = models.TextField(max_length = 1000, null = True)
				viewed = models.BooleanField(default = False, null = True)
				user = models.ForeignKey(User, null=True, on_delete=models.CASCADE)
				dateCreated = models.DateTimeField(auto_now_add=True, null = True)
					Viewed default value is False.
					---------- 	
					test_notification_defaultViewed passed successfully
					---------- 	

TC_NM_003	Notification String display	High	self.user.username= “testuser”
			self.title = “Hello”
				def __str__(self):
				return f"Notification for {self.user.username}. Title: {self.title}"
				
				def test_notification_str_display(self):
				expected_output = "Notification for testuser. Title: Hello"
				self.assertEqual(str(self.notification), expected_output)
				print(f"Notification string representation: {str(self.notification)}")
					Notification string representation: Notification for testuser. Title: Hello
					---------- 	
					test_notification_str_display passed successfully ----------

2.2.10 Banned User Model Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output
TC_BM_001	Create banned user object	High	def setUpTestData(cls): super().setUpTestData() cls.banUser = User.objects.create_user(username='test', password="password") cls.bannedUser = BannedUser.objects.create(user=cls.banUser,) Input needed to create a banned user object, with username “test”.	def test_bannedUser_creation(self): self.assertIsInstance(self.bannedUser, BannedUser) self.assertEqual(self.bannedUser.user, self.banUser) print("BannedUser instance created successfully with user.") The banned user object should have the same information as the input.	Banned user: test BannedUser instance created successfully with user. ---------- test_bannedUser_creation passed successfully ----------

TC_BM_002	Default isBanned status	Medium	def test_bannedUser_isBanned(self): self.assertTrue(self.bannedUser.isBanned) print("isBanned default value is True.") The default status of isBanned, which checks that the user is banned, is true.	class BannedUser(models.Model): user = models.OneToOneField(User, on_delete=models.CASCADE) bannedOn = models.DateTimeField(auto_now_add=True) isBanned = models.BooleanField(default=True)	isBanned default value is True. ---------- test_bannedUser_isBanned passed successfully ----------

TC_BM_003	Banned user String display	High	self.user.username= “test”	def __str__(self):
				return f"Banned user: {self.user.username}."
				
				def test_bannedUser_str_display(self):
				expected_output = "Banned user: test."
				
				self.assertEqual(str(self.bannedUser), expected_output)
				print(f"BannedUser string representation: {str(self.bannedUser)}")
					BannedUser string representation: Banned user: test.
					---------- 	
					test_bannedUser_str_display passed successfully ----------

2.3 Views Test:

2.3.1 Create Review Views Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_CRV_001	Create review get request	High	cls.create_review_url = reverse('createReview', args = [cls.consumer.id, cls.food.id]) cls.user = User.objects.create_user(username = "testuser", password="password") cls.canteen = Canteen.objects.create(location = Location.objects.create(canteenName = "Test Canteen")) cls.store = Store.objects.create(storeName = "Test Store", canteen = cls.canteen) cls.food = Food.objects.create(foodName = "Pizza", store = cls.store, rating = 0) Consumer has to be logged in and select a food to review to get the review form for that particular food.	url = reverse("createReview", args=[self.consumer.id, self.food.id]) response = self.client.get(url) print(f"\nResponse URL: {url}") # Check that the form is displayed self.assertEqual(response.status_code, 200) self.assertTemplateUsed(response, "NTUFoodieApp/reviewform.html") # Print the response URL and template used # Confirm no review was created self.assertEqual(Review.objects.count(), 0) Expected: form to be redirected with url of correct consumer.id and food.id, with status code 200, using the correct html template. The review count should be 0 as it's just displaying the form, no review submitted yet.	Response URL: /consumer/create_review/1/1/ Response Status Code: 200 Templates used: NTUFoodieApp/reviewform.html Number of Review created: 0 ---------- test_createReview_get_request passed successfully ----------

TC_CRV_002	Submit review successful	High	self.client.login(username="testuser", password="password")
			form_data = {
			'rating': 4,
			'description': "Delicious food!",
			}
			
			Consumer is logged in and inputs
				response = self.client.post(self.create_review_url, data=form_data)
				
				# Check for redirection status code after successful review creation
				
				self.assertEqual(response.status_code, 302)
					isBanned default value is True.
					---------- 	
					test_bannedUser_isBanned passed successfully ----------

			into the review form with rating = 4 and description “Delicious food”.	# Expected URL with trailing slash
				food_detail_url = reverse('foodDetail', args=[self.food.id])
				
				Review should be posted to the create_review_url with the data inputs. Then, the user will be redirected to the food_detail_url, which is the same as the create_review_url, with status code 302.
				
				# Check if the review was created
				
				self.assertEqual(Review.objects.count(), 1)
				
				# Check if the rating was saved correctly
				review = Review.objects.first()
				self.assertEqual(review.rating, 4.0)
				
				# Verify if the food rating is updated
				self.food.refresh_from_db()
				self.assertEqual(self.food.rating, 4.0)
				
				# Check if success message was sent
				messages = list(get_messages(response.wsgi_request))
				self.assertTrue(any("Review Created!" in str(message) for message in messages))
				print(messages)
					Response URL: /consumer/food/1/
					Response Status Code: 302
					Food Detail URL: /consumer/food/1/
					Number of Review created: 1
					Rating saved: 4.0
					Updated Food Rating: 4.0
					[Message(level=25, message='Review Created!')]
					---------- 	
					test_createReview_successful passed successfully
					---------- 	

				Review should be successfully created, so count =1. The review should also capture the rating of 4.0, and a success message is displayed.	

2.3.2 Store Owner Add Listing Views Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output
TC_SOALV_001	Store owner access	High	storeowner_group, created = Group.objects.get_or_create(name="storeowner") cls.user = User.objects.create_user(username = "teststoreowner", password="password") Store owner has to be logged in to see the listing page.	cls.url = reverse("soHome") response = self.client.get(self.url) print(f"Response URL: {self.url}") # Check that the form is displayed self.assertEqual(response.status_code, 200) self.assertTemplateUsed(response, "NTUFoodieApp/sohome.html") Expected: Listings page should be displayed in the store owner's home page, with url "soHome", with status code 200, using the correct html template.	Response URL: /storeowner/home/ Response Status Code: 200 Templates used: NTUFoodieApp/sohome.html ---------- test_storeOwner_access passed successfully ----------

TC_SOALV_002	Submit food listing successful	High	self.client.login(username="teststoreowner", password="password")
			
			cls.store = Store.objects.create(storeName = "Test Store")
			cls.food = Food.objects.create(store = cls.store)
			
			cls.form_data = {
			'foodName': "Pork Chop Fried Rice",
			'price': 5.50,
			'type': "Non-Halal",
			'cuisine': "Western",
			'description': "Delicious food!",
			}
			
			Store owner is logged in and inputs into the listing form with foodName, price, type, cuisine and description.
				cls.url = reverse("soHome")
				
				response = self.client.post(self.url, data=self.form_data)
				
				self.assertEqual(response.status_code, 302)
				
				self.assertRedirects(response, self.url)
				
				# Check if success message was sent
				messages = list(get_messages(response.wsgi_request))
				self.assertTrue(any("Listing Added Successfully!" in str(message) for message in messages))
				print(messages)
				
				Expected: Listing should be posted to the url with the data inputs. Then, the store owner will be redirected back to the url with status code 302, and a success message is displayed.
					Response URL: /storeowner/home/
					Response Status Code: 302
					URL Redirected
					[Message(level=25, message='Listing Added Successfully!')]
					---------- 	
					test_foodListing_creation passed successfully ----------

TC_SOALV_003	Invalid price input	High	self.client.login(username="teststoreowner", password="password")
			form_data_invalidPrice = self.form_data.copy()
			form_data_invalidPrice.update({
			
			'price': -10,
			})
				response = self.client.post(self.url, data=form_data_invalidPrice)
				
				self.assertEqual(response.status_code, 200)
				
				# Check if error message was sent
				messages = list(get_messages(response.wsgi_reques
					Response URL: /storeowner/home/
					Response Status Code: 200
					[Message(level=40, message='Error in price: * Price cannot be negative.')]
					---------- 	

			Store owner is logged in and inputs a negative price point to the listing.	t)) self.assertTrue(any("Error in price: * Price cannot be negative." in str(message) for message in messages)) print(messages) Expected: Store owner will be redirected back to url with status code 200, as the page successfully redirects back. However, listing is not added successfully, as price cannot be negative and an error message is displayed.	test_invalidPrice_input passed successfully ----------

2.3.3 Search Food Views Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output

TC_SFV_001	Search food by food name	High	self.client.login(username="testuser", password = "password")
			
			cls.food = Food.objects.create(foodName = "Pizza", store = cls.store, rating = 0)
			
			ls.food2 = Food.objects.create(foodName = "Burger", store = cls.store, rating = 5)
			
			Two food objects are created with foodName “Pizza” and “Burger”.
			
			Consumer is logged in and searches for “Pizza”.
				response = self.client.get(reverse("searchFood"), {'searchKey': 'Pizza'})
				
				#check that food page is displayed
				self.assertEqual(response.status_code, 200)
				
				self.assertTemplateUsed(response, "NTUFoodieApp/foodresult.html")
				
				self.assertContains(response, "Pizza")
				self.assertNotContains(response, "Burger")
				
				Expected: Search results should be displayed with only “Pizza”, with status code 200, using the correct html template.
					Pizza
					Response URL: /consumer/searchFood/
					Response Status Code: 200
					Templates used: NTUFoodieApp/foodresult.html
					---------- 	
					test_searchFood_by_foodName passed successfully
					---------- 	

TC_SFV_002	Search food by canteen name	High	self.client.login(username="testuser", password = "password") cls.canteen = Canteen.objects.create(location = Location.objects.create(canteenName = "Test Canteen")) Consumer is logged in and searches for “Test Canteen”.	response = self.client.get(reverse("searchFood"), {'searchKey': 'Test Canteen'}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertTemplateUsed(response, "NTUFoodieApp/foodresult.html") self.assertContains(response, "Pizza") self.assertContains(response, "Burger")	HTML display: <tbody> <td>Pizza</td> <td>Burger</td> <tbody> Test Canteen Response URL: /consumer/searchFood/ Response Status Code: 200 Templates used: NTUFoodieApp/foodresult.ht

				Expected: Search results should be displayed with both food “Pizza” and “Burger” as they are both part of “Test Canteen”, with status code 200, using the correct html template.	ml ---------- test_searchFood_by_canteenName passed successfully ----------
TC_SFV_003	Search for food that doesn’t exist.	High	self.client.login(username="testuser", password = "password") 'searchKey': 'Sushi' Consumer is logged in and searches for “Sushi”.	response = self.client.get(reverse("searchFood"), {'searchKey': 'Sushi'}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertTemplateUsed(response, "NTUFoodieApp/foodresult.html") self.assertNotContains(response, "Pizza") self.assertNotContains(response, "Burger") Expected: Search results will be empty as there is no food object named “Sushi”, status code is 200 as search results page will still be displayed, using the correct html template.	HTML display: <tbody> </tbody> tbody is empty Sushi Response Status Code: 200 Templates used: NTUFoodieApp/foodresult.html ---------- test_searchFood_noResults passed successfully ----------

2.3.4 Filter Food Views Test:

Test Case ID	Description	Priority	Input	Expected Output	Actual Output
TC_FFV_001	Filter food by minimum price	High	cls.food1 = Food.objects.create(foodName = "Sushi", price = 5.50, type = "Non-Halal", cuisine = "Japanese", isDiscounted = False, discountRate = 0) cls.food2 = Food.objects.create(foodName = "Pasta", price = 3, type = "Halal", cuisine = "Italian", isDiscounted = False, discountRate = 0) cls.food3 = Food.objects.create(foodName = "Fries", price = 7, type = "Halal", cuisine = "Western", isDiscounted = True, discountRate = 10) cls.food4 = Food.objects.create(foodName = "Chicken Burger", price = 10, type = "Halal", cuisine = "Western", isDiscounted = True, discountRate = 20) self.client.login(username="testuser", password = "password") Four food objects created to simulate the filtering. Consumer is logged in and filters food by a minimum price of $4.	response = self.client.get(reverse('filterFood'), {'min_price': 4}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertEqual(len(response.context['filteredFood']), 3) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html") Expected: All food with price higher than $4 are displayed in the filter results, with status code 200, using the correct html template.	HTML display: <tbody> <td>Sushi</td> <td>Fries</td> <td>Chicken Burger</td> <tbody> Response Status Code: 200 Number of Filtered Food: 3 Templates used: NTUFoodieApp/foodfilter.html ---------- test_filterBy_minPrice passed successfully ----------

TC_FFV_002	Filter food by maximum price	High	self.client.login(username="testuser", password = "password") 'max_price': 10 Consumer is logged in and filters food by a maximum price of $10.	response = self.client.get(reverse('filterFood'), {'max_price': 10}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertEqual(len(response.context['filteredFood']), 4) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html") Expected: All food with price lower than $10 are displayed in the filter results, with status code 200, using the correct html template.	HTML display: <tbody> <td>Sushi</td> <td>Pasta</td> <td>Fries</td> <td>Chicken Burger</td> <tbody> Response Status Code: 200 Number of Filtered Food: 4 Templates used: NTUFoodieApp/foodfilter.html ---------- test_filterBy_maxPrice passed successfully ----------
TC_FFV_003	Filter food by food type	High	self.client.login(username="testuser", password = "password") 'type': 'Non-Halal' Consumer is logged in and filters food by food type “Non-Halal”.	response = self.client.get(reverse('filterFood'), {'type': 'Non-Halal'}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertEqual(len(response.context['filteredFood']), 1) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html")	HTML display: <tbody> <td>Sushi</td> <tbody> Response Status Code: 200 Number of Filtered Food: 1 Templates used: NTUFoodieApp/foodfilter.html ----------

				Expected: All non-halal food are displayed in the filter results, with status code 200, using the correct html template.	test_filterBy_foodType passed successfully ----------
TC_FFV_004	Filter food by cuisine	High	self.client.login(username="testuser", password = "password") 'cuisine': 'Western' Consumer is logged in and filters food by cuisine “Western”.	response = self.client.get(reverse('filterFood'), {'cuisine': 'Western'}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertEqual(len(response.context['filteredFood']), 2) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html") Expected: All Western food are displayed in the filter results, with status code 200, using the correct html template.	HTML display: <tbody> <td>Fries</td> <td>Chicken Burger</td> <tbody> Response Status Code: 200 Number of Filtered Food: 2 Templates used: NTUFoodieApp/foodfilter.html ---------- test_filterBy_cuisine passed successfully ----------
TC_FFV_005	Filter by discounted food	High	self.client.login(username="testuser", password = "password") ‘discounted': 'true' Consumer is logged in and filters food by discounted food.	response = self.client.get(reverse('filterFood'), {'discounted': 'true'}) #check that food page is displayed self.assertEqual(response.status_code, 200)	HTML display: <tbody> <td>Fries</td> <td>Chicken Burger</td> <tbody>

				self.assertEqual(len(response.context['filteredFood']), 2) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html") Expected: All discounted food are displayed in the filter results, with status code 200, using the correct html template.	Response Status Code: 200 Number of Filtered Food: 2 Templates used: NTUFoodieApp/foodfilter.html ---------- test_filterBy_discountedFood passed successfully ----------
TC_FFV_006	Filter food by combined criteria	High	self.client.login(username="testuser", password = "password") 'min_price': 3, 'max_price': 10, 'type': 'Halal', 'discounted': "true" Consumer is logged in and filters food by combined criteria.	response = self.client.get(reverse('filterFood'), {'min_price': 3, 'max_price': 10, 'type': 'Halal', 'discounted': "true"}) #check that food page is displayed self.assertEqual(response.status_code, 200) self.assertEqual(len(response.context['filteredFood']), 2) self.assertTemplateUsed(response, "NTUFoodieApp/foodfilter.html") Expected: Food that meets this criteria is displayed in the filter results, with status code 200, using the correct html template.	HTML display: <tbody> <td>Fries</td> <td>Chicken Burger</td> <tbody> Response Status Code: 200 Number of Filtered Food: 2 Templates used: NTUFoodieApp/foodfilter.html ---------- test_filterBy_combinedCriteria passed successfully ----------