

Test Document

for

Quick Crave

Version 1.0

Prepared by: Cache Me If You Can (Team 9)

Dnyanesh Hemendra Shinde	230390	dnyanesh23@iitk.ac.in
Nikhilesh Joshi	230700	nikhileshj23@iitk.ac.in
Kamatam Akshay Reddy	230517	areddy23@iitk.ac.in
Arihant Kumar	230189	arihantkmr23@iitk.ac.in
Kshitij Gupta	230581	kshitijg23@iitk.ac.in
Parnika Mittal	230736	mparnika23@iitk.ac.in
Rohit Vijaykumar Somani	230871	rohitvs23@iitk.ac.in
Suvid Goel	231065	gsuvid23@iitk.ac.in
Yashaswi Raj	231191	yashaswir23@iitk.ac.in
Yatharth Dangi	231197	yatharth23@iitk.ac.in

Course: CS253

Mentor TA: Nij Bharatkumar Padariya

Submission Date: April 05, 2025

Contents

1	Introduction	5
1.1	Test Strategy	5
1.2	Testing Timeline	5
1.3	Testers	5
1.4	Coverage Criteria	5
1.5	Tools used for testing	5
2	Unit Testing	6
2.1	Authentication	6
2.1.1	Registering a Customer	6
2.1.2	Email Verification by OTP	6
2.1.3	Registering a Canteen	6
2.1.4	Logging in	6
2.2	Customer Side Functionalities	6
2.2.1	Search Canteen	7
2.2.2	Search Dish	7
2.2.3	Fetch Reservations	8
2.2.4	Add Dish To Cart	9
2.2.5	Remove Dish From Cart	10
2.2.6	View Order History	11
2.2.7	View Favourites	12
2.2.8	Add Favourite	12
2.2.9	View Wallet	13
2.2.10	Update Wallet	14
2.2.11	Add Reservation	14
2.2.12	View Profile	15
2.2.13	Fetch Top 5 Dishes	16
2.2.14	Reset Password	16
2.2.15	Update Profile Photo	17
2.2.16	Update Profile	17
2.2.17	Change Password	18
2.2.18	Fetch Dishes	19
2.2.19	Fetch Canteens	19
2.2.20	Fetch Menu	20
2.2.21	View Cart	21
2.2.22	Requesting Order	21
2.2.23	Placing Order	22
2.3	Canteen Side Functionalities	23
2.3.1	Active Orders / Pending Order Requests	23
2.3.2	Confirmed / Pending Reservation Requests	23
2.3.3	Active Coupons / Discounts	24
2.3.4	Trending Picks	24
2.3.5	Order Queue	25
2.3.6	Accept Order	26
2.3.7	Reject Order	26
2.3.8	Mark Order As Ready	26
2.3.9	Reservation Queue	27
2.3.10	Accept Reservation	28
2.3.11	Reject Reservation	28

2.3.12 See Active Discounts	28
2.3.13 Create Discounts	29
2.3.14 Delete Discount	30
2.3.15 See Active Coupons	30
2.3.16 Create Coupon	31
2.3.17 Delete Coupon	32
2.3.18 Get Profile Details	32
2.3.19 Edit Profile	33
2.3.20 Edit Password	33
2.3.21 Show Menu Items	34
2.3.22 Edit Dish Details	35
2.3.23 Add New Dish	36
2.3.24 Get Statistics	36
2.3.25 Get Graph Details	37
3 Integration Testing	39
3.1 Authentication	39
3.1.1 Registering a customer	39
3.1.2 Customer	41
3.1.3 Registering a canteen	42
3.1.4 Logging in as customer	43
3.1.5 Logging in as canteen manager	44
3.2 Customer Side	45
3.2.1 Customer Home	45
3.2.2 Customer Profile	45
3.2.3 Cart	51
3.2.4 Sidebar	52
3.2.5 Reservation	54
3.2.6 Food Delivery and Dineout	55
3.2.7 My Favourites	57
3.3 Canteen Side	60
3.3.1 Canteen Home Page	60
3.3.2 Order Queue Page	60
3.3.3 Reservation Queue Page	61
3.3.4 Discounts and Coupons Page	62
3.3.5 Menu Management Page	63
3.3.6 Profile Page	63
3.3.7 Statistics Page	65
4 System Testing	67
4.1 Load testing	67
4.1.1 customer/customer-search-canteen	67
4.1.2 customer/customer-search-dish	68
4.1.3 /customer/customer-fetch-reservations	69
4.1.4 /customer/customer-add-dish	70
4.1.5 /customer/customer-remove-dish	71
4.1.6 /customer/customer-view-order-history	72
4.1.7 /customer/customer-view-favourites	73
4.1.8 /customer/customer-wallet	74
4.1.9 /customer/customer-view-profile	75
4.1.10 /customer/customer-fetch-top5	76
4.1.11 /customer/customer-fetch-dishes	77

4.1.12 /customer/customer-fetch-menu	78
--	----

1 Introduction

1.1 Test Strategy

All components of the system were manually tested to verify correctness and functionality. Each feature was individually executed and its output was validated against expected results to ensure reliable performance.

1.2 Testing Timeline

1.3 Testers

The testing process was carried out internally by our development team. However, to maintain objectivity and reduce potential bias, we ensured that the developer who implemented a specific feature was not the one responsible for testing that particular functionality. This approach helped us maintain a level of separation between development and testing, thereby enhancing the reliability and credibility of our test results.

1.4 Coverage Criteria

We used branch coverage as our test coverage criterion to ensure that all possible branches in the code were executed at least once during testing. This helps in identifying untested paths and improves the overall reliability of the system.

1.5 Tools used for testing

Postman was used to extensively test and validate the backend APIs.

2 Unit Testing

2.1 Authentication

2.1.1 Registering a Customer

API Endpoint: /auth/signup

Test Owner: Nikhilesh Joshi

Date: 29/03/2025

Test Description: This test checks if the API correctly accepts the details of the customer and proceeds to verification.

Test Result: The customer is able to proceed to OTP verification page after providing all his details. If his username, phone number or email has already been used earlier, an appropriate message is shown. An OTP is sent to the given email and the user is redirected to the OTP Verification Page.

2.1.2 Email Verification by OTP

API Endpoint: /auth/otp-verification

Test Owner: Nikhilesh Joshi

Date: 30/03/2025

Test Description: This test checks if the API correctly signs up a customer if the OTP entered is correct.

Test Result: The customer is able to successfully signup and his details are entered in the database if the OTP entered is correct. However, after 3 incorrect attempts, the user is redirected to sign-up page.

2.1.3 Registering a Canteen

API Endpoint: auth/signup

Test Owner: Kshitij Gupta

Date: 30/3/2025

Test Description: This test checks whether canteen is registered

Test Result: This API works as expected

2.1.4 Logging in

API Endpoint: /auth/login

Test Owner: Yatharth Dangi

Date: 29/03/2025

Test Description: This test checks if the API correctly if the user is able to Login with his Username and password.

Test Result: Allows the user to login and then redirects him to the respective home page. If incorrect password is entered, then an appropriate message is shown.

2.2 Customer Side Functionalities

Accessible by the User

NOTE: In unit testing, the canteen_id is sent in the body of a post request, but in the final web app, it will be sent via an http-only cookie and not inside the body of the request

2.2.1 Search Canteen

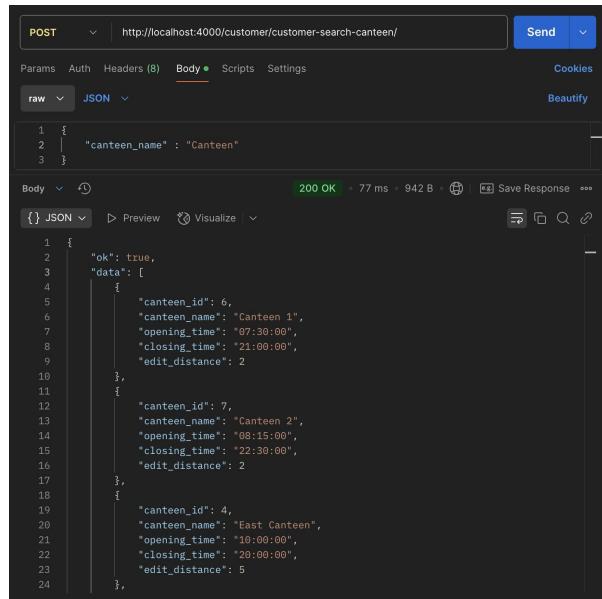
API Endpoint: /customer-search-canteen

Test Owner: Arihant Kumar

Date: 29/03/2025

Test Description: This test checks if the API correctly returns a list of canteens based on the customer's search query

Test Result: A list of 5 canteens are returned whose names are the closest to the search query made by the customer.



```
POST http://localhost:4000/customer/customer-search-canteen/ Send
Params Auth Headers (8) Body Scripts Settings
raw JSON Beautify
1 {
2   "canteen_name": "Canteen"
3 }
Body 200 OK 77 ms 942 B Save Response ...
{} JSON Preview Visualize
1 {
2   "ok": true,
3   "data": [
4     {
5       "canteen_id": 6,
6       "canteen_name": "Canteen 1",
7       "opening_time": "07:30:00",
8       "closing_time": "21:00:00",
9       "edit_distance": 2
10    },
11    {
12      "canteen_id": 7,
13      "canteen_name": "Canteen 2",
14      "opening_time": "08:15:00",
15      "closing_time": "22:30:00",
16      "edit_distance": 2
17    },
18    {
19      "canteen_id": 4,
20      "canteen_name": "East Canteen",
21      "opening_time": "10:00:00",
22      "closing_time": "20:00:00",
23      "edit_distance": 5
24    }
]
```

Figure 1: Searching for Canteens

2.2.2 Search Dish

API Endpoint: /customer-search-dish

Test Owner: Dnyanesh Hemendra Shinde

Date: 29/03/2025

Test Description: This test checks if the API correctly returns a list of dishes matching the customer's search input

Test Result: A list of 5 dishes are returned whose names are the closest to the search query made by the customer.

The screenshot shows a POST request to `http://localhost:4000/customer/customer-search-dish/`. The Body tab contains the JSON payload:

```

1 {
2   "dishNameToSearch": "Paneer"
3 }

```

The response status is 200 OK, with a response time of 9 ms and a size of 702 B. The response body is:

```

1 {
2   "success": true,
3   "data": [
4     {
5       "dish_name": "Paneer Tikka",
6       "img_url": "https://example.com/paneertikka.jpg",
7       "rating": 4.5,
8       "is_veg": true,
9       "price": 100,
10      "canteen_name": "North Canteen",
11      "edit_distance": 6
12    },
13    {
14      "dish_name": "Paneer Butter Masala",
15      "img_url": "https://example.com/paneer.jpg",
16      "rating": 4.8,
17      "is_veg": true,
18      "price": 120,
19      "canteen_name": "North Canteen",
20      "edit_distance": 14
21    }
22  ]
23 }

```

Figure 2: Searching by Dish Name

2.2.3 Fetch Reservations

API Endpoint: /customer-fetch-reservations

Test Owner: Parnika Mittal

Date: 29/03/2025

Test Description: This test checks if the API returns all active and past reservation details of the logged-in customer

Test Result: Returns the details of the all the reservations that have been made my the customer, active as well as present.

The screenshot shows a POST request to `http://localhost:4000/customer/customer-fetch-reservations/`. The Body tab contains the JSON payload:

```

1 {
2   "customer_id": 25
3 }

```

The response status is 200 OK, with a response time of 14 ms and a size of 465 B. The response body is:

```

1 {
2   "reservations": [
3     {
4       "reservation_id": 13,
5       "status": 0,
6       "name": "North Canteen",
7       "time": "2025-04-05T14:30:00.000Z",
8       "num_people": 4
9     }
10   ]
11 }

```

Figure 3: List of Reservavtions

```

POST http://localhost:4000/customer/customer-fetch-reservations/
Send

Params Auth Headers (8) Body Scripts Settings Cookies
raw JSON

1 {
2   "customer_id": 25
3 }

Body JSON Preview Visualize
200 OK 10 ms 360 B Save Response

1
2
3

```

Figure 4: No Reservations

2.2.4 Add Dish To Cart

API Endpoint: /customer-add-dish

Test Owner: Kamatam Akshay Reddy

Date: 29/03/2025

Test Description: This test checks if the API correctly adds a selected dish to the customer's cart

Test Result: Adds the dish selected to the cart of the customer and makes appropriate changes in the database as well. The current price of the cart is also updated. It creates a new cart if no previous cart exists, otherwise adds to the existing cart only if the current dish being added is of the same canteen as the dishes in the cart. An appropriate message is shown.

```

POST http://localhost:4000/customer/customer-add-dish/
Send

Params Auth Headers (8) Body Scripts Settings Cookies
raw JSON

1 {
2   "customer_id": 25,
3   "dish_id": 2
4 }

Body JSON Preview Visualize
201 Created 29 ms 453 B Save Response

1
2
3
4
5
6
7
8
9
10
11

```

Figure 5: Adding Dish to a New Cart

```

POST http://localhost:4000/customer/customer-add-dish/
Params Auth Headers (8) Body Scripts Settings Cookies
raw JSON
1 {
2   "customer_id": 25,
3   "dish_id": 3
4 }

Body 200 OK 27 ms 451 B Save Response
[] JSON > Preview Visualize
1 {
2   "message": "Dish added to existing cart",
3   "cart": {
4     "dish_map": {
5       "2": 1,
6       "3": 1
7     },
8     "final_amount": 270
9   },
10  "success": true
11 }

```

Figure 6: Adding Dish to an Existing Cart

```

POST http://localhost:4000/customer/customer-add-dish/
Params Auth Headers (8) Body Scripts Settings Cookies
raw JSON
2 {
3   "customer_id": 25,
4   "dish_id": 2
5 }

Body 401 Unauthorized 52 ms 469 B Save Response
[] JSON > Preview Visualize
1 {
2   "message": "The current canteen of the customer is different from the canteen of the dish selected.",
3   "success": false
4 }

```

Figure 7: When dishes are from different Canteens

2.2.5 Remove Dish From Cart

API Endpoint: /customer-remove-dish

Test Owner: Kshitij Gupta

Date: 29/03/2025

Test Description: This test checks if the API successfully removes a specific dish from the customer's cart

Test Result: Removes the dish selected from the cart of the customer and makes appropriate changes in the database as well. If the cart of the user is now empty, then appropriate changes are made to the database.

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/customer/customer-remove-dish/`. The request body is JSON, containing:

```

1  {
2   |   "customer_id": 25,
3   |   "dish_id": 3
4 }

```

The response status is 200 OK, with a response time of 26 ms and a size of 439 B. The response body is:

```

1  {
2   |   "message": "Dish removed from cart",
3   |   "cart": {
4   |   |   "dish_map": {
5   |   |   |   "2": 1
6   |   |   },
7   |   |   "final_amount": 120
8   |   },
9   |   "success": true
10 }

```

Figure 8: Removing Dish from Cart

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/customer/customer-remove-dish/`. The request body is JSON, containing:

```

1  {
2   |   "customer_id": 25,
3   |   "dish_id": 4
4 }

```

The response status is 404 Not Found, with a response time of 43 ms and a size of 400 B. The response body is:

```

1  {
2   |   "message": "Dish not found in cart",
3   |   "success": false
4 }

```

Figure 9: When Dish does not exist

2.2.6 View Order History

API Endpoint: /customer-view-order-history

Test Owner: Arihant Kumar

Date: 01/04/2025

Test Description: This test checks if the API correctly returns the customer's complete order history

Test Result: Returns a list of orders made by the customer. The customer can filter the results based on date as well as status of the order.

```

POST http://localhost:4000/customer/customer-view-order-history
{
  "customer_id": 25
}
200 OK
{
  "message": "Got the results.",
  "data": [
    {
      "order_id": 2,
      "customer_id": 25,
      "canteen_id": 2,
      "status": 0,
      "payment_id": "",
      "coupon_id": null,
      "bill_amount": 0,
      "discount_amount": 0,
      "final_amount": 0,
      "time": "2025-04-04T11:35:36.964Z",
      "is_delivery": false,
      "dish_map": {}
    },
    {
      "order_id": 8,
      "customer_id": 25,
    }
  ]
}

```

Figure 10: History of the customer

2.2.7 View Favourites

API Endpoint: /customer-view-favourites

Test Owner: Suvid Goel

Date: 02/04/2025

Test Description: This test checks if the API returns a list of the customer's favourite dishes

Test Result: Returns a list of the favorite dishes selected by the customer along with all the details.

```

POST http://localhost:4000/customer/customer-view-favourites/
{
  "customer_id": 25
}
200 OK
{
  "rows": [
    {
      "dish_name": "Paneer Butter Masala",
      "rating": 4.8,
      "is_veg": true,
      "img_url": "https://example.com/paneer.jpg",
      "price": 120
    },
    {
      "dish_name": "Chicken Biryani",
      "rating": 4.5,
      "is_veg": false,
      "img_url": "https://example.com/biryani.jpg",
      "price": 150
    },
    {
      "dish_name": "Pasta Alfredo",
      "rating": 4.2,
      "is_veg": true
    }
  ]
}

```

Figure 11: Viewing favourites

2.2.8 Add Favourite

API Endpoint: /customer-add-favourite

Test Owner: Yatharth Dangi

Date: 03/04/2025

Test Description: This test checks if the API adds a dish to the customer's favourites successfully

Test Result: Adds the dish selected by the customer to the favorites of the customer and makes appropriate changes to the database.

```

POST http://localhost:4000/customer/customer-add-favourite/
{
  "customer_id": 25,
  "dish_id": 5
}
200 OK
{
  "message": "Dish added to favorites successfully",
  "success": true
}

```

Figure 12: Adding a favourite

2.2.9 View Wallet

API Endpoint: /customer-wallet

Test Owner: Rohit Vijaykumar Somani

Date: 04/04/2025

Test Description: This test checks if the API returns the current wallet balance of the customer

Test Result: Returns the current wallet balance of the customer.

```

POST http://localhost:4000/customer/customer-wallet/
{
  "message": "Data Acquired",
  "data": [
    {
      "transaction_id": 8,
      "order_id": 0,
      "customer_id": 25,
      "time": "2025-04-05T16:20:15.338Z",
      "amount": 40
    },
    {
      "transaction_id": 7,
      "order_id": 0,
      "customer_id": 25,
      "time": "2025-04-05T15:59:55.590Z",
      "amount": 1000
    }
  ],
  "balance": {
    "command": "SELECT",
    "rowCount": 1,
    "oid": null,
    "rows": [
      {
        "wallet_balance": 210
      }
    ]
  }
}
200 OK
{
  "message": "Data Acquired",
  "data": [
    {
      "transaction_id": 8,
      "order_id": 0,
      "customer_id": 25,
      "time": "2025-04-05T16:20:15.338Z",
      "amount": 40
    },
    {
      "transaction_id": 7,
      "order_id": 0,
      "customer_id": 25,
      "time": "2025-04-05T15:59:55.590Z",
      "amount": 1000
    }
  ],
  "balance": {
    "command": "SELECT",
    "rowCount": 1,
    "oid": null,
    "rows": [
      {
        "wallet_balance": 210
      }
    ]
  }
}

```

Figure 13: Wallet Balance and Transactions

```

POST http://localhost:4000/customer/customer-wallet/
{
  "customer_id": 25
}
200 OK
{
  "message": "User has no transaction till now",
  "success": true,
  "walletBalance": 0
}

```

Figure 14: Empty Wallet

2.2.10 Update Wallet

API Endpoint: /customer-update-wallet

Test Owner: Yashaswi Raj

Date: 04/03/2025

Test Description: This test checks if the API correctly updates the customer's wallet balance after a transaction

Test Result: After a transaction is made by the customer, the wallet of the customer is updated appropriately and database is also updated.

```

POST http://localhost:4000/customer/customer-update-wallet/
{
  "customer_id": 25,
  "moneyToAdd": 1000
}
200 OK
{
  "message": "Wallet updated successfully!",
  "success": true,
  "wallet_balance": 1000,
  "transaction": {
    "transaction_id": 7,
    "order_id": 0,
    "customer_id": 25,
    "time": "2025-04-05T15:59:55.590Z",
    "amount": 1000
  }
}

```

Figure 15: Updating Wallet

2.2.11 Add Reservation

API Endpoint: /customer-add-reservations

Test Owner: Arihant Kumar

Date: 04/04/2025

Test Description: This test checks if the API allows the customer to successfully place a reservation request

Test Result: Takes the necessary details from the customer and sends the reservation request to the canteen manager.

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/customer/customer-add-reservations/`. The request body is a JSON object:

```

1 {
2   "customer_id": 25,
3   "canteen_id": 2,
4   "num_people": 4,
5   "reservation_time": "2025-04-06T12:00:00.000Z"
6 }

```

The response status is 200 OK, with a response body:

```

1 {
2   "message": "Everything done",
3   "transaction_id": 8,
4   "reservation_id": 14,
5   "success": true
6 }

```

Figure 16: Making a Reservation

2.2.12 View Profile

API Endpoint: /customer-view-profile

Test Owner: Dnyanesh Hemendra Shinde

Date: 03/04/2025

Test Description: This test checks if the API returns the complete profile details of the logged-in customer

Test Result: Returns the details of the customer, along with order and reservation data. The details are successfully displayed on the customer profile page.

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/customer/customer-view-profile/`. The request body is a JSON object:

```

1 {
2   "customer_id": 26
3 }

```

The response status is 200 OK, with a response body:

```

1 {
2   "message": "Profile fetched successfully",
3   "success": true,
4   "profile": [
5     {
6       "name": "NIKHILESH JOSHI",
7       "username": "N3",
8       "email": "nikhileshjoshi20@gmail.com",
9       "phone_number": "9818201609",
10      "wallet_balance": 0,
11      "img_url": "https://res.cloudinary.com/dg08t3wkg/image/upload/v1743846597/vnq0d4aoxbxwnekmvzfd.jpg",
12      "address": "C-124, Hall 3, IIT Kanpur",
13      "total_orders": 14,
14      "food_delivery": 0,
15      "dineout": 14,
16      "total_reservations": 0
17    }
18  ]
19 }

```

Figure 17: Viewing Profile

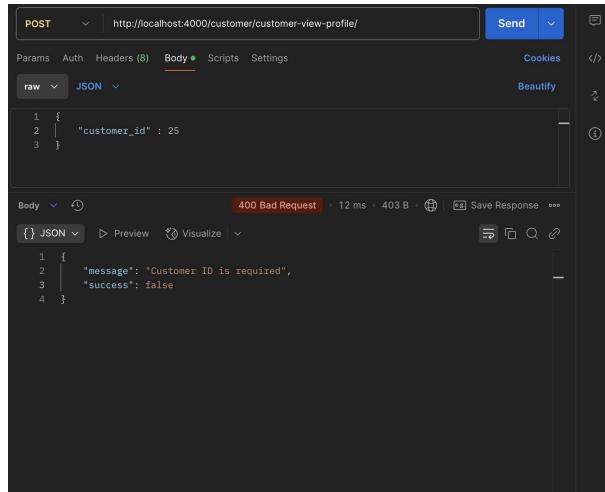


Figure 18: Viewing Profile Edge Case

2.2.13 Fetch Top 5 Dishes

API Endpoint: /customer-fetch-top5

Test Owner: Parnika Mittal

Date: 02/04/2025

Test Description: This test checks if the API returns the top 5 most frequently ordered dishes

Test Result: Fetches the top 5 dishes according to the order count. The picture of these dishes, on clicking will redirect to the corresponding canteen's details page.

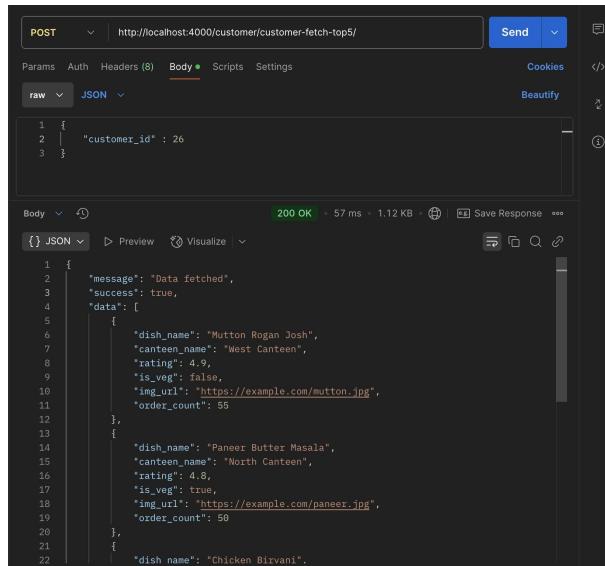


Figure 19: Fetching the Recommended Dishes

2.2.14 Reset Password

API Endpoint: /customer-reset-password

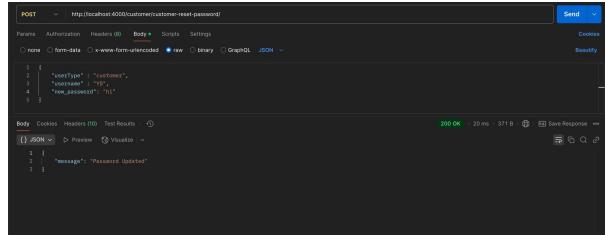
Test Owner: Kamatam Akshay

Date: 02/04/2025

Test Description: This test checks if the API allows the customer to reset their password successfully

Test Result: Allows the customer to rest their password, in case they have forgotten their

password. They will provide their username or email address and using the OTP the new password will get updated. The new password must follow the set security standards.



A screenshot of the Postman application interface. The request is a POST to `http://localhost:4000/customer/customer-reset-password`. The body contains the following JSON:

```
1 {  
2   "userType": "customer",  
3   "username": "test",  
4   "new_password": "1234"  
5 }
```

The response status is 200 OK, with a message: "Password Updated".

Figure 20: Resetting Password

2.2.15 Update Profile Photo

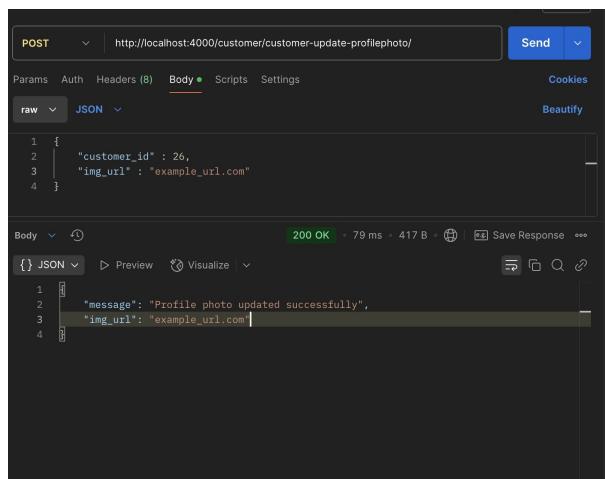
API Endpoint: /customer-update-profilephoto

Test Owner: Kshitij Gupta

Date: 02/04/2025

Test Description: This test checks if the API successfully updates the profile picture of the customer

Test Result: Allows the customer to change their profile photo and make the changes in the database. The image is uploaded and a unique url is fetched and is updated in the backend.



A screenshot of the Postman application interface. The request is a POST to `http://localhost:4000/customer/customer-update-profilephoto/`. The body contains the following JSON:

```
1 {  
2   "customer_id": 26,  
3   "img_url": "example_url.com"  
4 }
```

The response status is 200 OK, with a message: "Profile photo updated successfully", and an img_url: "example_url.com".

Figure 21: Updating Profile Photo

2.2.16 Update Profile

API Endpoint: /customer-update-profile

Test Owner: Nikhilesh Joshi

Date: 31/03/2025

Test Description: This test checks if the API correctly updates the customer's profile information

Test Result: Allows the user to alter some attributes of their profile and make the appropriate changes in the database. They may not change their username.

The screenshot shows a Postman request to `http://localhost:4000/customer/customer-update-profile/` using a POST method. The request body is JSON, containing:

```

1 {
2   "customer_id": 26,
3   "name": "Yatharth Dangi",
4   "phone_number": "9999999999",
5   "address": "D-214, IIT Kanpur"
}

```

The response status is 200 OK, with a response time of 52 ms and a response size of 383 B. The response body is:

```

1 {
2   "message": "Profile updated successfully"
3 }

```

Figure 22: Updating Profile Details

2.2.17 Change Password

API Endpoint: /customer-change-password

Test Owner: Suvid Goel

Date: 04/04/2025

Test Description: This test checks if the API allows the customer to successfully change their current password

Test Result: Allows the user to change his password as long as the new password follows the set security standards and they remember their old password.

The screenshot shows a Postman request to `http://localhost:4000/customer/customer-change-password/` using a POST method. The request body is JSON, containing:

```

1 {
2   "customer_id": 25,
3   "currentPassword": "ok",
4   "newPassword": "hi"
5 }

```

The response status is 200 OK, with a response time of 175 ms and a response size of 466 B. The response body is:

```

1 {
2   "success": true,
3   "message": "Password changed successfully"
4 }

```

Figure 23: Changing the Password

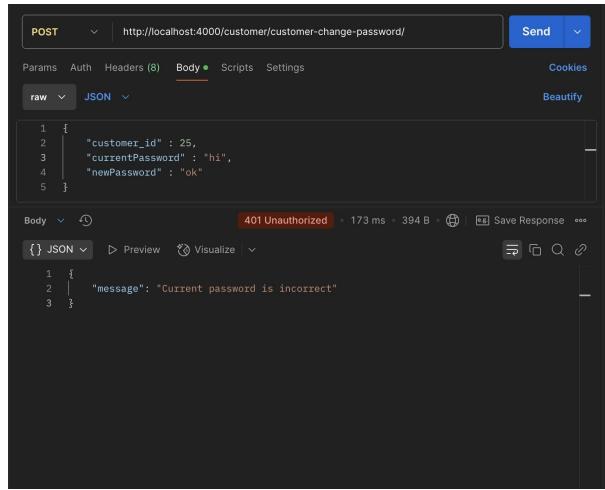


Figure 24: When entered password is wrong

2.2.18 Fetch Dishes

API Endpoint: /customer-fetch-dishes

Test Owner: Yatharth Dangi

Date: 03/04/2025

Test Description: This test checks if the API returns a list of dishes available across canteens

Test Result: Returns a list of dishes across all canteens successfully allowing the user to add them to cart or remove them directly without entering a canteen's menu.

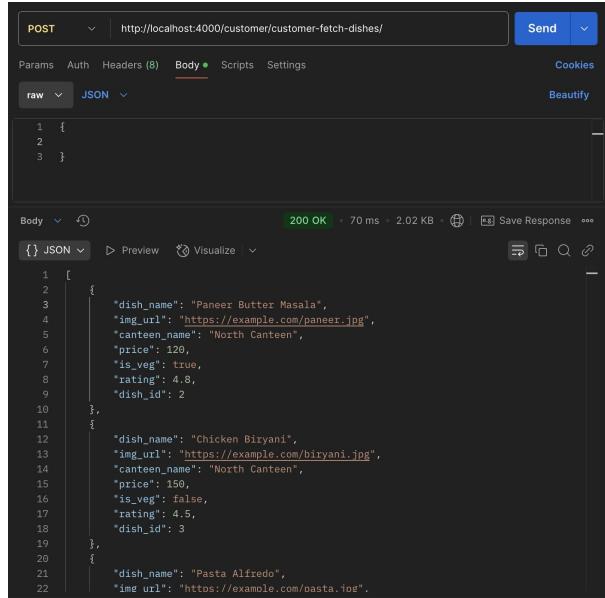


Figure 25: Fetching available dishes

2.2.19 Fetch Canteens

API Endpoint: /customer-fetch-canteens

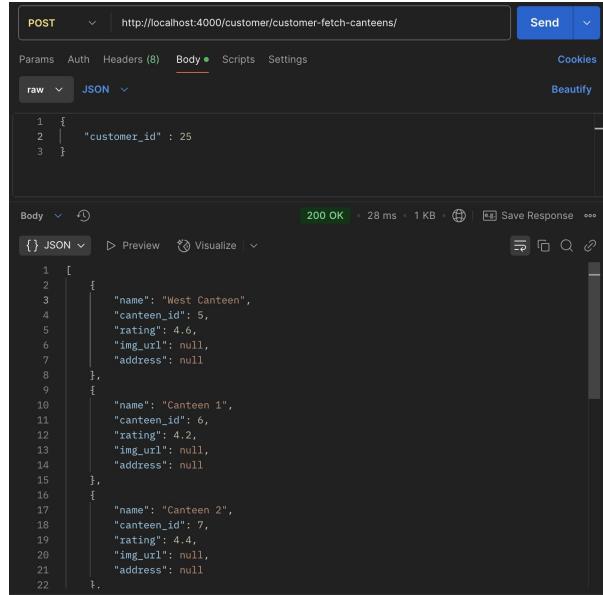
Test Owner: Rohit Vijaykumar Soman

Date: 04/04/2025

Test Description: This test checks if the API returns a list of registered canteens with their

details

Test Result: Returns a list of all registered canteens with their details successfully.



The screenshot shows a POST request to `http://localhost:4000/customer/customer-fetch-canteens/`. The request body is a JSON object with a single key-value pair: `"customer_id": 25`. The response status is 200 OK, with a response time of 28 ms and a response size of 1 KB. The response body is a JSON array containing three objects, each representing a canteen with fields: name, canteen_id, rating, img_url, and address. The first canteen is "West Canteen" (id 5), the second is "Canteen 1" (id 6), and the third is "Canteen 2" (id 7).

```
POST http://localhost:4000/customer/customer-fetch-canteens/ Send
Params Auth Headers (8) Body Scripts Settings Cookies Beautify
raw JSON
1 {
2   "customer_id": 25
3 }

Body 200 OK 28 ms 1 KB Save Response
[{"name": "West Canteen", "canteen_id": 5, "rating": 4.6, "img_url": null, "address": null}, {"name": "Canteen 1", "canteen_id": 6, "rating": 4.2, "img_url": null, "address": null}, {"name": "Canteen 2", "canteen_id": 7, "rating": 4.4, "img_url": null, "address": null}]
```

Figure 26: Fetching all registered canteens

2.2.20 Fetch Menu

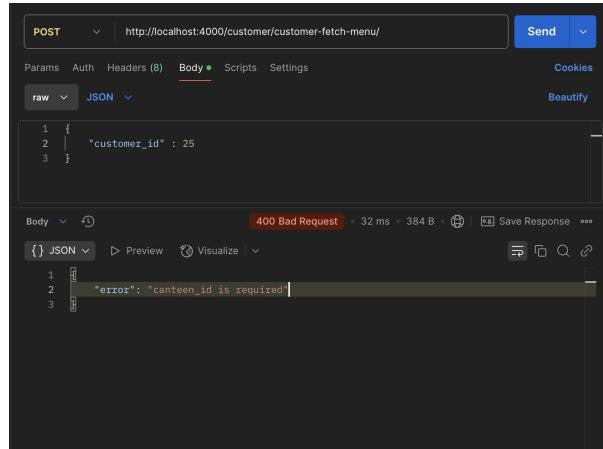
API Endpoint: /customer-fetch-menu

Test Owner: Yashaswi Raj

Date: 02/04/2025

Test Description: This test checks if the API returns the complete menu of a selected canteen

Test Result: Returns a list of food items available in the selected canteen successfully along with their count if they are already added in a cart.



The screenshot shows a POST request to `http://localhost:4000/customer/customer-fetch-menu/`. The request body is a JSON object with a single key-value pair: `"customer_id": 25`. The response status is 400 Bad Request, with a response time of 32 ms and a response size of 384 B. The response body is a JSON object with a single key `"error": "customer_id is required"`.

```
POST http://localhost:4000/customer/customer-fetch-menu/ Send
Params Auth Headers (8) Body Scripts Settings Cookies Beautify
raw JSON
1 {
2   "customer_id": 25
3 }

Body 400 Bad Request 32 ms 384 B Save Response
{"error": "customer_id is required"}]
```

Figure 27: Fetching Menu of a Canteen

```

POST http://localhost:4000/customer/customer-add-favourite/
{
  "customer_id": 25,
  "dish_id": 5
}
200 OK
{
  "message": "Dish added to favorites successfully",
  "success": true
}

```

Figure 28: Edge Case

2.2.21 View Cart

API Endpoint: /customer-view-cart

Test Owner: Arijant Kumar

Date: 02/04/2025

Test Description: This test checks if the API returns the current cart contents of the customer

Test Result: Returns the details of all dishes currently in the cart of the customer successfully. If the cart is empty then appropriate page is rendered on the site.

```

POST http://localhost:4000/customer/customer-view-cart/
{
  "customer_id": 25
}
200 OK
{
  "canteen_id": 1,
  "is_delivery": false,
  "bill_amount": 120,
  "discount_amount": 0,
  "final_amount": 120,
  "dishes": [
    {
      "dish_name": "Paneer Butter Masala",
      "img_url": "https://example.com/paneer.jpg",
      "rating": 4.8,
      "price": 120,
      "is_veg": true,
      "dish_id": 2,
      "dish_tag": "North Indian"
    }
  ],
  "dish_map": {
    "2": 1
  }
}

```

Figure 29: Viewing Cart

2.2.22 Requesting Order

API Endpoint: /request-order

Test Owner: Yatharth Dangi

Date: 30/03/2025

Test Description: This test checks if the API places the order and makes necessary changes

Test Result: Places the order at the requested canteen. The amount is deducted from the wallet of the customer, order details have been added to the database and the canteen manager is also informed.

2.2.23 Placing Order

API Endpoint: /place-order

Test Owner: Rohit Vijaykumar Somanı

Date: 29/03/2025

Test Description: This test checks if the API requests the order from the canteen manager of the corresponding canteen

Test Result: The request is conveyed to the canteen manager of the canteen from which the order is requested. If the auto accept of the canteen manager is on, the order is automatically accepted. Otherwise, if the order is not accepted within 5 minutes of the request, it is automatically canceled.

2.3 Canteen Side Functionalities

Accessible by the Canteen Manager

NOTE: In unit testing, the canteen_id is sent in the body of a post request, but in the final web app, it will be sent via an http-only cookie and not inside the body of the request

2.3.1 Active Orders / Pending Order Requests

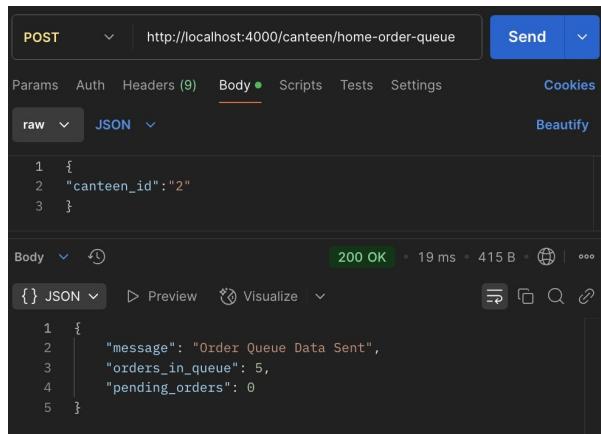
API Endpoint: /canteen/home-order-queue

Test Owner: Arihant Kumar

Date: 01/04/2025

Test Description: This test checks if the API returns the correct number of active orders and pending order requests of the past 24 hours

Test Result: The API is working as expected



The screenshot shows the Postman application interface. A POST request is being made to the URL `http://localhost:4000/canteen/home-order-queue`. The request body is set to JSON and contains the following data:

```
1 {
2   "canteen_id": "2"
3 }
```

The response status is 200 OK, with a response time of 19 ms and a response size of 415 B. The response body is also in JSON format:

```
1 {
2   "message": "Order Queue Data Sent",
3   "orders_in_queue": 5,
4   "pending_orders": 0
5 }
```

Figure 30: home-order-queue request

2.3.2 Confirmed / Pending Reservation Requests

API Endpoint: /canteen/home-reservation-queue

Test Owner: Dnyanesh Hemendra Shinde

Date: 02/04/2025

Test Description: This test checks if the API returns the correct number of confirmed and pending reservation requests of the past week

Test Result: The API is working as expected

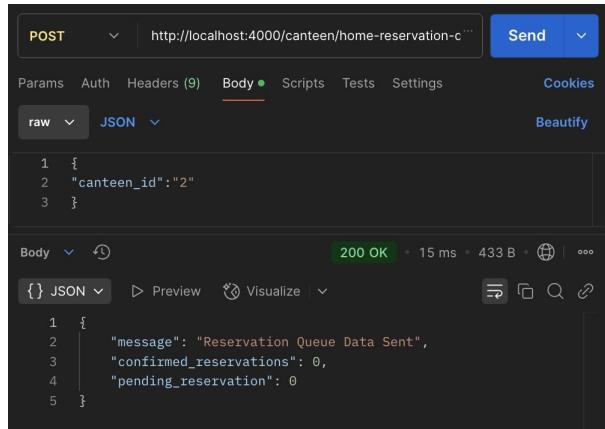


Figure 31: home-reservation-queue request

2.3.3 Active Coupons / Discounts

API Endpoint: /canteen/home-discount-n-coupons

Test Owner: Nikhilesh Joshi

Date: 03/04/2025

Test Description: This test checks if the API returns the correct number of currently active discounts and coupons

Test Result: The API is working as expected

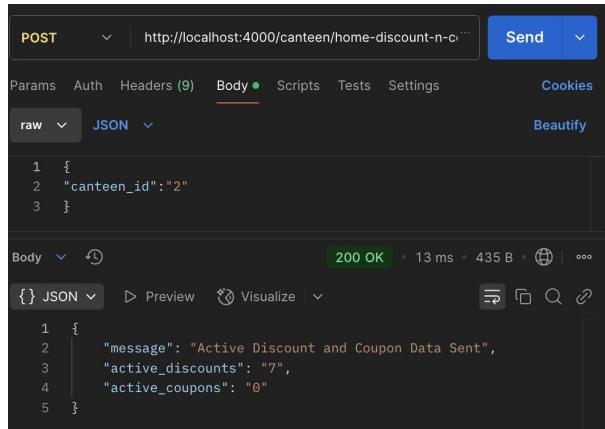


Figure 32: home-discount-n-coupons request

2.3.4 Trending Picks

API Endpoint: /canteen/trending-picks

Test Owner: Kamatam Akshay Reddy

Date: 01/04/2025

Test Description: This test checks if the API correctly returns the data of the most ordered dishes from the canteen in the past week

Test Result: The API is working as expected

```

POST http://localhost:4000/canteen/trending-picks
Body
{
  "canteen_id": "2"
}

{
  "message": "Trending Picks Sent",
  "trending_picks": [
    {
      "dish_name": "Veg Biryani",
      "dish_id": "6",
      "total_quantity": "10",
      "img_url": "img2.jpg"
    },
    {
      "dish_name": "Samosa",
      "dish_id": "9",
      "total_quantity": "13",
      "img_url": "img5.jpg"
    },
    {
      "dish_name": "Idli Sambar",
      "dish_id": "14",
      "total_quantity": "12",
      "img_url": "img10.jpg"
    }
  ]
}

```

Figure 33: trending-picks request

2.3.5 Order Queue

API Endpoint: /canteen/order-queue

Test Owner: Kshitij Gupta

Date: 31/03/2025

Test Description: This test checks if the API correctly returns the data orders received by the canteen in the past 24 hours

Test Result: The API is working as expected

```

POST http://localhost:4000/canteen/order-queue
Body
{
  "canteen_id": "2"
}

{
  "message": "Order Queue Sent",
  "orders": [
    {
      "order_id": 116,
      "canteen_id": 2,
      "status": 1,
      "payment_id": "54d3285e8e37d672de135cab5c921d7b",
      "coupon_id": null,
      "bill_amount": 350.57,
      "discount_amount": 12.94,
      "final_amount": 223.85,
      "time": "2025-04-04T13:08:34.309Z",
      "is_delivery": false,
      "customer_name": "hemendra",
      "customer_address": null,
      "customer_image": null,
      "dishes": [
        {
          "dish_name": "Poha",
          "quantity": 2
        },
        {
          "dish_name": "Masala Dosa",
          "quantity": 5
        }
      ]
    }
  ]
}

```

Figure 34: order-queue request

2.3.6 Accept Order

API Endpoint: /canteen/accept-order

Test Owner: Rohit Somani

Date: 31/03/2025

Test Description: This test checks if the API successfully updates the status of a given order to 'accepted' and returns a confirmation response

Test Result: The API is working as expected

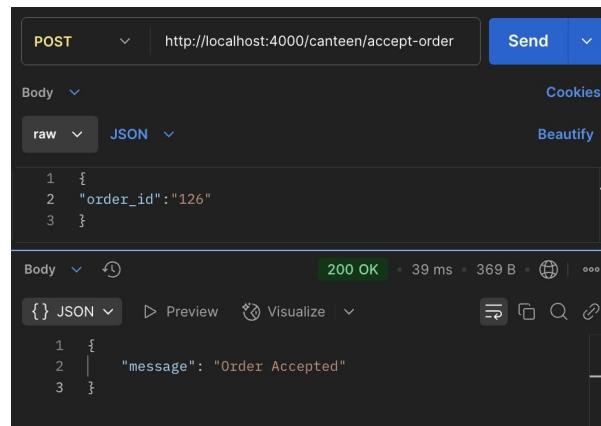


Figure 35: accept-order request

2.3.7 Reject Order

API Endpoint: /canteen/reject-order

Test Owner: Yashaswi Raj

Date: 01/04/2025

Test Description: This test checks if the API correctly marks an order as 'rejected' and returns the expected status update

Test Result: The API is working as expected

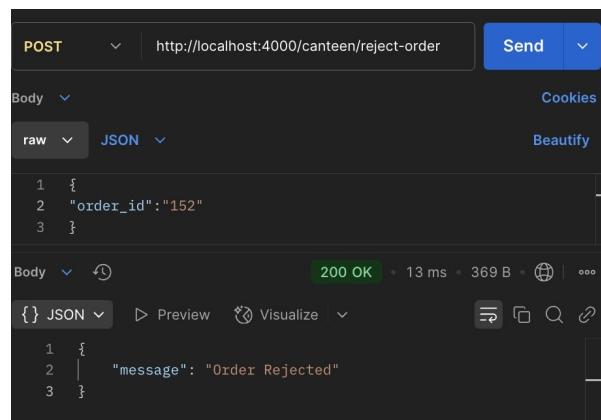


Figure 36: reject-order request

2.3.8 Mark Order As Ready

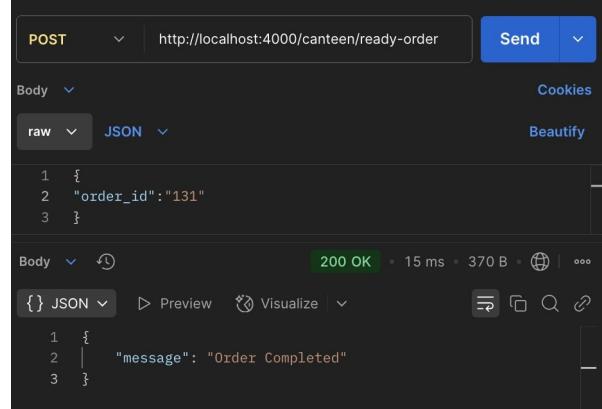
API Endpoint: /canteen/ready-order

Test Owner: Parnika Mittal

Date: 03/04/2025

Test Description: This test checks whether the API marks a previously accepted order as ready and sends a valid response

Test Result: The API is working as expected



The screenshot shows a Postman request to the endpoint `http://localhost:4000/canteen/ready-order`. The request method is `POST`. The request body is a JSON object with the key `order_id` and value `"131"`. The response status is `200 OK`, and the message is `"Order Completed"`.

Figure 37: ready-order request

2.3.9 Reservation Queue

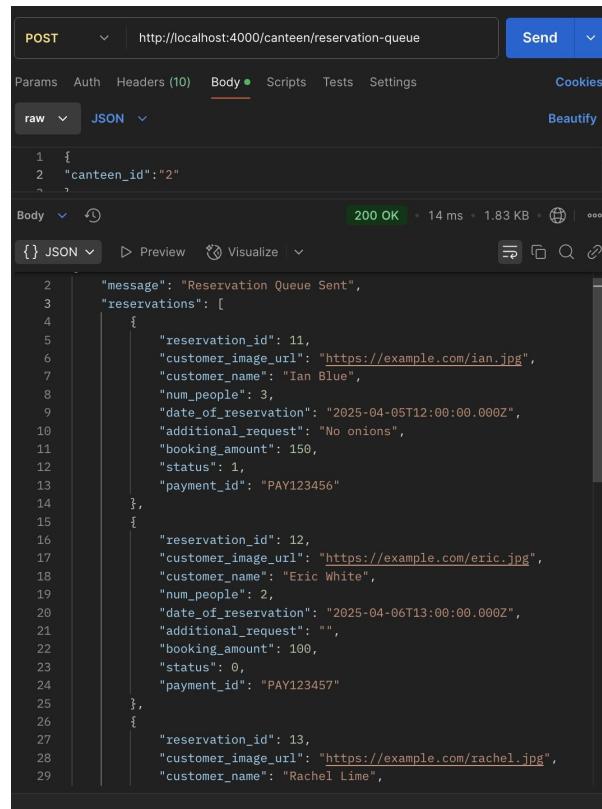
API Endpoint: /canteen/reservation-queue

Test Owner: Suvid Goel

Date: 30/03/2025

Test Description: This test checks if the API returns all reservation requests that are either pending or confirmed in the last 7 days

Test Result: The API is working as expected



The screenshot shows a Postman request to the endpoint `http://localhost:4000/canteen/reservation-queue`. The request method is `POST`. The request body is a JSON object with the key `canteen_id` and value `"2"`. The response status is `200 OK`, and it contains three reservation objects.

```
1 {
2   "canteen_id": "2"
3 }
4
5   "message": "Reservation Queue Sent",
6   "reservations": [
7     {
8       "reservation_id": 11,
9       "customer_image_url": "https://example.com/ian.jpg",
10      "customer_name": "Ian Blue",
11      "num_people": 3,
12      "date_of_reservation": "2025-04-05T12:00:00.000Z",
13      "additional_request": "No onions",
14      "booking_amount": 150,
15      "status": 1,
16      "payment_id": "PAY123456"
17    },
18    {
19      "reservation_id": 12,
20      "customer_image_url": "https://example.com/eric.jpg",
21      "customer_name": "Eric White",
22      "num_people": 2,
23      "date_of_reservation": "2025-04-06T13:00:00.000Z",
24      "additional_request": "",
25      "booking_amount": 100,
26      "status": 0,
27      "payment_id": "PAY123457"
28    },
29    {
30      "reservation_id": 13,
31      "customer_image_url": "https://example.com/rachel.jpg",
32      "customer_name": "Rachel Lime",
33    }
34 ]
```

Figure 38: reservation-queue request

2.3.10 Accept Reservation

API Endpoint: /canteen/accept-reservation

Test Owner: Yatharth Dangi

Date: 04/04/2025

Test Description: This test checks if the API correctly updates the status of a reservation to 'accepted' and returns a success message

Test Result: The API is working as expected

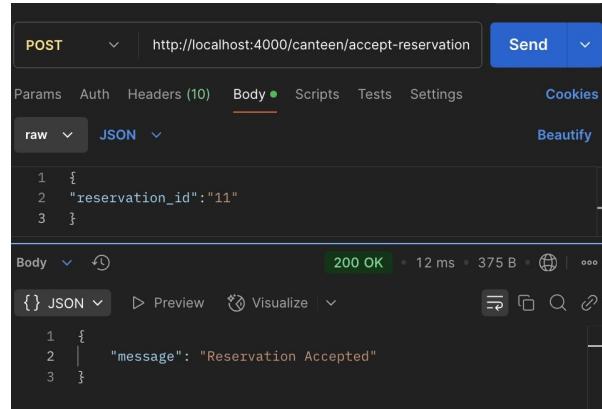


Figure 39: accept-reservation request

2.3.11 Reject Reservation

API Endpoint: /canteen/reject-reservation

Test Owner: Kamatam Akshay Reddy

Date: 01/04/2025

Test Description: This test checks whether the API properly sets the reservation status to 'rejected' and returns the appropriate response

Test Result: The API is working as expected

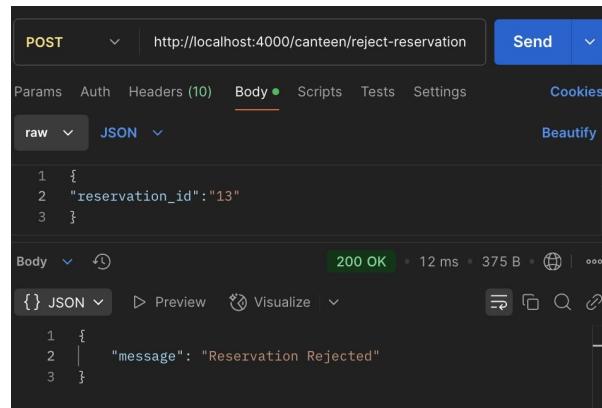


Figure 40: reject-reservation request

2.3.12 See Active Discounts

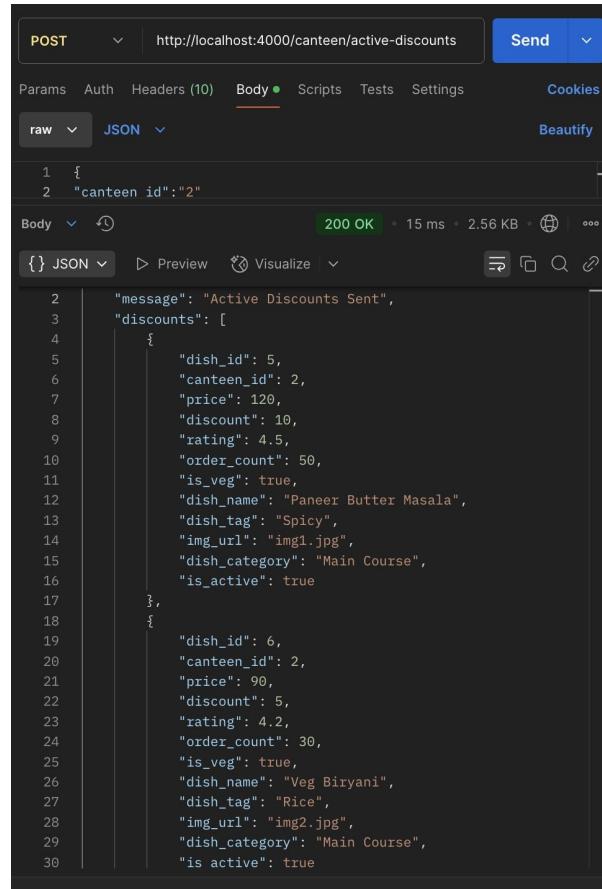
API Endpoint: /canteen/active-discounts

Test Owner: Dnyanesh Hemendra Shinde

Date: 01/04/2025

Test Description: This test checks if the API returns a list of all currently active discounts for the canteen

Test Result: The API is working as expected



```
POST http://localhost:4000/canteen/active-discounts
```

Params Auth Headers (10) Body Scripts Tests Settings Cookies

raw JSON Beautify

```
1 {
2   "canteen_id": "2"
```

Body 200 OK 15 ms 2.56 KB

{ } JSON Preview Visualize

```
2   "message": "Active Discounts Sent",
3   "discounts": [
4     {
5       "dish_id": 5,
6       "canteen_id": 2,
7       "price": 120,
8       "discount": 10,
9       "rating": 4.5,
10      "order_count": 50,
11      "is_veg": true,
12      "dish_name": "Paneer Butter Masala",
13      "dish_tag": "Spicy",
14      "img_url": "img1.jpg",
15      "dish_category": "Main Course",
16      "is_active": true
17    },
18    {
19      "dish_id": 6,
20      "canteen_id": 2,
21      "price": 90,
22      "discount": 5,
23      "rating": 4.2,
24      "order_count": 30,
25      "is_veg": true,
26      "dish_name": "Veg Biryani",
27      "dish_tag": "Rice",
28      "img_url": "img2.jpg",
29      "dish_category": "Main Course",
30      "is_active": true
31    }
32  ]
```

Figure 41: active-discounts request

2.3.13 Create Discounts

API Endpoint: /canteen/create-discounts

Test Owner: Yatharth Dangi

Date: 04/04/2025

Test Description: This test checks if the API successfully adds a new discount and validates the input fields

Test Result: The API is working as expected

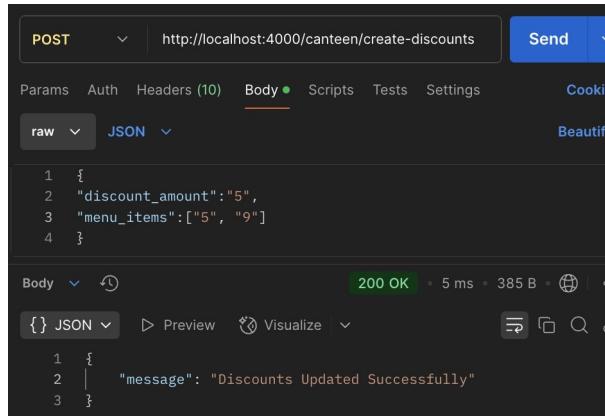


Figure 42: create-discounts request

2.3.14 Delete Discount

API Endpoint: /canteen/delete-discount

Test Owner: Kamatam Akshay Reddy

Date: 31/03/2025

Test Description: This test checks whether the API removes the specified discount and confirms deletion in the response

Test Result: The API is working as expected

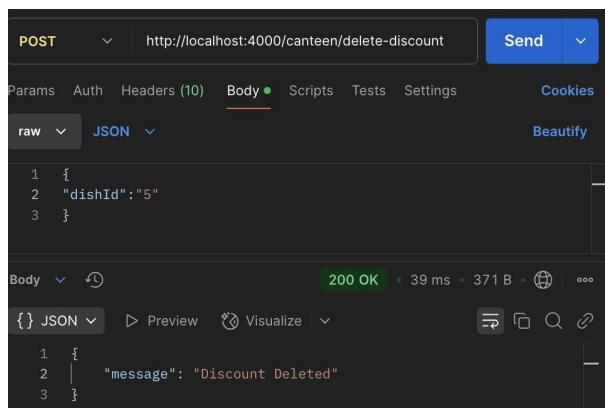


Figure 43: delete-discount request

2.3.15 See Active Coupons

API Endpoint: /canteen/active-coupons

Test Owner: Kshitij Gupta

Date: 03/04/2025

Test Description: This test checks if the API returns all the currently available and valid coupons for the canteen

Test Result: The API is working as expected

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/canteen/active-coupons`. The request body is a JSON object with a single key-value pair: `"canteen_id": "2"`. The response status is `200 OK` with a response time of 10 ms and a size of 973 B. The response body is a JSON object containing a message and a list of coupons. The message is `"Active Coupons Sent"`. The coupons array contains four objects, each representing a coupon with fields: `id`, `code`, `value`, `min_order_value`, `usage_limit`, and `valid_until`.

```

{
  "message": "Active Coupons Sent",
  "coupons": [
    {
      "id": 1,
      "code": "SAVE10",
      "value": 10,
      "min_order_value": 50,
      "usage_limit": 100,
      "valid_until": "2025-06-29T18:30:00.000Z"
    },
    {
      "id": 3,
      "code": "LUNCH15",
      "value": 15,
      "min_order_value": 80,
      "usage_limit": 75,
      "valid_until": "2025-07-14T18:30:00.000Z"
    },
    {
      "id": 5,
      "code": "BRKFST25",
      "value": 25,
      "min_order_value": 120,
      "usage_limit": 40,
      "valid_until": "2025-07-09T18:30:00.000Z"
    },
    {
      "id": 7,
      "code": "EVESPL10",
      "value": 10,
      "min_order_value": 30,
      "usage_limit": 50,
      "valid_until": "2025-07-24T18:30:00.000Z"
    }
  ]
}

```

Figure 44: active-coupons request

2.3.16 Create Coupon

API Endpoint: /canteen/create-coupon

Test Owner: Parnika Mittal

Date: 02/04/2025

Test Description: This test checks if the API creates a new coupon with the specified parameters and returns success

Test Result: The API is working as expected

The screenshot shows a Postman interface with a POST request to `http://localhost:4000/canteen/create-coupon`. The request body is a JSON object with fields: `canteen_id` (value 2), `code` (value ABCDEFG), `value` (value 100), `min_order_value` (value 500), `usage_limit` (value 10), and `valid_until` (value 2025-11-13). The response status is `200 OK` with a response time of 40 ms and a size of 382 B. The response body is a JSON object with a single key-value pair: `"message": "Coupon Created Successfully"`.

```

{
  "message": "Coupon Created Successfully"
}

```

Figure 45: create-coupon request

2.3.17 Delete Coupon

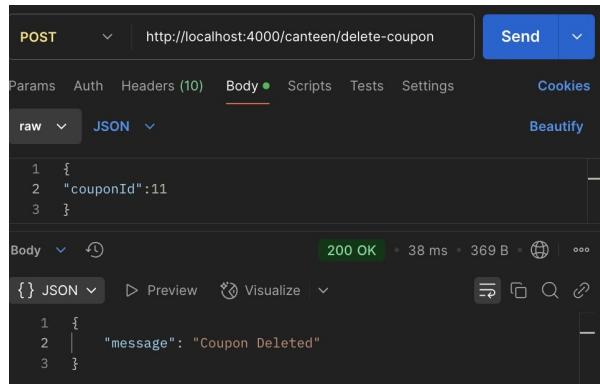
API Endpoint: /canteen/delete-coupon

Test Owner: Suvid Goel

Date: 29/03/2025

Test Description: This test verifies whether the API deletes the given coupon entry and returns a confirmation

Test Result: The API is working as expected



A screenshot of the Postman application interface. The top bar shows 'POST' and the URL 'http://localhost:4000/canteen/delete-coupon'. Below the URL, there are tabs for 'Params', 'Auth', 'Headers (10)', 'Body' (which is selected), 'Scripts', 'Tests', 'Settings', and 'Cookies'. Under 'Body', the dropdown is set to 'JSON'. The JSON body is: { "couponId": 11 }. The response section shows a green '200 OK' status with a response time of 38 ms and a body size of 369 B. The response content is: { "message": "Coupon Deleted" }.

Figure 46: delete-coupon request

2.3.18 Get Profile Details

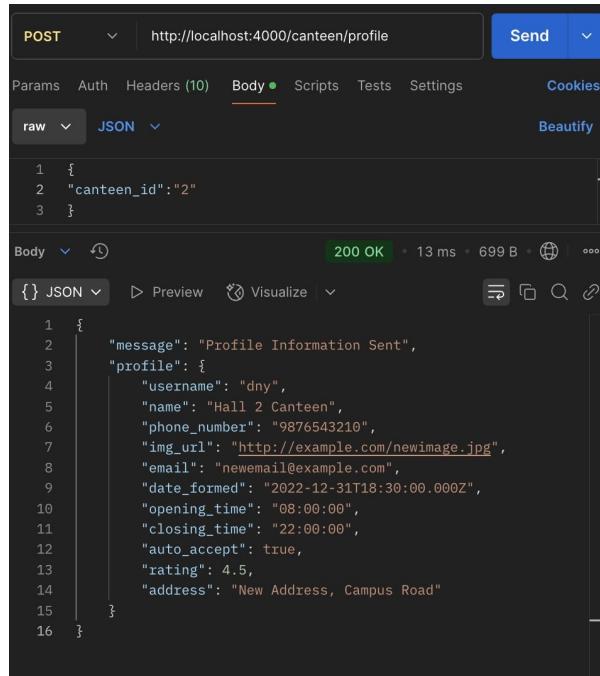
API Endpoint: /canteen/profile

Test Owner: Rohit Somani

Date: 31/03/2025

Test Description: This test checks whether the API fetches the profile details of the currently logged-in canteen

Test Result: The API is working as expected



A screenshot of the Postman application interface. The top bar shows 'POST' and the URL 'http://localhost:4000/canteen/profile'. Below the URL, there are tabs for 'Params', 'Auth', 'Headers (10)', 'Body' (which is selected), 'Scripts', 'Tests', 'Settings', and 'Cookies'. Under 'Body', the dropdown is set to 'JSON'. The JSON body is: { "canteen_id": "2" }. The response section shows a green '200 OK' status with a response time of 13 ms and a body size of 699 B. The response content is: { "message": "Profile Information Sent", "profile": { "username": "dny", "name": "Hall 2 Canteen", "phone_number": "9876543210", "img_url": "http://example.com/newimage.jpg", "email": "newemail@example.com", "date_formed": "2022-12-31T18:30:00.000Z", "opening_time": "08:00:00", "closing_time": "22:00:00", "auto_accept": true, "rating": 4.5, "address": "New Address, Campus Road" } }.

Figure 47: profile request

2.3.19 Edit Profile

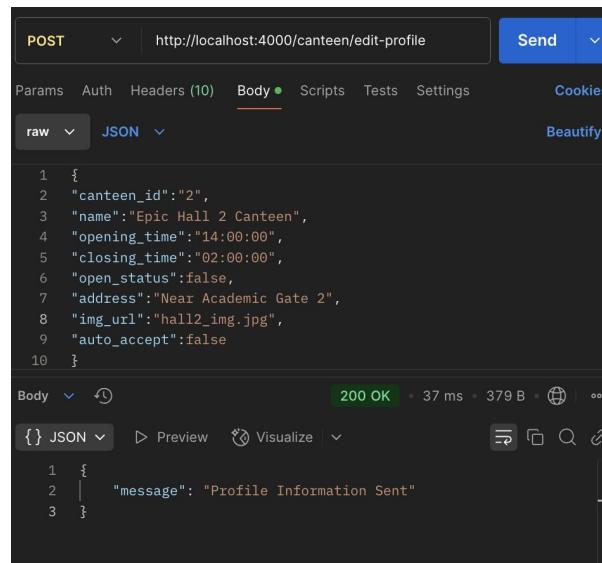
API Endpoint: /canteen/edit-profile

Test Owner: Yatharth Dangi

Date: 02/04/2025

Test Description: This test checks if the API updates the canteen's profile information and confirms the update

Test Result: The API is working as expected



The screenshot shows the Postman application interface. The top bar indicates a POST method and the URL <http://localhost:4000/canteen/edit-profile>. Below the URL, there are tabs for Params, Auth, Headers (10), Body (selected), Scripts, Tests, Settings, and Cookies. The Body tab is set to JSON, showing the following payload:

```
1  {
2   "canteen_id": "2",
3   "name": "Epic Hall 2 Canteen",
4   "opening_time": "14:00:00",
5   "closing_time": "02:00:00",
6   "open_status": false,
7   "address": "Near Academic Gate 2",
8   "img_url": "hall2_img.jpg",
9   "auto_accept": false
10 }
```

Below the body, the response status is 200 OK with a response time of 37 ms and a body size of 379 B. The response body is displayed as JSON:

```
1  {}
2  |   {
3  |   |   "message": "Profile Information Sent"
4  |   }
```

Figure 48: edit-profile request

2.3.20 Edit Password

API Endpoint: /canteen/edit-password

Test Owner: Yashaswi Raj

Date: 31/03/2025

Test Description: This test verifies if the API allows the canteen user to securely change their password

Test Result: The API is working as expected

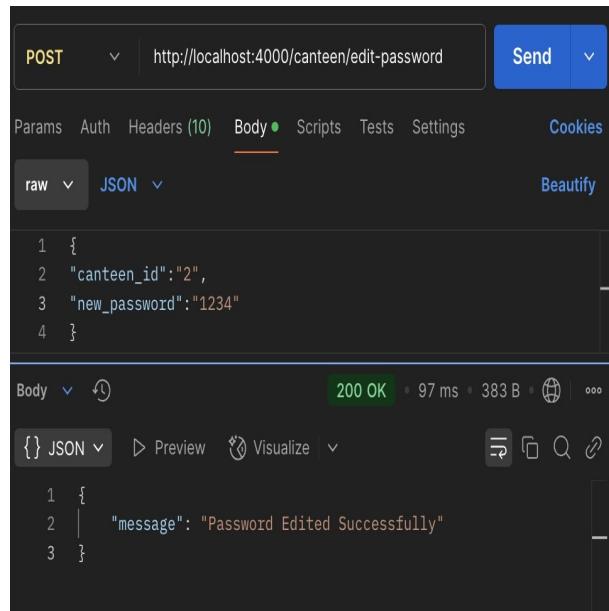


Figure 49: edit-password request

2.3.21 Show Menu Items

API Endpoint: /canteen/menu-items

Test Owner: Dnyanesh Hemendra Shinde

Date: 29/03/2025

Test Description: This test checks if the API returns all current dishes available in the canteen menu with its details

Test Result: The API is working as expected

```

POST http://localhost:4000/canteen/menu-items
Body
{
  "canteen_id": "2"
}

200 OK
{
  "menu": [
    {
      "dish_id": 6,
      "canteen_id": 2,
      "price": 90,
      "discount": 5,
      "rating": 4.2,
      "order_count": 30,
      "is_veg": true,
      "dish_name": "Veg Biryani",
      "dish_tag": "Rice",
      "img_url": "img2.jpg",
      "dish_category": "Main Course",
      "is_active": true
    },
    {
      "dish_id": 7,
      "canteen_id": 2,
      "price": 150,
      "discount": 15,
      "rating": 4.8,
      "order_count": 70,
      "is_veg": false,
      "dish_name": "Chicken Curry",
      "dish_tag": "Spicy",
      "img_url": "img3.jpg",
      "dish_category": "Main Course",
      "is_active": true
    }
  ]
}

```

Figure 50: menu-items request

2.3.22 Edit Dish Details

API Endpoint: /canteen/edit-dish

Test Owner: Nikhilesh Joshi

Date: 30/03/2025

Test Description: This test checks if the API successfully updates the details of an existing dish and confirms the modification

Test Result: The API is working as expected

```

POST http://localhost:4000/canteen/edit-dish
Body
{
  "dish_id": "6",
  "price": 100,
  "is_veg": true,
  "dish_name": "kebab",
  "dish_tag": "continental",
  "dish_category": "starters",
  "img_url": "sample.com"
}

200 OK
{
  "message": "Dish Details Edited Successfully"
}

```

Figure 51: edit-dish request

2.3.23 Add New Dish

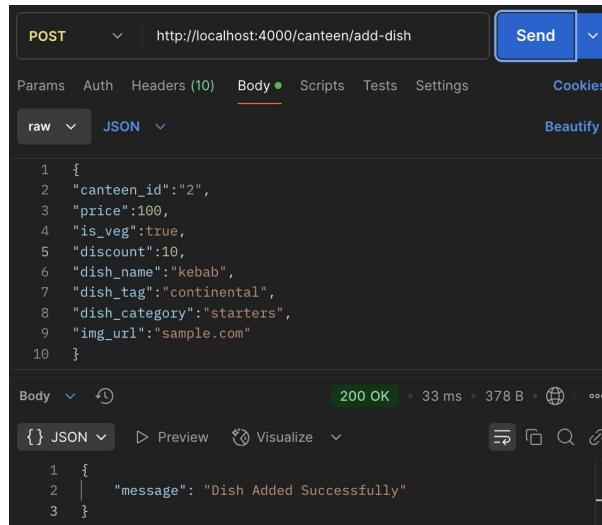
API Endpoint: /canteen/add-dish

Test Owner: Nikhilesh Joshi

Date: 21/03/2025

Test Description: This test verifies that the API correctly adds a new dish to the menu and returns a success response

Test Result: The API is working as expected



The screenshot shows the Postman application interface. A POST request is being made to the endpoint `http://localhost:4000/canteen/add-dish`. The request body is a JSON object containing dish details. The response status is 200 OK, and the message is "Dish Added Successfully".

```
POST http://localhost:4000/canteen/add-dish
{
  "canteen_id": "2",
  "price": 100,
  "is_veg": true,
  "discount": 10,
  "dish_name": "kebab",
  "dish_tag": "continental",
  "dish_category": "starters",
  "img_url": "sample.com"
}

200 OK
{
  "message": "Dish Added Successfully"
}
```

Figure 52: add-dish request

2.3.24 Get Statistics

API Endpoint: /canteen/get-statistics

Test Owner: Rohit Somani

Date: 02/04/2025

Test Description: This test checks if the API returns accurate statistical data about sales, orders, average order value and total customers along with their growth rate

Test Result: The API is working as expected

The screenshot shows a Postman interface with the following details:

- Method: POST
- URL: <http://localhost:4000/canteen/get-statistics>
- Body tab is selected, showing a JSON payload:

```
1 {
2   "canteen_id": "2"
3 }
```
- Response status: 200 OK
- Response time: 20 ms
- Response size: 571 B
- Response content (JSON):

```
1 {
2   "message": "Statistics Sent",
3   "curr_month": {
4     "total_sales": 2080.47,
5     "total_orders": 6,
6     "avg_order_value": 346.7450002034505,
7     "total_customers": 6
8   },
9   "delta": {
10    "sales": 2080.47,
11    "order": 6,
12    "avg_order_value": 346.7450002034505,
13    "customers": 6
14  }
15 }
```

Figure 53: get-statistics request

2.3.25 Get Graph Details

API Endpoint: /canteen/get-graphs

Test Owner: Arihant Kumar

Date: 30/03/2025

Test Description: This test checks whether the API provides the required graph data related to orders and sales trends

Test Result: The API is working as expected

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:4000/canteen/get-graphs
- Body (JSON):** {"canteen_id": "2"}
- Status:** 200 OK
- Response Body (JSON):**

```
1 {
2   "message": "Graph Data Sent",
3   "monthly": [
4     {
5       "order_date": "2025-03-06T18:30:00.000Z",
6       "total_sales": 0,
7       "total_orders": "0",
8       "avg_order_value": 0,
9       "total_customers": "0"
10      },
11      {
12        "order_date": "2025-03-07T18:30:00.000Z",
13        "total_sales": 0,
14        "total_orders": "0",
15        "avg_order_value": 0,
16        "total_customers": "0"
17      },
18      {
19        "order_date": "2025-03-08T18:30:00.000Z",
20        "total_sales": 0,
21        "total_orders": "0",
22        "avg_order_value": 0,
23        "total_customers": "0"
24      },
25      {
26        "order_date": "2025-03-09T18:30:00.000Z",
27        "total_sales": 0,
28        "total_orders": "0",
29        "avg_order_value": 0
30      }
31    ]
32  }
```

Figure 54: get-graphs request

3 Integration Testing

3.1 Authentication

3.1.1 Registering a customer

Module Details: This module integrates the user registration validation process, ensuring unique usernames and verifying the correctness and strength of user input during account creation

Test Owner: Yashaswi Raj

Test Date: 29/03/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** The system checks that the username entered by the user is not already taken by another user.

Result: An alert is displayed in red stating “*Username already taken*”

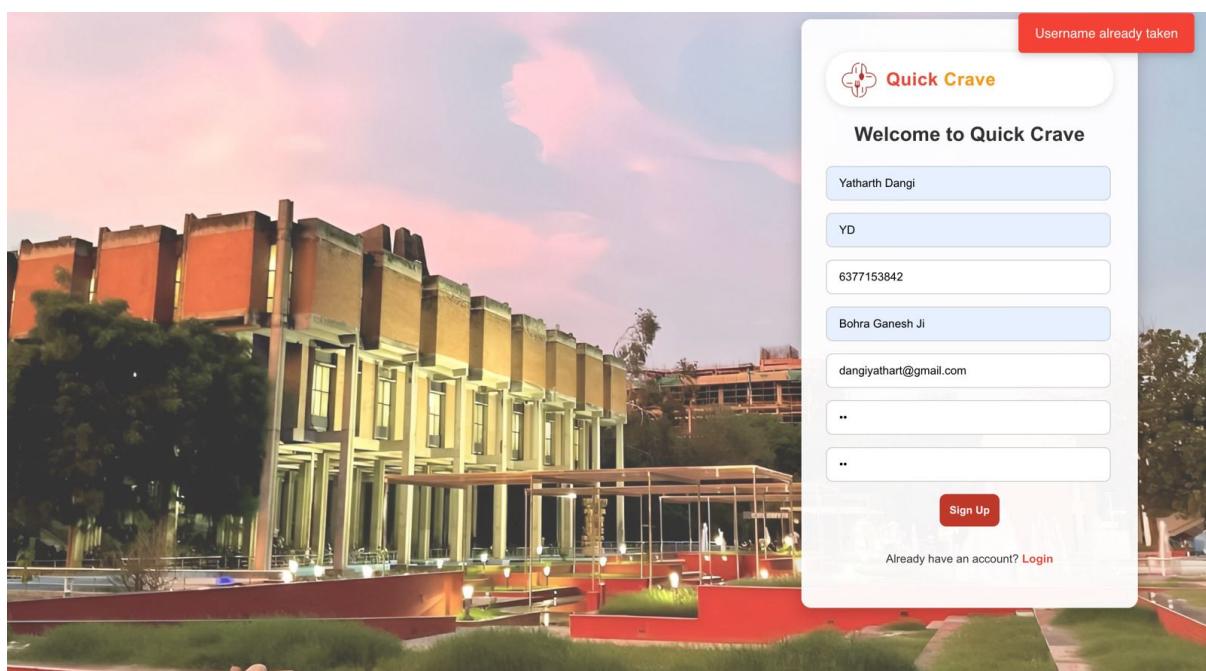


Figure 55: Customer Register user name already taken

- **Test Case 2:** The system checks that the email ID entered by the user is valid
Result: If the email format is invalid, the system displays the error message “*Enter a valid email address.*” beneath the email input field in red text. Examples of invalid formats (e.g., missing '@' or domain) trigger this message

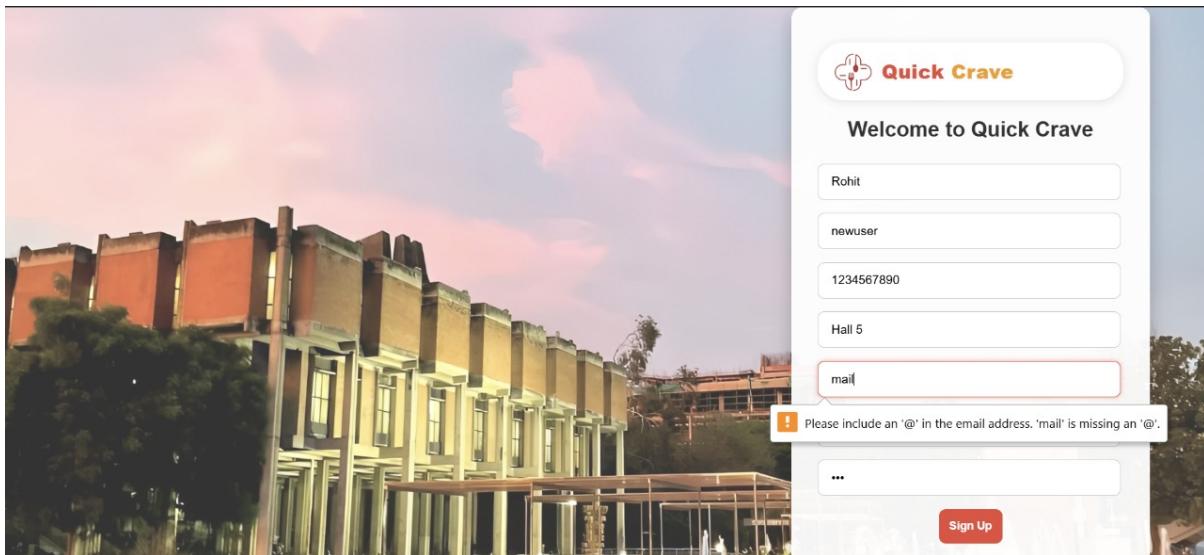


Figure 56: Customer register wrong email id

- **Test Case 3:** The system checks that the phone number entered by the user is valid. System checks if the phone number is of 10 digits.
Result: If the phone number does not meet the required format (e.g., incorrect digit count or inclusion of alphabets), the error message "*Phone number should be of 10 digits.*" is shown in the top right.

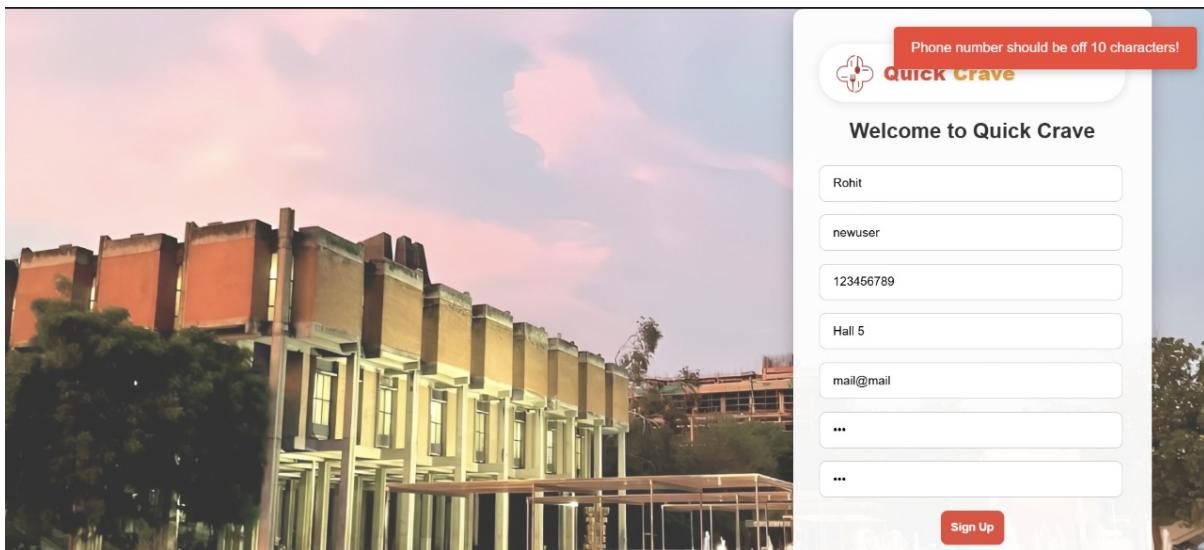


Figure 57: Customer Register wrong phone number

- **Test Case 4:** The system checks that the password set by the user follows standard protocols for a strong password
Result: If the password does not meet the required strength (e.g., at least 8 characters, including uppercase, lowercase, digit, and special character), the system displays a list of unmet criteria. The user is only allowed to proceed once all password requirements are fulfilled.

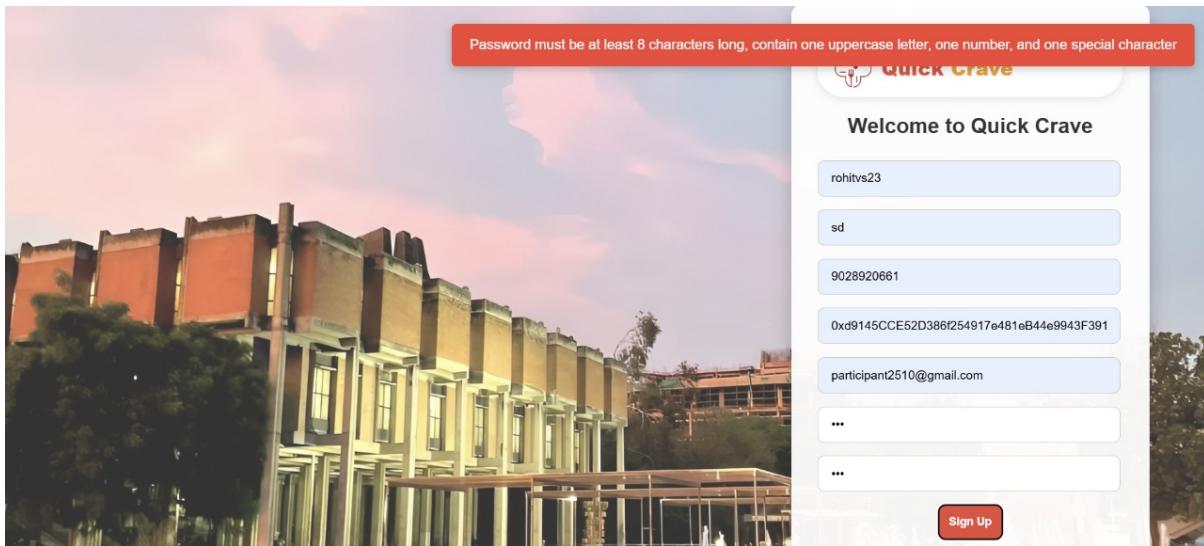


Figure 58: Customer Register Strong Password

3.1.2 Customer

- **Test Case 1:** The system checks that the OTP entered by the user is valid. System checks if the OTP is of 6 digits
Result: If the OTP does not meet the required format (e.g incorrect digit count), the error message "*Please enter a valid 6-digit OTP.*" is shown in the top right.

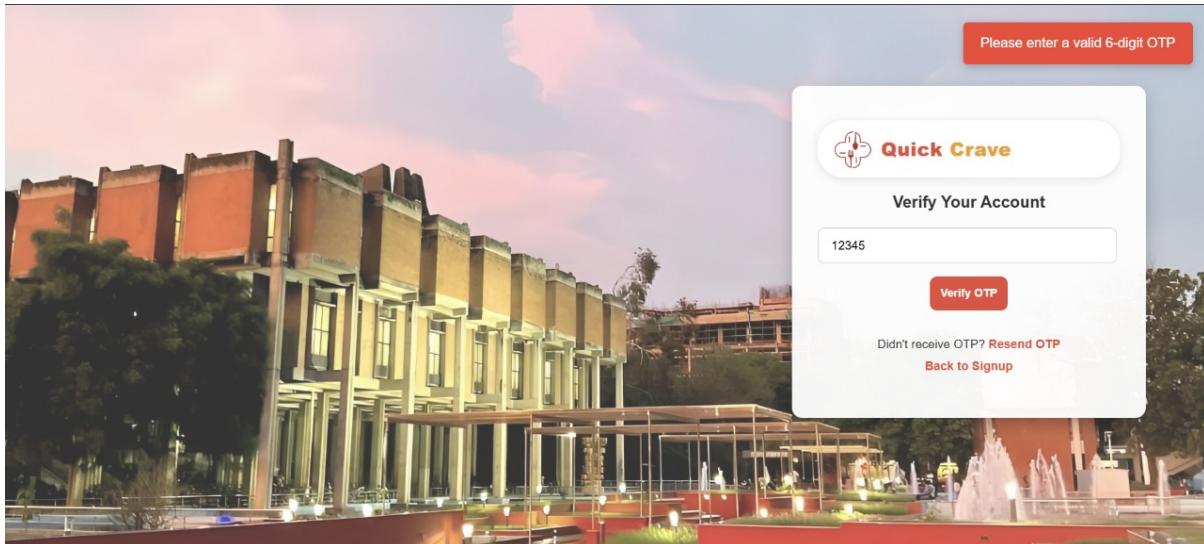


Figure 59: Customer register enter a valid six digit OTP

- **Test case 2:** The system checks that the 6-digit OTP entered by the user is valid.
Result: If he enters the wrong 6-digit OTP in the initial attempt, the error message "*Incorrect OTP. Attempts left: 2*" is shown in the top right.

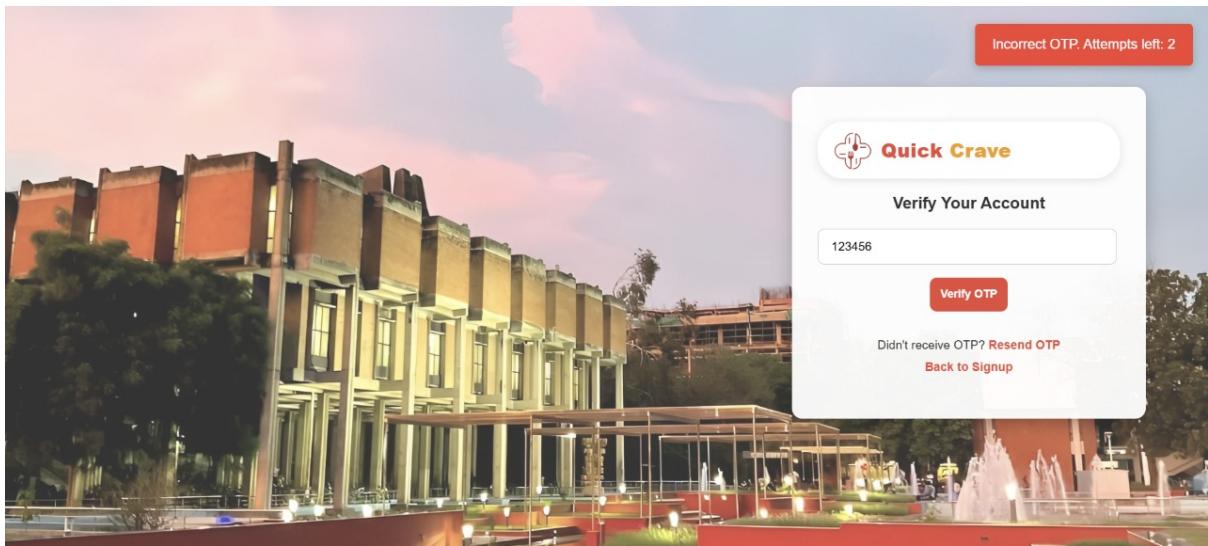


Figure 60: Customer register incorrect OTP

- **Test case 3:** The system checks that the sign up is successful.

Result: If all the details entered are valid then the system redirects the user to the login page and, the message “*Account Created Successfully*” is shown in the top right.

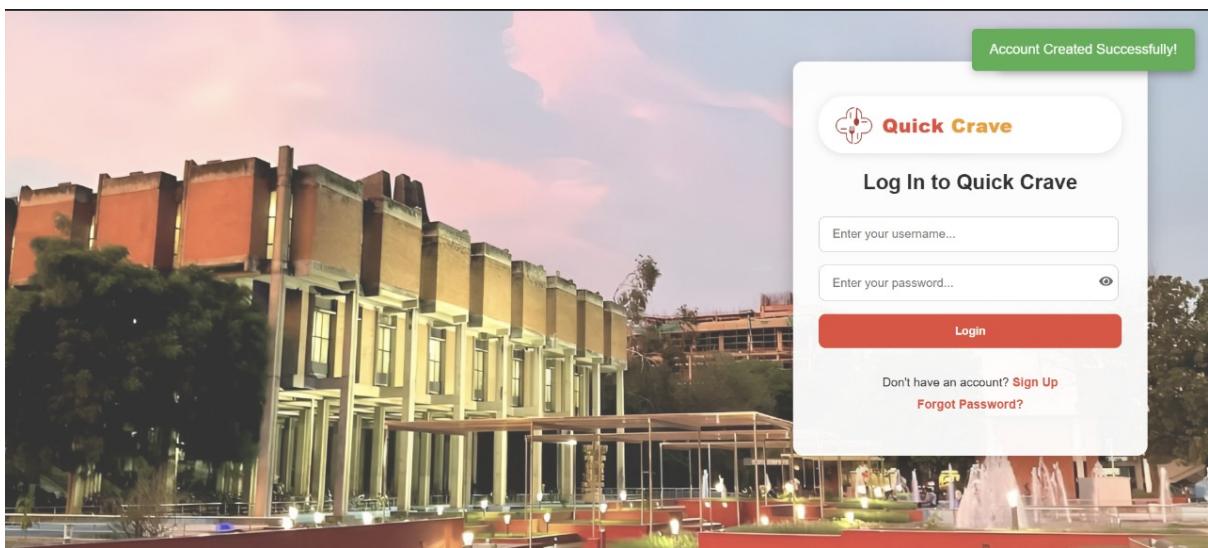


Figure 61: Customer registered successfully

3.1.3 Registering a canteen

Module Details: This module integrates the registration process for new canteens, providing necessary functionalities

Test Owner: Kshitij Gupta **Test Date:** 31/03/2025 **Test Results:** The integrated module is working as expected

- **Test Case :** Documents are added either by dragging them into the designated upload area or by searching and selecting from the local system
- **Test Case :** The system successfully sends the document through the app's mail service

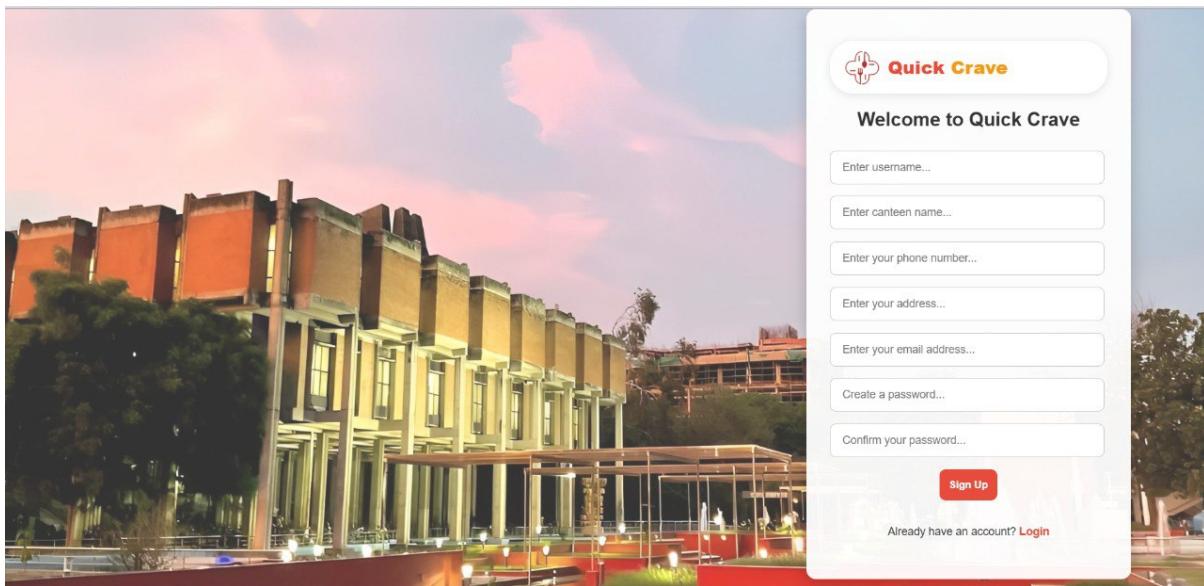


Figure 62: Customer sign up

3.1.4 Logging in as customer

Module Details: This module is integration of all components of Login Page

Test Owner: Dnyanesh Hemendra Shinde

Test Date: 31/03/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** The system checks that the username and password entered by the customer is valid.
Result: If the customer enters invalid username or password, the error message *Wrong username or password* is shown in the top right.

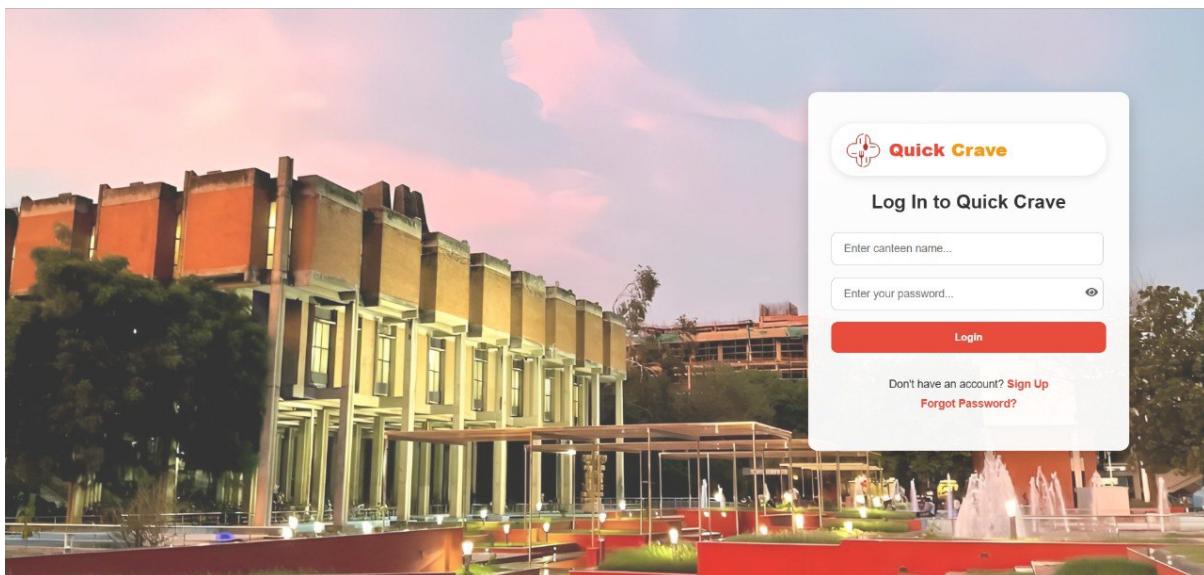


Figure 63: Customer Login Page

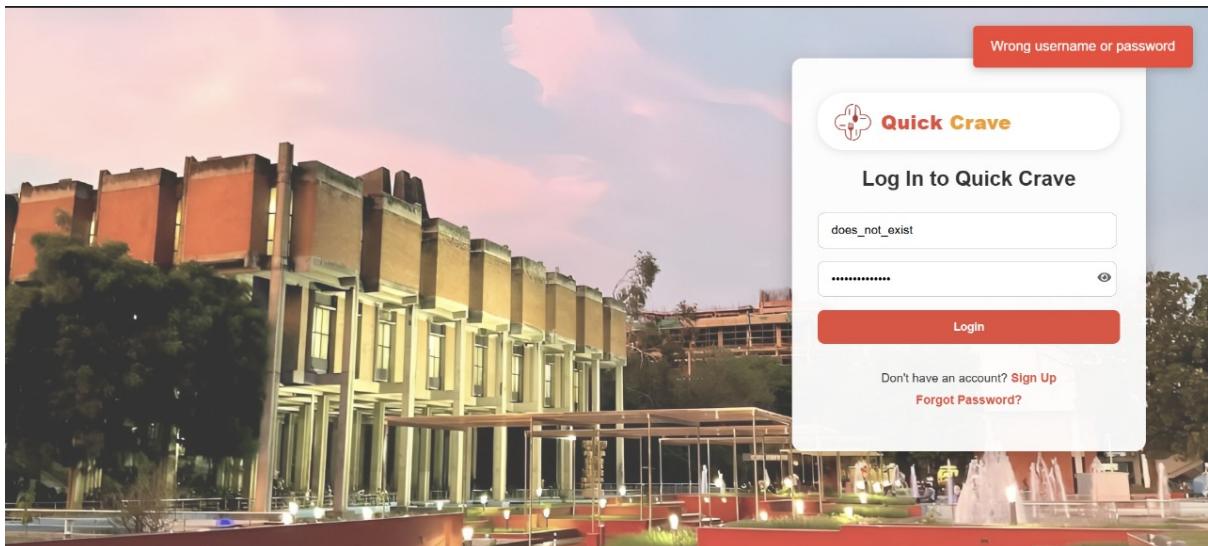


Figure 64: Customer Login wrong credentials

- **Test Case 2:** The system checks that the username and password entered by the customer is valid.
Result If the customer enters valid username and password then the system redirects the customer to the customer home page and the message “*Login successful as customer*” is shown in the top right.

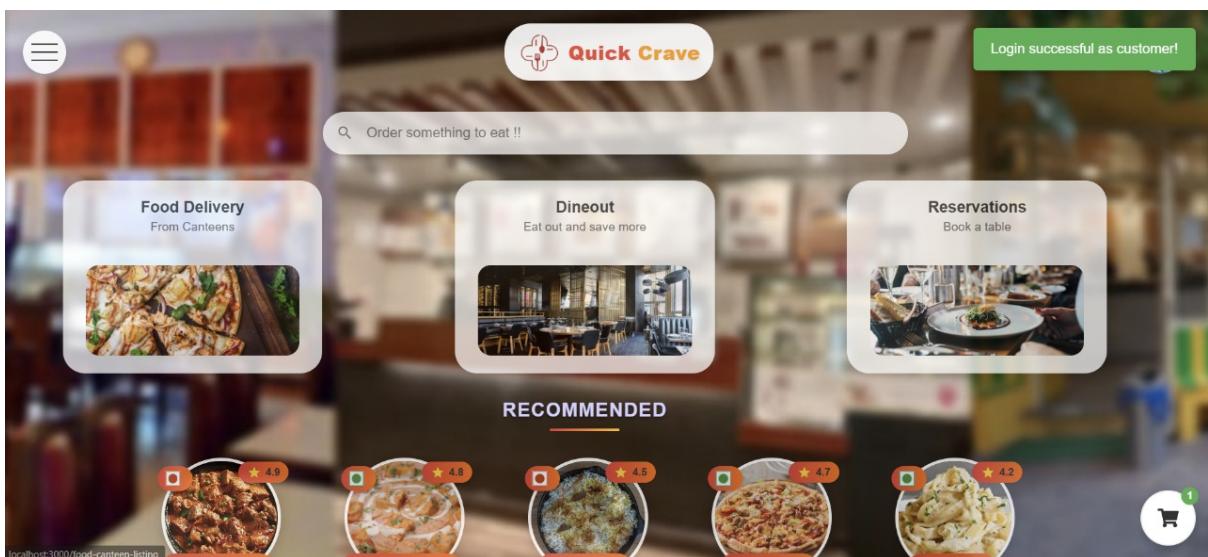


Figure 65: Customer Login Successful

3.1.5 Logging in as canteen manager

Module Details:

Test Owner: Yashaswi Raj **Test Date:** 01/04/2025 **Test Results:** The integrated module is working as expected

- **Test Case 1:** If incorrect credentials are entered
Result: Error prompt appears
- **Test Case 2:** Correct credentials are entered
Result: Successful Login

3.2 Customer Side

3.2.1 Customer Home

Module Details: This module integrates the Customer Home Page, providing necessary functionalities.

Test Owner: Nikhilesh Joshi

Test Date: 31/03/2025

Test Results: The integrated module loads all the components that are being called upon being directed to customer home page.

- **Test Case 1:** Upon loading the customer home page, the top 5 recommended dishes are being loaded and displayed on the home page of the customer.

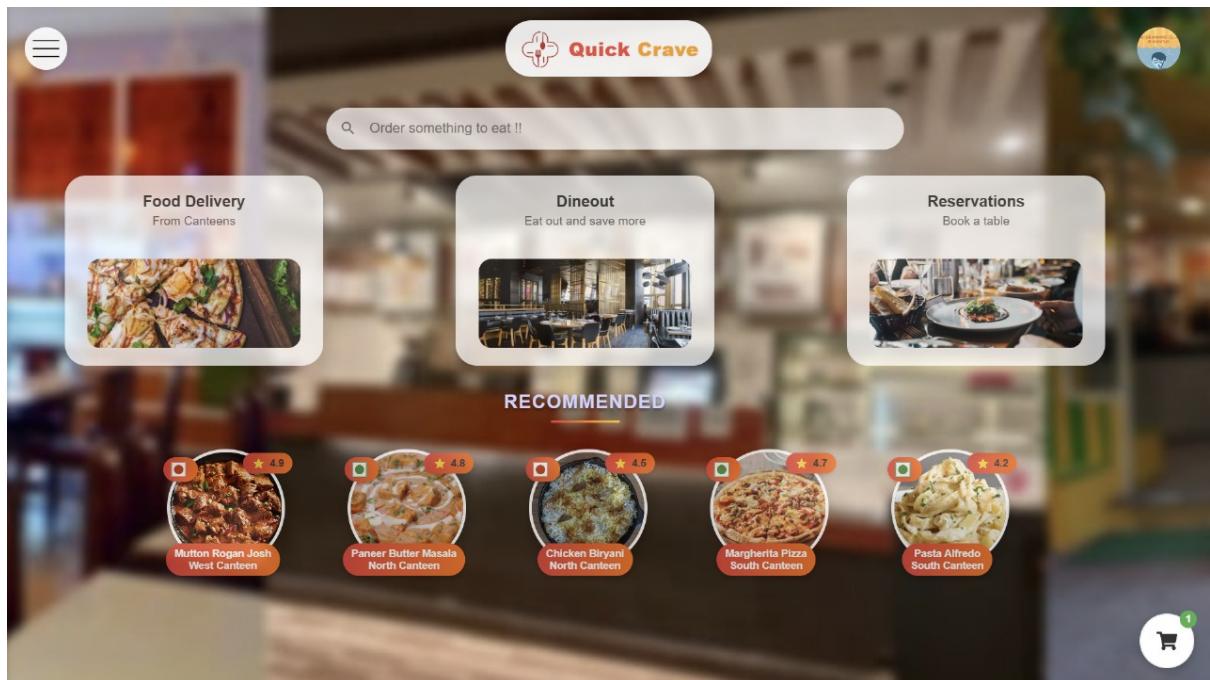


Figure 66: Customer Home Page

- **Test Case 2:** Upon loading the customer home page, the cart of the customer is also being loaded and displayed. The data is being taken from the database and then begin stored in the cart.
- **Test Case 3:** Upon loading the customer home page, the profile of the customer is also being loaded and displayed. The data is being taken from the database and begin stored in session storage and then called when required. Any changes being made to the profile is also being updated in the session storage so that data is consistent throughout.

3.2.2 Customer Profile

Module Details: This module integrates the Customer Profile Page, providing necessary functionalities.

Test Owner: Nikhilesh Joshi

Test Date: 01/04/2025

Test Results: The integrated module loads all the components that are being called upon being redirected to view profile page.

- **Test Case 1:** The details of the customer are correctly loaded along with the correct wallet balance as well the order details of the customer. Profile photo is also being loaded from the link provided through Cloudinary.

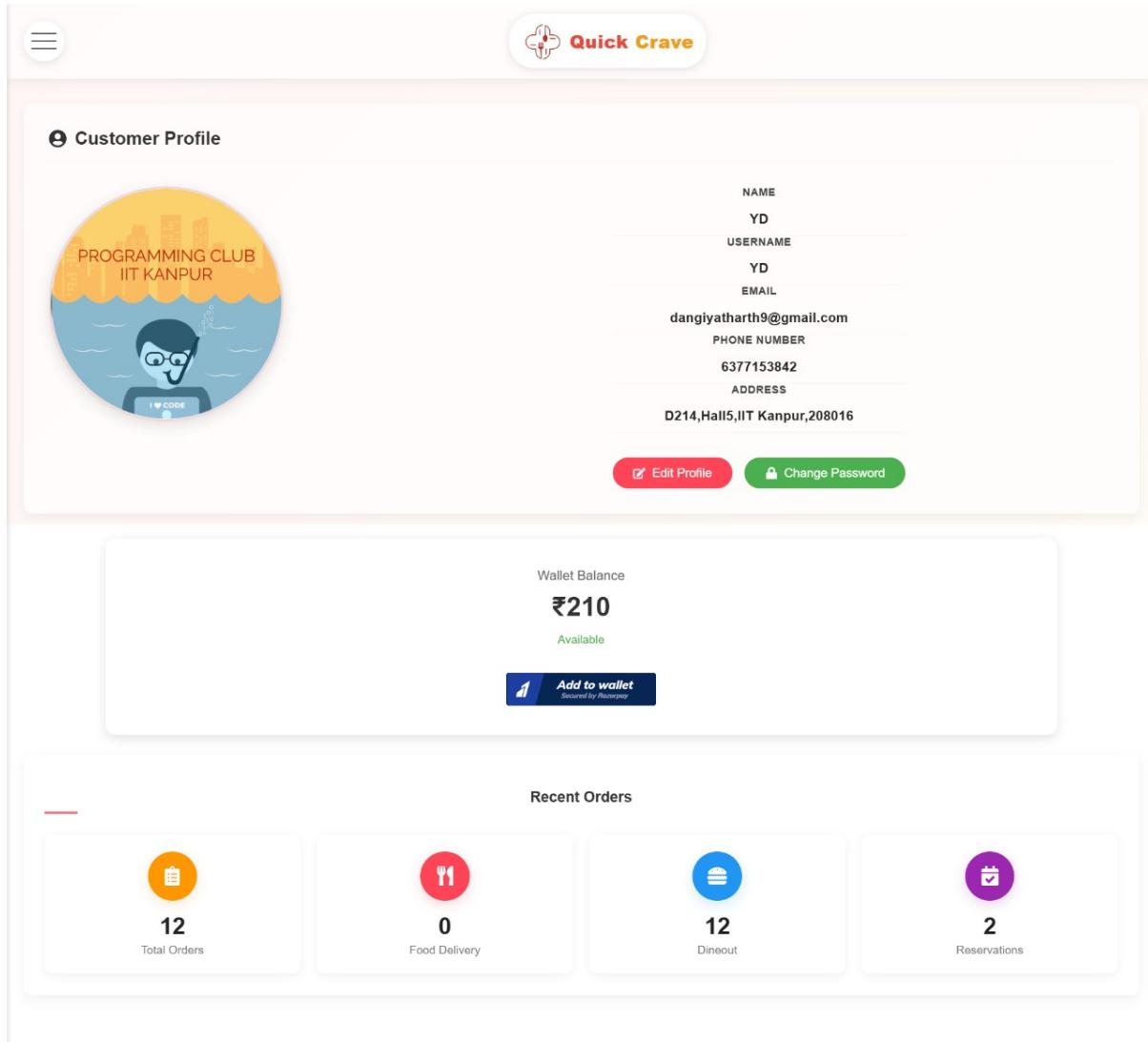


Figure 67: Customer Profile Page

- **Test Case 2:** The system checks that the customer is able to edit the profile.
Result: When the customer presses the edit profile button, the system navigates him to the edit profile page.

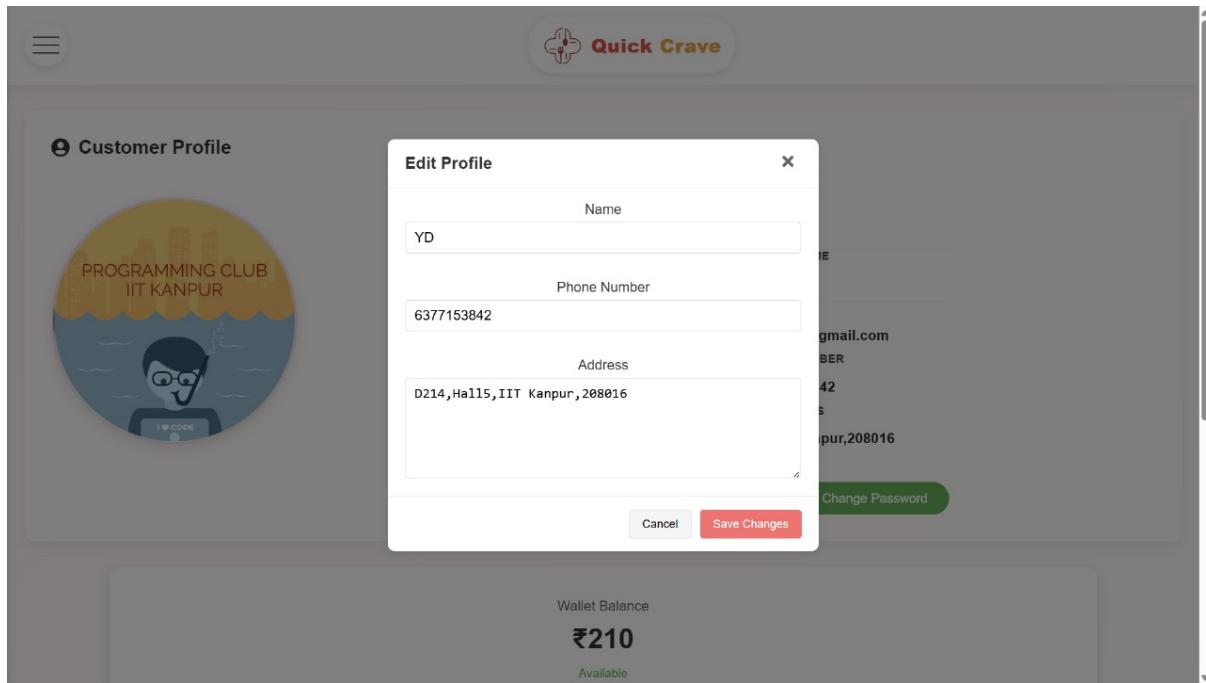


Figure 68: Customer Edit Profile

- **Test Case 3:** The system checks that the profile is updated successfully.

Result: When the profile is updated successfully the system navigates the customer back to customer profile page and the message “*Profile updated successfully!*” is shown in the top right.

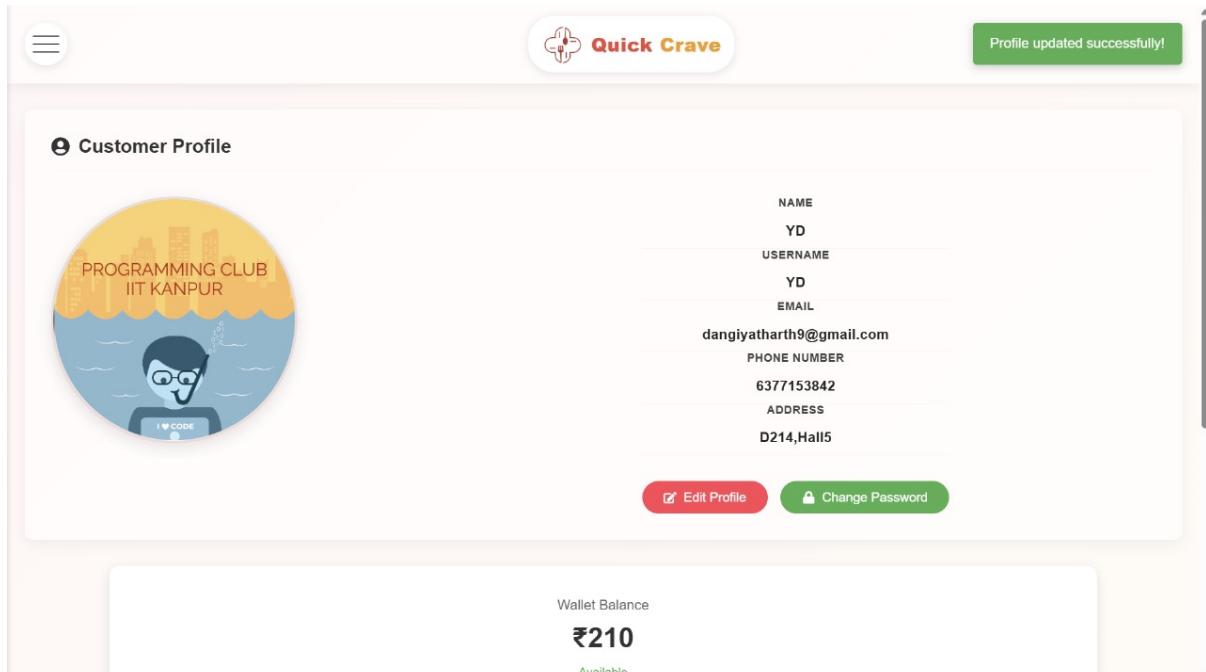


Figure 69: Customer Profile Updated Successfully

- **Test Case 4:** The system checks that the password is changed successfully.

Result: When the customer enters the new password and confirms the new password and presses Update Password then the message “*Password changed successfully! Logging you*

out..." is shown in the top right.

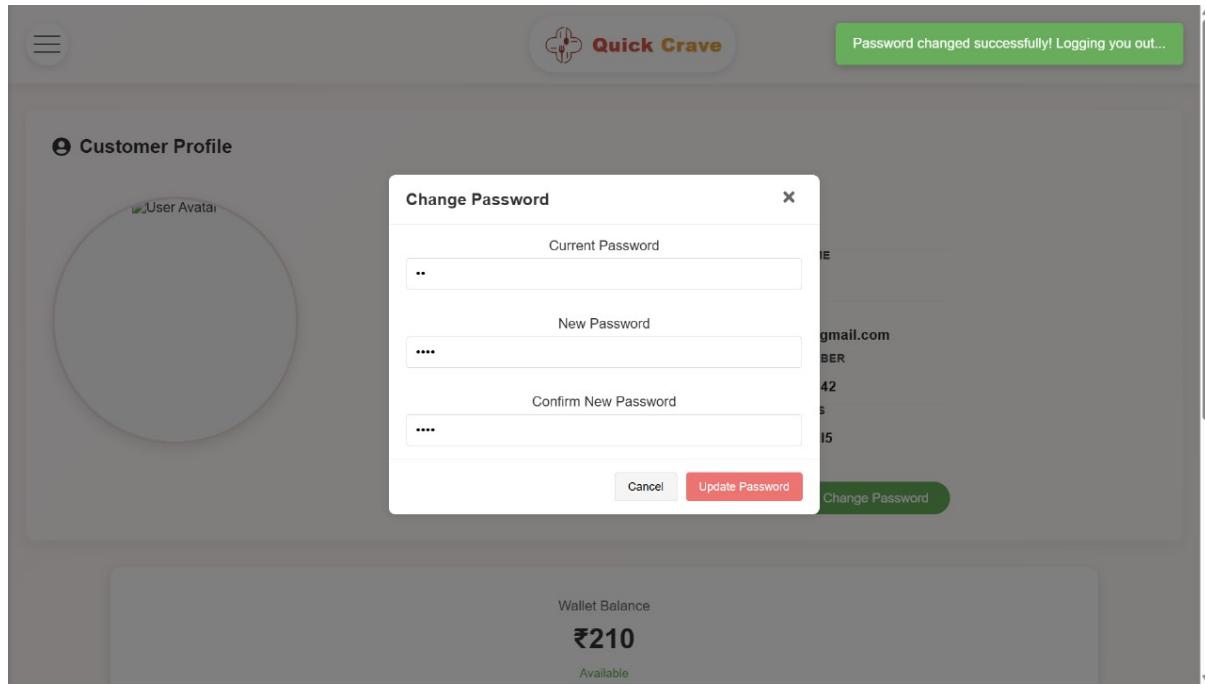


Figure 70: Customer Password Changed Successfully

Test Case 5: The system checks that the customer is able to add money to the wallet.

Result: When the customer presses the add to wallet button the system navigates the customer to "Add to wallet" page, and the customer enters the amount here.

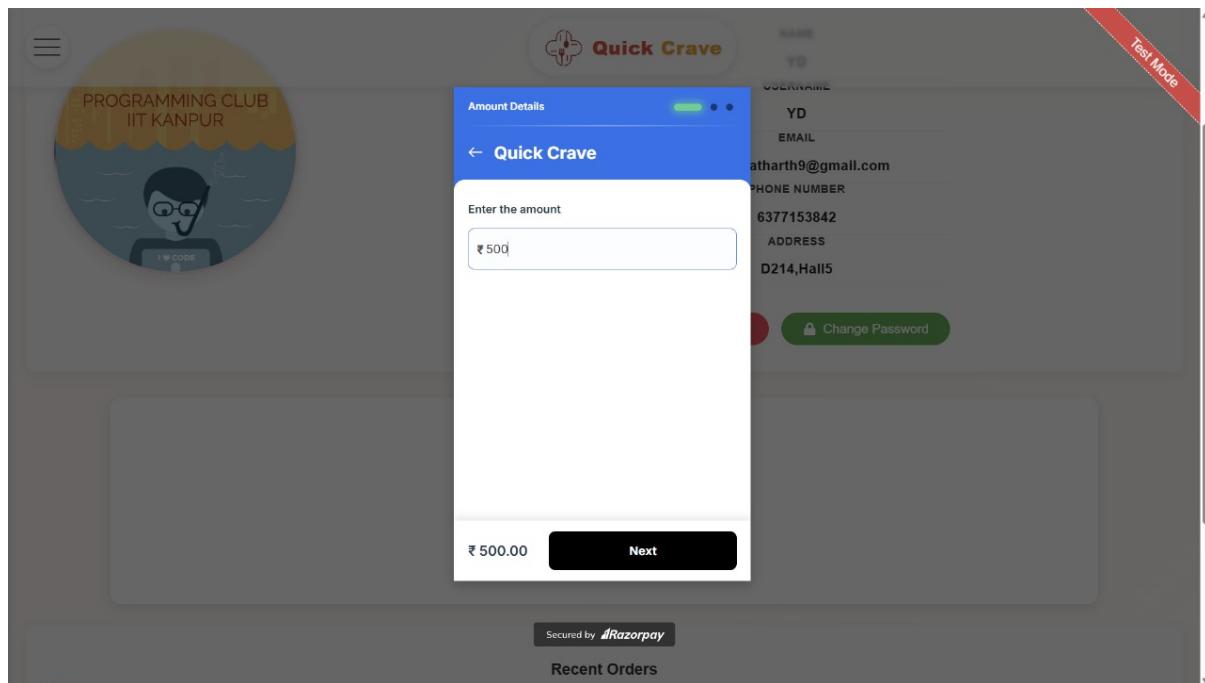


Figure 71: Customer Add to Wallet page

Test Case 6: The system checks that the money added to the wallet is updated in the customer's profile.

Result: When the customer enters the amount and presses the "Next" button, the system navigates the customer back to the profile page, and the wallet balance is updated.

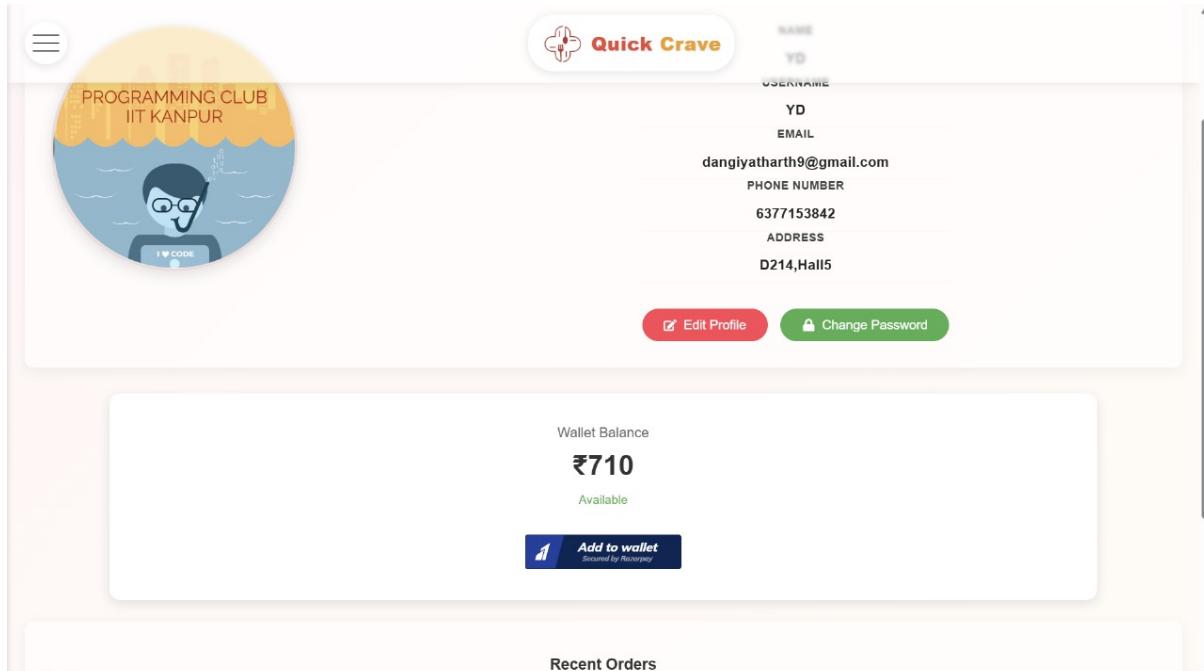


Figure 72: Customer wallet updated

Test Case 7: The system checks that the customer is able to update profile photo.

Result: When the customer presses over the profile photo he is directed to "File Explorer" where image is selected.

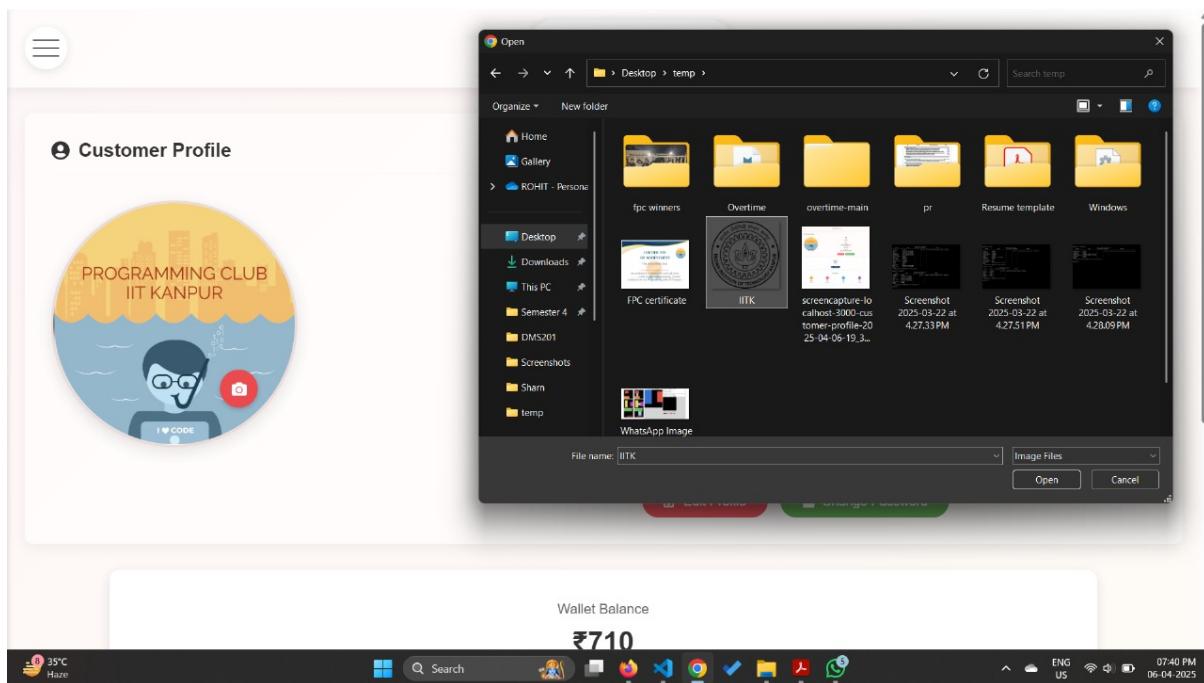


Figure 73: Customer update profile photo

Test case 8: The system checks that the profile photo is updated.

Result: When the customer selects the image from "File Explorer," he is navigated back to the customer profile page where the profile photo is updated.

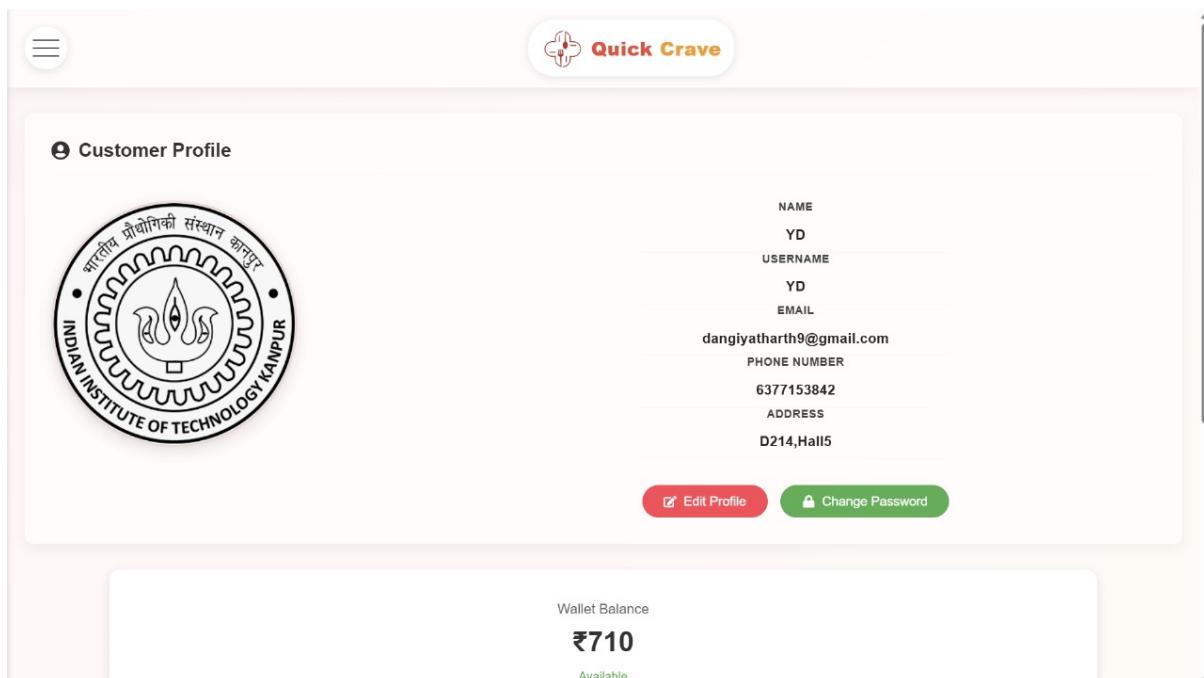


Figure 74: Customer profile photo updated

3.2.3 Cart

Module Details: This module is the integration of the complete Cart Page showing all the food items added from a certain canteen and the final submission of the order

Test Owner: Arihant Kumar

Test Date: 03-04-2025

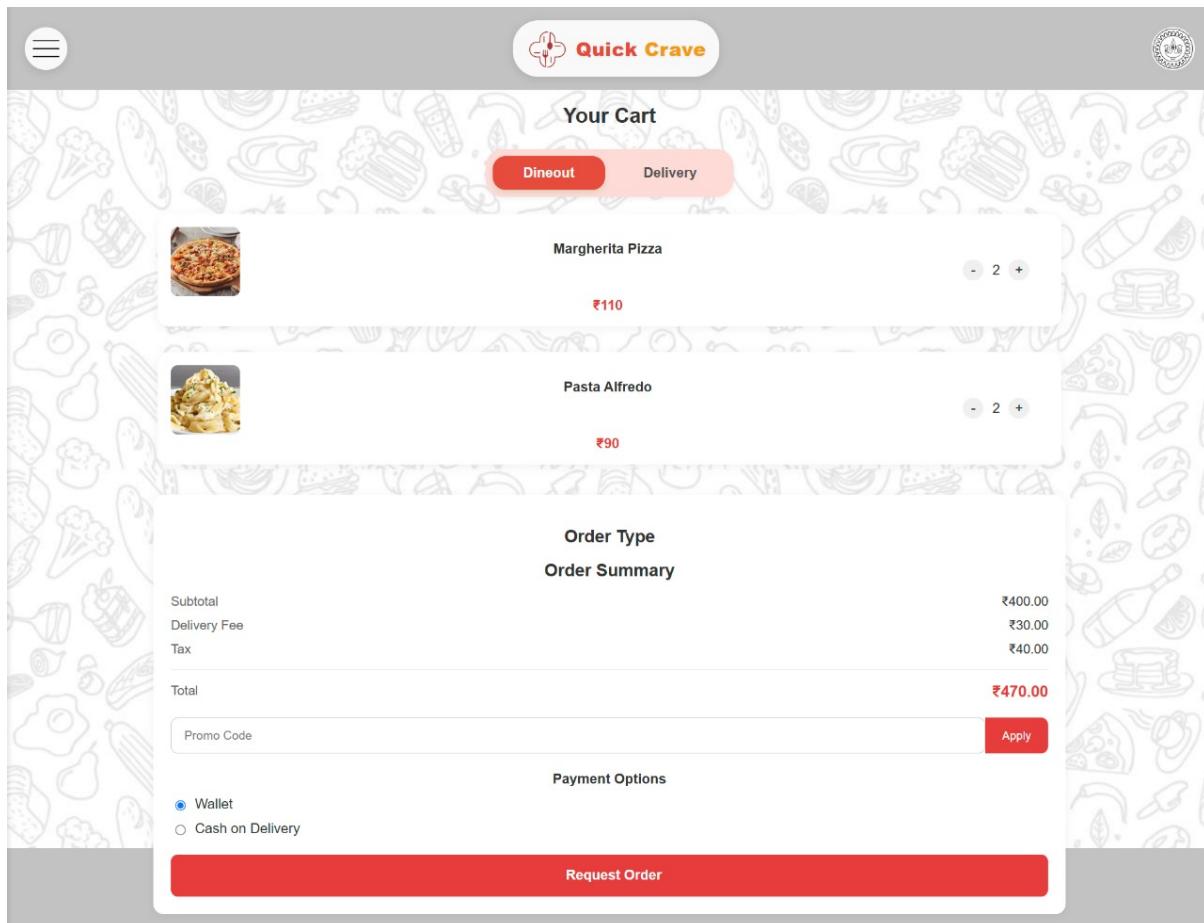


Figure 75: The Cart Page

Test Results: The module correctly loads all the details from the cart of the customer.

- **Test Case :** The module correctly checks if the request of order is being correctly placed.

Result : Once the items in the Cart are finalized, on clicking the **Request Order** button, the Order is successful sent to the Canteen Manager to be accepted. After requesting the order, the page shows a countdown which automatically closes after 5 minutes. However, if the auto accept of the canteen from which the dish is requested is on, then the order is automatically placed.

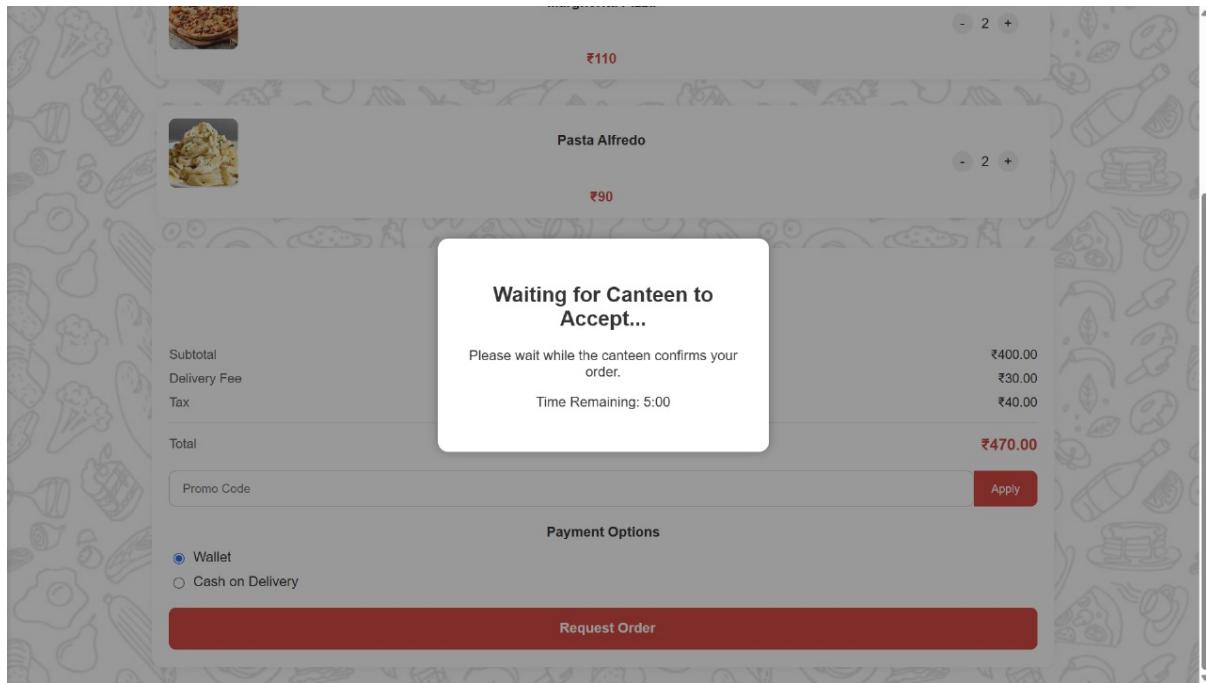


Figure 76: Successful Submission of Order

3.2.4 Sidebar

Module Details: This module is the integration of the complete Sidebar where different functionalities are available

Test Owner: Yatharth Dangi

Test Date: 23-03-2025

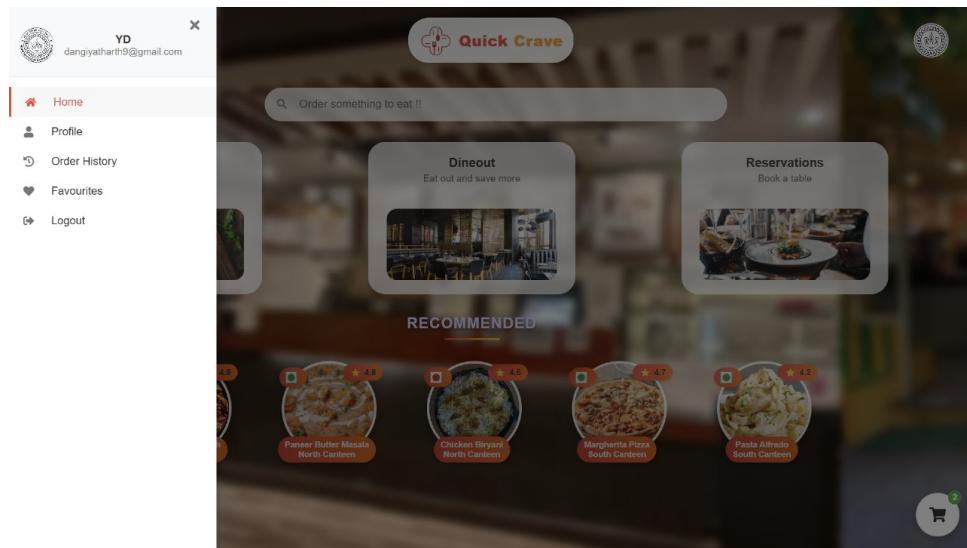


Figure 77: Sidebar

Test Results: The integrated module is working as expected fetching the user details successfully from backend.

- **Test Case :** The system checks the functioning of various options of Sidebar
Result : On opening the Sidebar, various options are available where the **Home** button takes back to the Home Page as demonstrated above; **Profile** option directs to the Profile as demonstrated above; **Order History** takes to the Order History Page shown below; **Favourites** takes to the Favourites Page shown below; **Log Out** logs out from the current account and takes back to the Login Page

The screenshot shows the 'Order History' page with a food-themed background. At the top, there are three tabs: 'Delivery' (selected), 'Dine-Out', and 'Reservations'. Below the tabs are filters for 'From' and 'To' dates, 'Status' (set to 'All Status'), and 'Sort By' (set to 'Date (Newest First)'). A red 'Apply Filters' button is located below the filters. The main section displays an order with the following details:

Order #ORD001		Completed	
Jun 25, 2023, 07:45 PM			
North Canteen ★ ★ ★ ★ 4.5			
Alfredo Pasta	x2	₹180	
Butter Naan	x2	₹70	
Total: ₹250		Reorder View Details	

Figure 78: Order History

3.2.5 Reservation

Module Details: This module is the integration of the complete Reservations page to reserve a place in a Canteen for a certain number of people at a certain date & time.

Test Owner: Parnika Mittal

Test Date: 01-04-2025

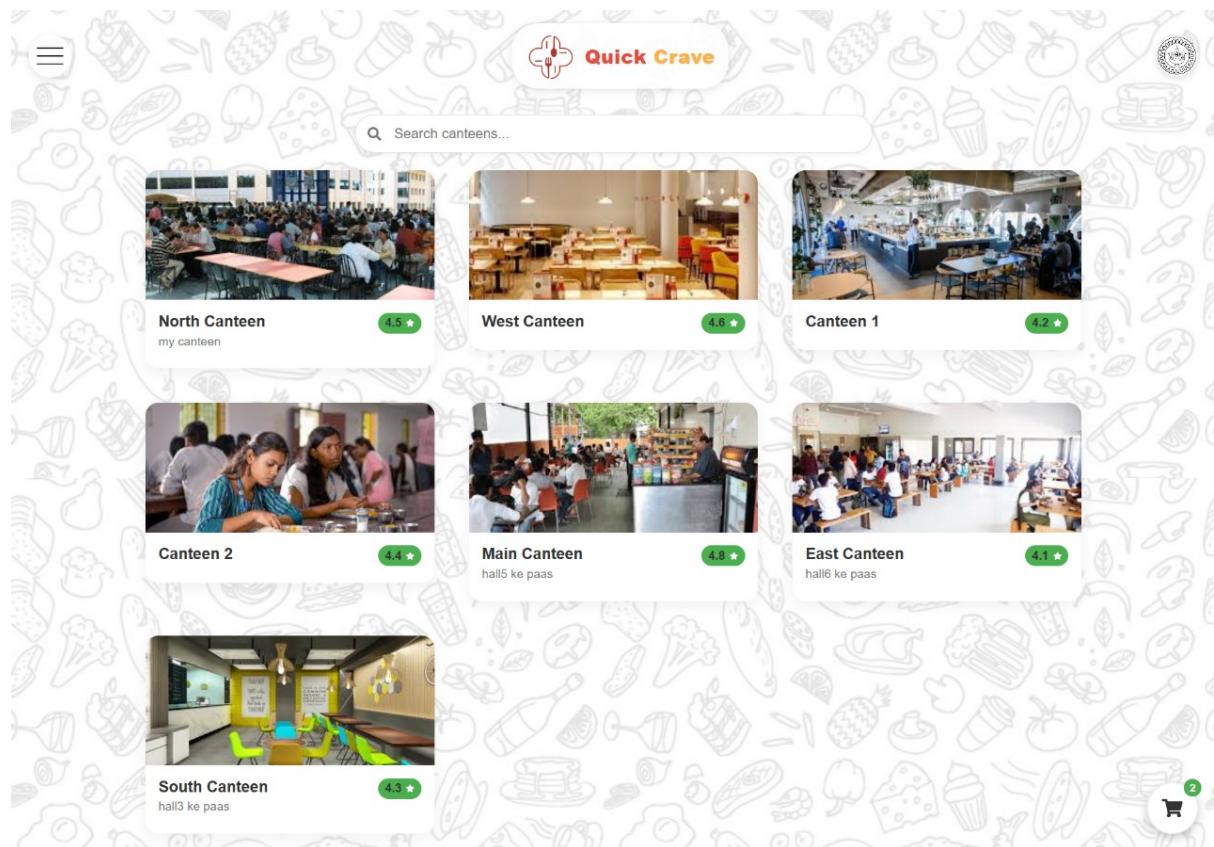


Figure 79: selecting Canteen for Reservation

Test Results: The integrated module is working as expected

- **Test Case 1 :** From the available list of canteens for reservation, after selecting the particular canteen it successfully directs to the Reservation Page of that canteen
- **Test Case 2 :** After filling in all the details for reservation and clicking the **Reserve Now** button, it successfully submits the Reservation to the Canteen Manager to be verified further.

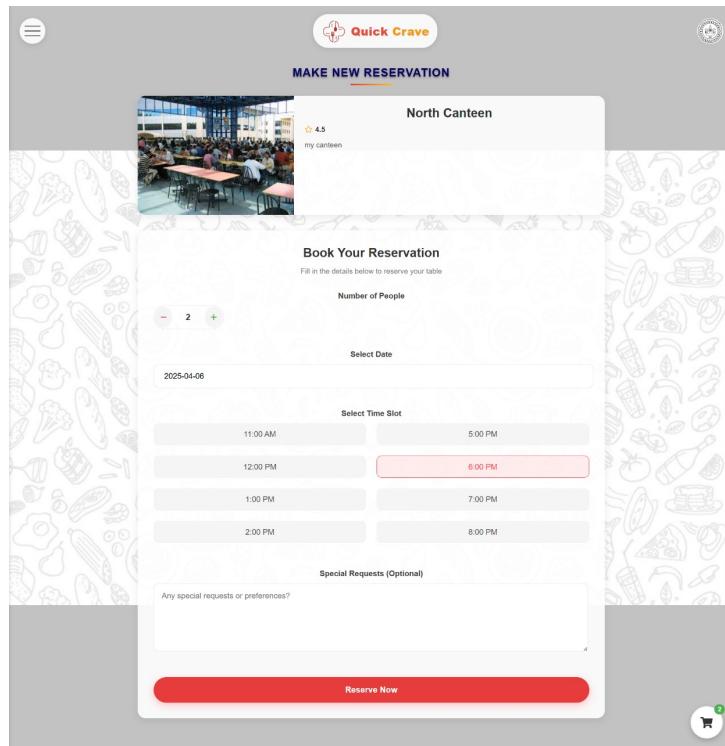


Figure 80: Filling in all the Details for Reservation

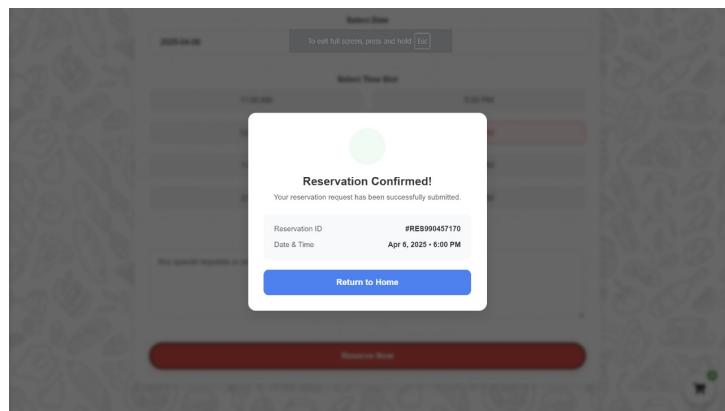


Figure 81: Successful Reservation Made

3.2.6 Food Delivery and Dineout

Module Details: This module is the integration of the complete Food Delivery and Dineout page to display all the food items across different canteens as well as all the canteens available for food delivery and dineout and to add food items to the Cart for such purposes.

Test Owner: Rohit Vijaykumar Soman

Test Date: 01/04/2025

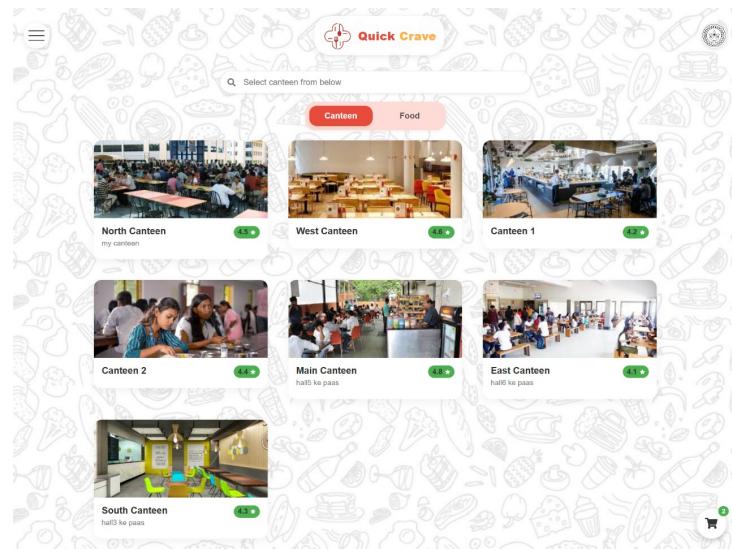


Figure 82: All the different canteen available

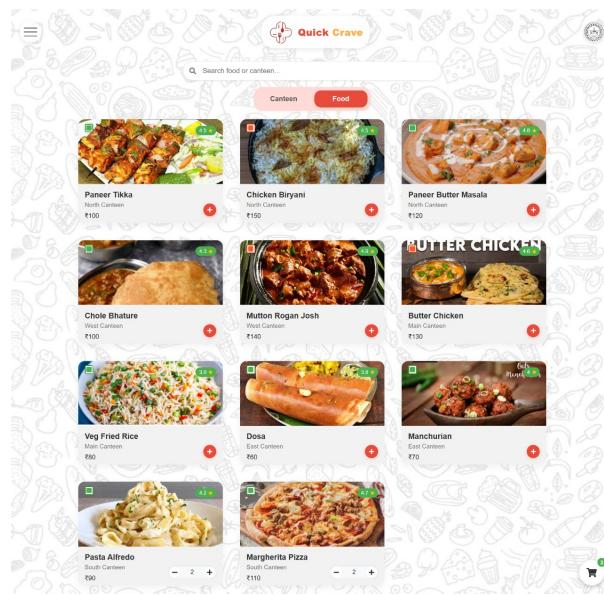


Figure 83: All the different food items available

Test Results: The integrated module is working as expected

- **Added to Empty Cart :** If the Cart is empty, any food item from any canteen is successfully added to the cart.
- **Adding to Non-Empty Cart :** If the Cart is not empty, and the new food item is from the same canteen as that of the items in that Cart, the food item is added successfully. If the new food item added is not from the same canteen, that food item is not added.

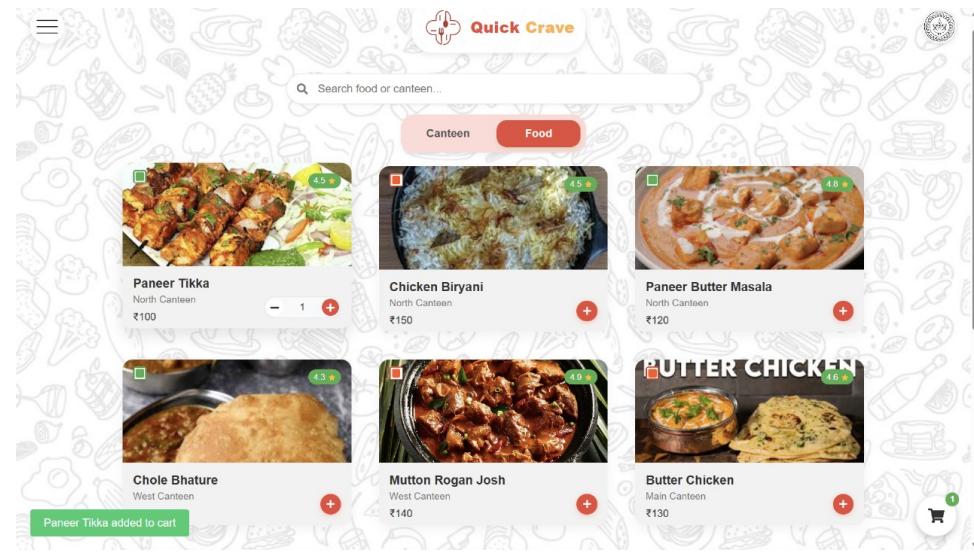


Figure 84: Selected Food Item Added

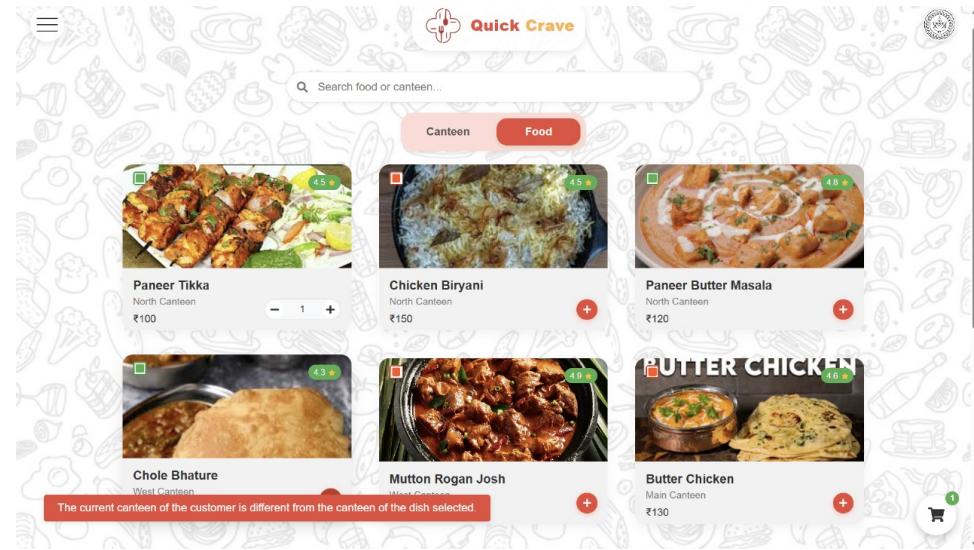


Figure 85: selected Food Item from Different Canteen not added

3.2.7 My Favourites

Module Details: This module is the integration of the complete My Favourites page to display all the favourite food items across different canteens and to display confirmation pop-up when any particular food item is to be removed by deselecting the Red Heart

Test Owner: Dnyanesh Hemendra Shinde

Test Date: 03/04/2025

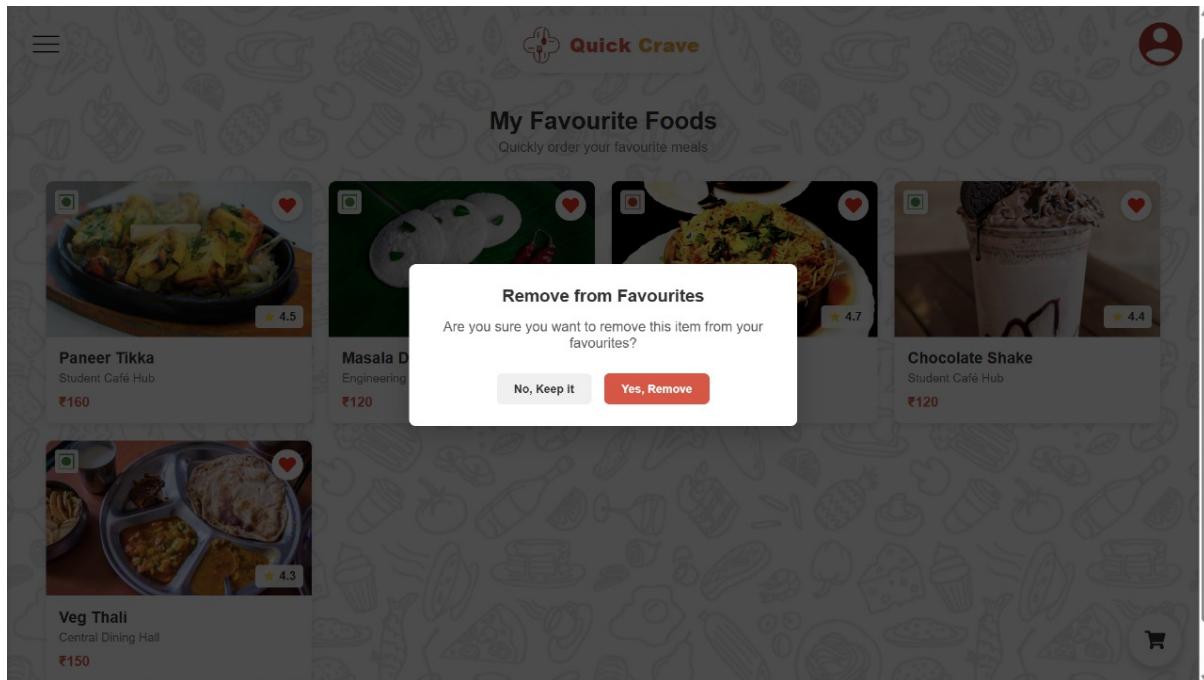


Figure 86: Removing Food Item from Favourites

Test Results: The integrated module is working as expected

- **Test Case 1 :** If the **Yes, Remove** option is selected on the pop-up, it removes the item from the Favourites page
- **Test Case 2 :** If the **No, Keep It** option is selected on the pop-up, it continues to keep it on the Favourites Page.

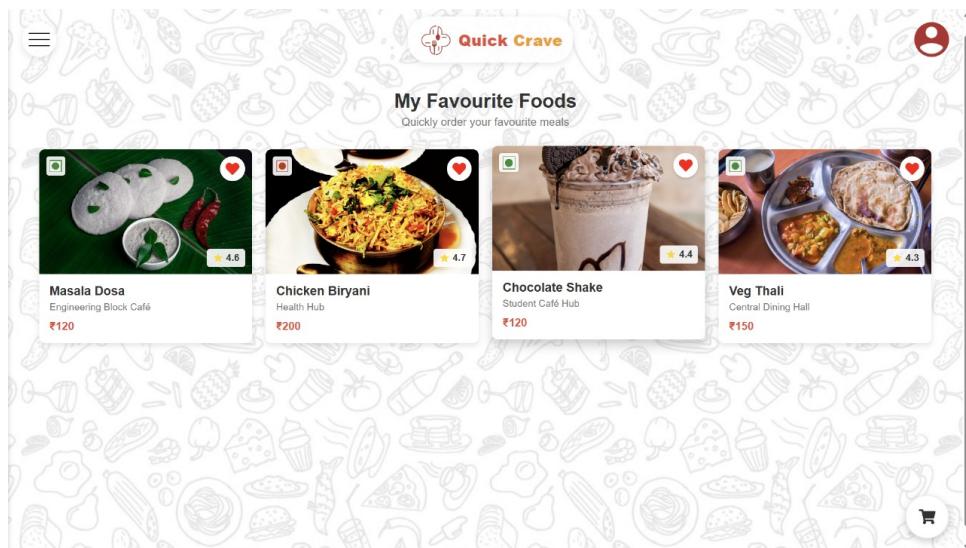


Figure 87: Selected Food Item Removed

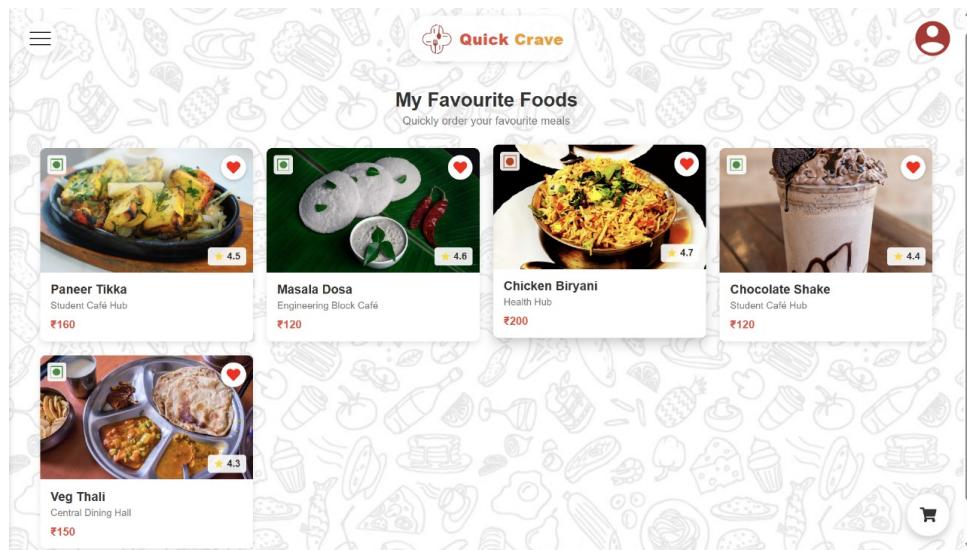


Figure 88: Selected Favourite Food Item Kept

3.3 Canteen Side

3.3.1 Canteen Home Page

Module Details: This module integrates all the relevant components to display the data on order queue, reservation queue , discount & coupons cards and the trending picks of the canteen

Test Owner: Parnika Mittal

Test Date: 03/04/2025

Test Results: The integrated module is working as expected

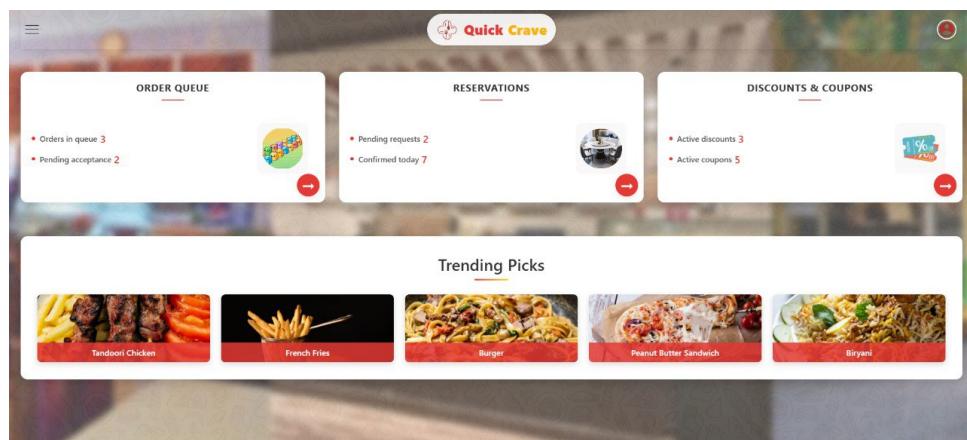


Figure 89: Canteen Home Page

3.3.2 Order Queue Page

Module Details: This module integrates all the relevant components to display and manage the order queue along with the current status of orders placed in the last 24 hours

Test Owner: Arijant Kumar

Test Date: 02/04/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** All Orders filter selected
Result: All orders correctly displayed
- **Test Case 2:** Waiting For Approval filter selected
Result: Orders waiting for approval correctly displayed
- **Test Case 3:** Waiting for payment filter selected
Result: Orders currently waiting for payment correctly displayed
- **Test Case 4:** Cooking filter selected
Result: Orders currently being cooked correctly displayed
- **Test Case 5:** Ready for delivery filter selected
Result: Orders ready for delivery correctly displayed
- **Test Case 6 :** Rejected/Failed filter selected
Result: Rejected/Failed Orders correctly displayed

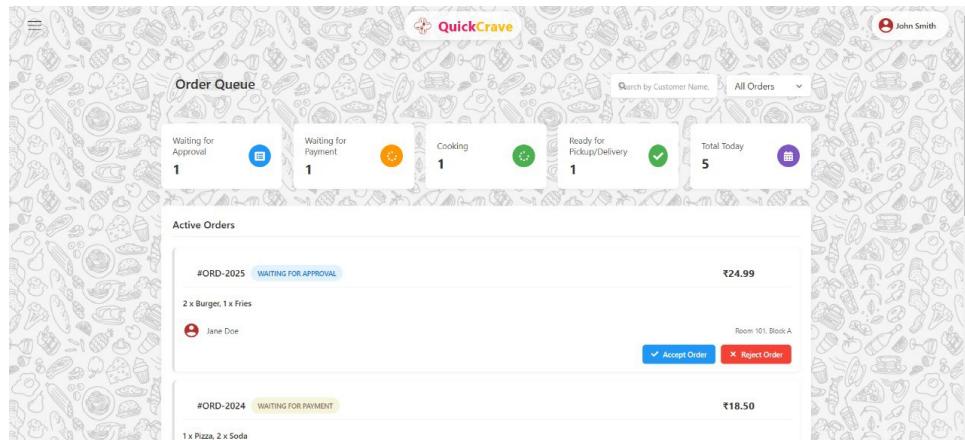


Figure 90: Order Queue

3.3.3 Reservation Queue Page

Module Details: This module integrates all the relevant components to display the reservations in the reservation queue and implemented filters

Test Owner: Suvid Goel

Test Date: 01/04/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** All Reservations filter selected
Result: All reservations correctly displayed
- **Test Case 2:** Pending Requests filter selected
Result: Requests rec correctly displayed
- **Test Case 3:** Accepted filter selected
Result: Reservations accepted in the past week correctly displayed
- **Test Case 4:** Rejected filter selected
Result: Reservations rejected in the past week correctly displayed

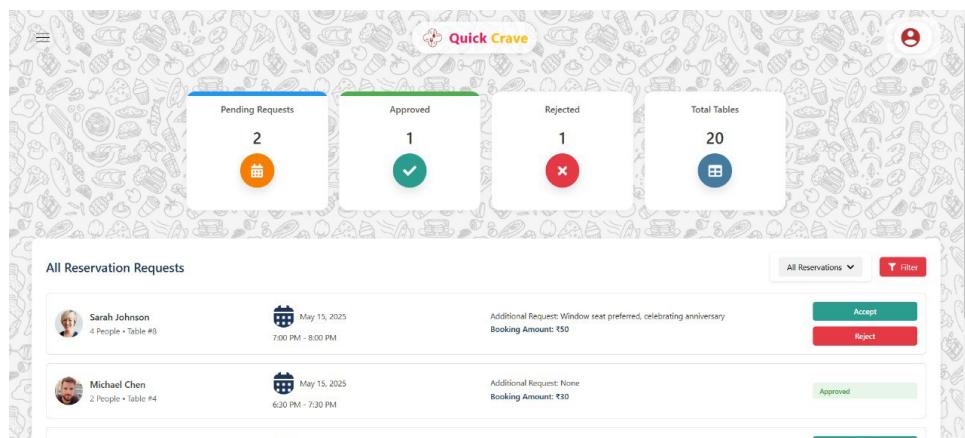


Figure 91: Reservation Queue

3.3.4 Discounts and Coupons Page

Module Details: This module integrates all the relevant components to display existing discounts and coupons and creating new discounts and coupons

Test Owner: Akshay Kamatam Reddy

Test Date: 04/04/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** Discount % is greater than 100

Result: Error prompt appears that discount % cannot be greater than 100

- **Test Case 2:** Discount % is between 0 and 100

Result: Discount is applied to dish

- **Test Case 3:** Discount value in coupon is greater than or equal to Minimum Order Value

Result: Error prompt appears

- **Test Case 4:** Discount value in coupon is less than Minimum Order Value

Result: Coupon is created

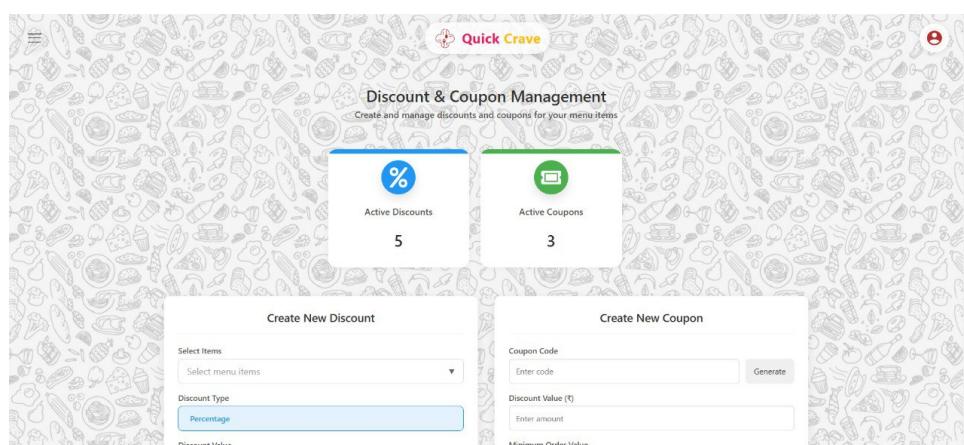


Figure 92: Discounts Page

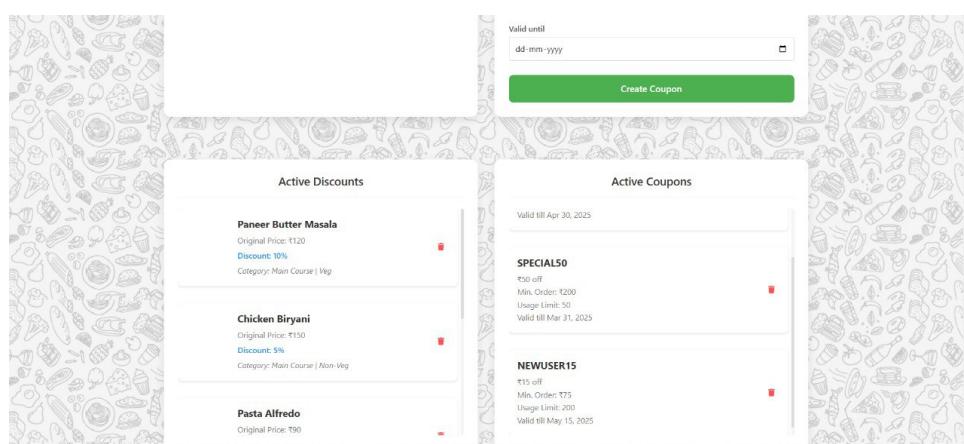


Figure 93: Discounts Page - 2

3.3.5 Menu Management Page

Module Details: This module integrates all the relevant components to display and modify the menu for the canteen

Test Owner: Kshitij Gupta

Test Date: 04/04/2025

Test Results: The integrated module is working as expected

- **Test Case 1:** Adding a new dish item on the Menu

Result: After filling all the relevant details the dish is successfully added

- **Test Case 2:** Discount % is greater than 100

Result: Error prompt appears that discount % cannot be greater than 100

- **Test Case 3:** Discount % is between 0 and 100

Result: Discount is applied to dish

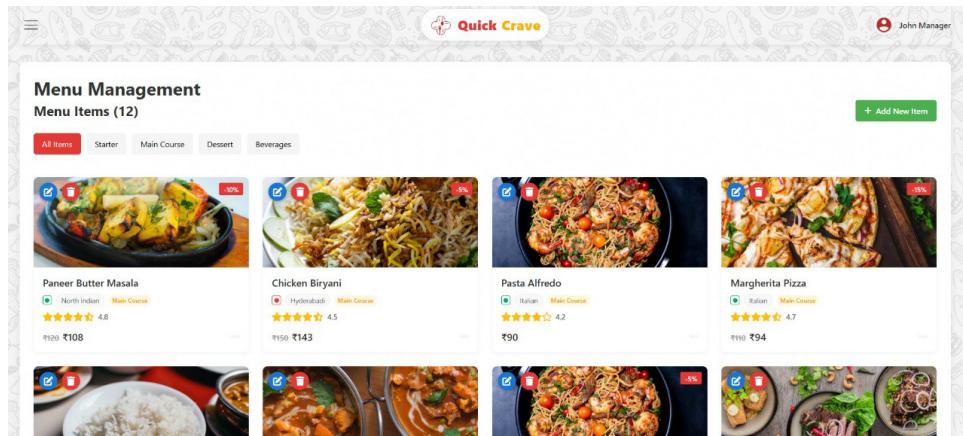


Figure 94: Menu Management Page

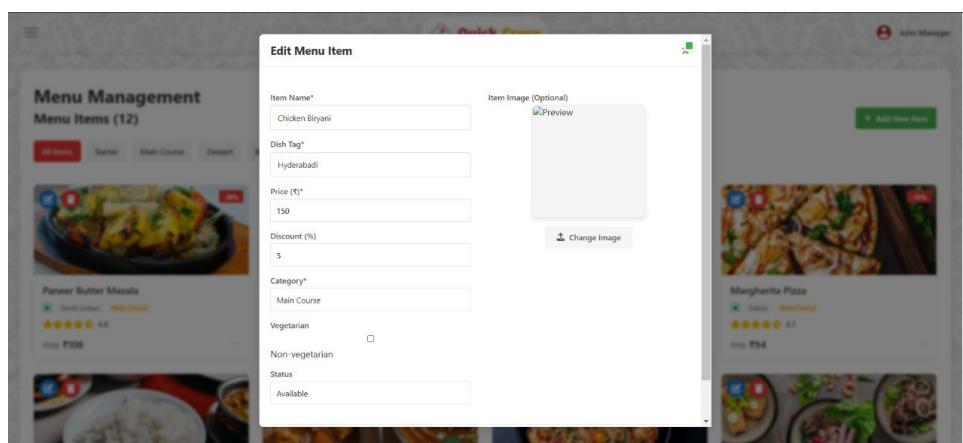


Figure 95: Adding Dish Menu Management Page

3.3.6 Profile Page

Module Details: This module integrates all the relevant components to display the orders in the order queue and implemented filters

Test Owner: Dnyanesh Hemendra Shinde

Test Date: 29/03/2025

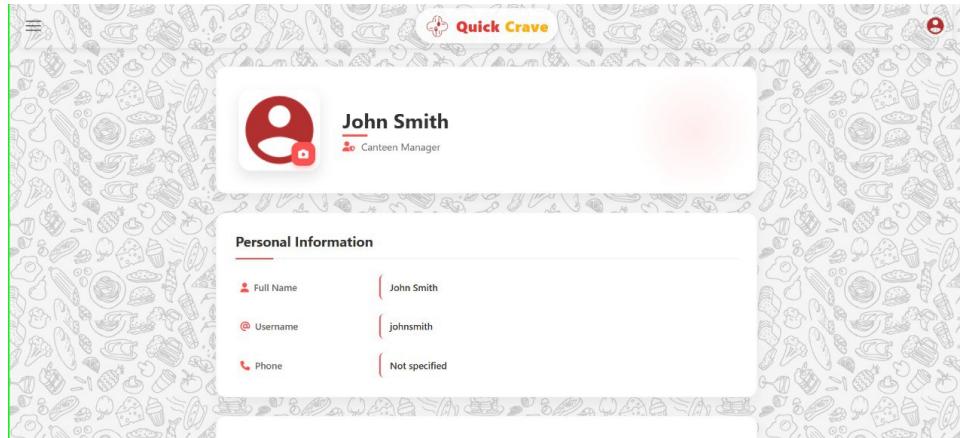


Figure 96: Canteen Manager Profile Page

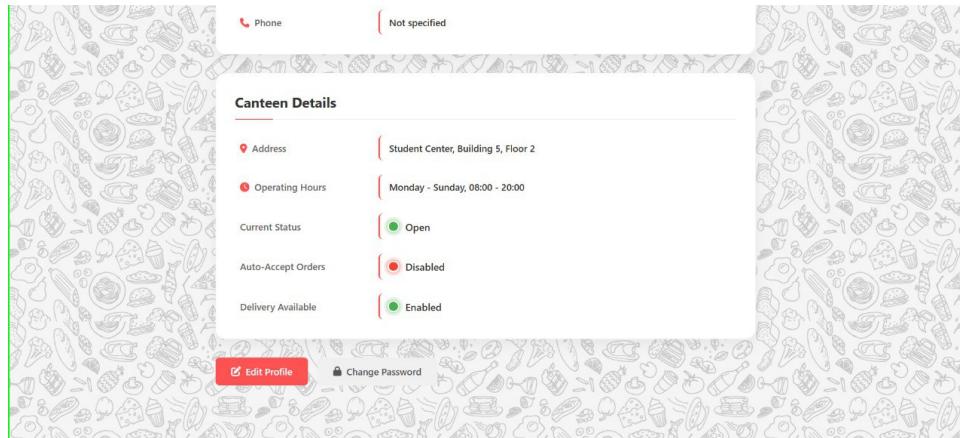


Figure 97: Canteen Manager Profile Page

Test Results: The integrated module is working as expected

- **Test Case 1:** If new username entered in edit profile is already taken
Result: Error prompt appears that username is already taken
- **Test Case 2:** If password does not follow security standards
Result: Error prompt appears displaying the security standard not followed

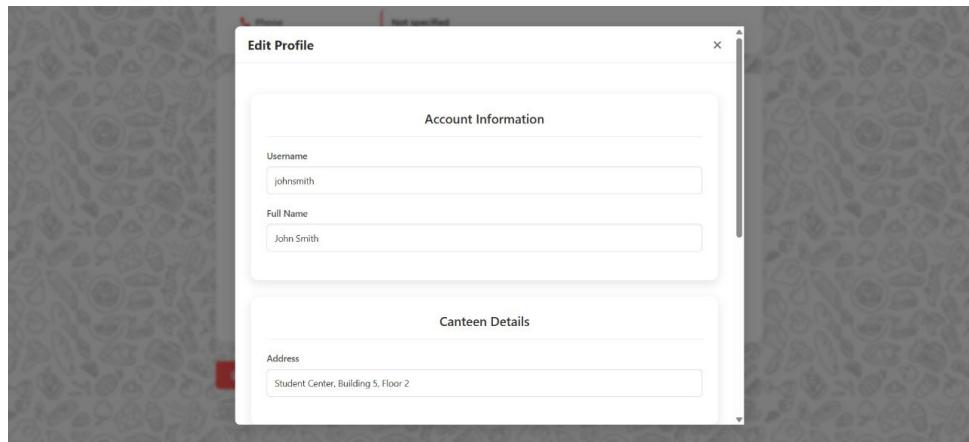


Figure 98: Edit Canteen Profile Page

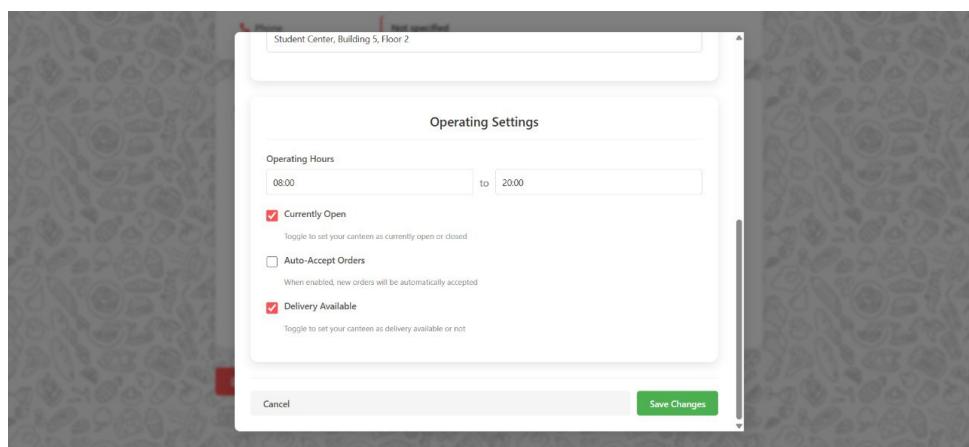


Figure 99: Edit Canteen Profile Page

3.3.7 Statistics Page

Module Details: This module integrates all the relevant components to display the statistics and implemented filters

Test Owner: Yashaswi Raj

Test Date: 03/04/2025

Test Results: The integrated module is working as expected

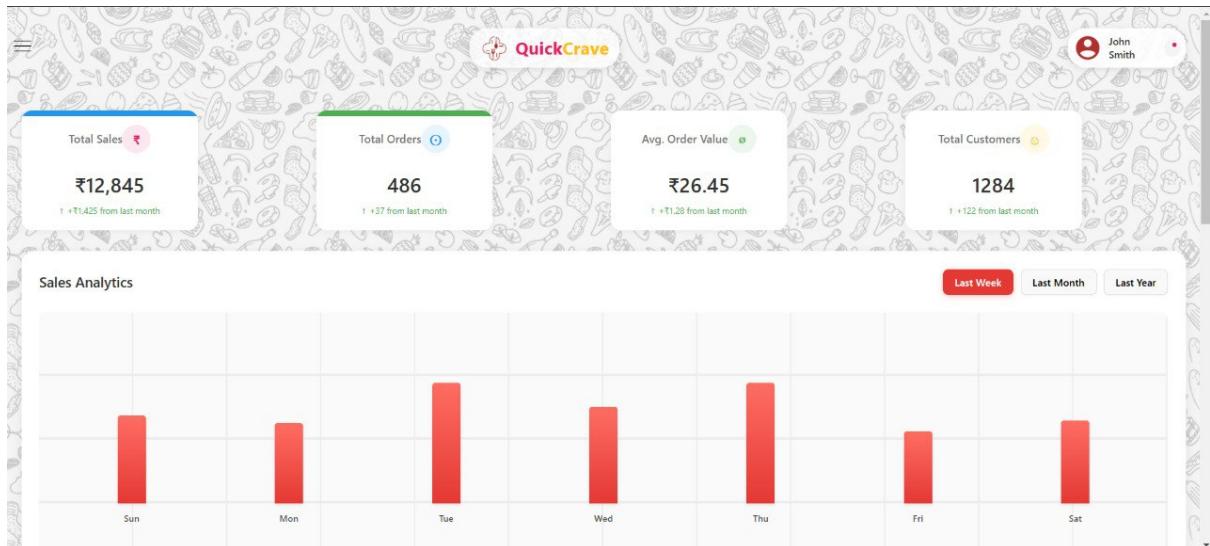


Figure 100: Canteen Manager Statistics Page



Figure 101: Canteen Manager Statistics Page

4 System Testing

4.1 Load testing

4.1.1 customer/customer-search-canteen

```
hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhlhdCI6MTc0Mzk1ODYwNywiZXh
._It7Pjc9AHKs0IiX_dSkYsXu-fb14e06i7IgYMS-1x8" \
-d '{"canteen_name": "North Canteen"}' \
http://localhost:4000/customer/customer-search-canteen

Summary:
Total:          0.3585 secs
Slowest:        0.1893 secs
Fastest:        0.0014 secs
Average:        0.0153 secs
Requests/sec:  5579.1488

Total data:    418000 bytes
Size/request: 209 bytes

Response time histogram:
0.001 [1] |
0.020 [1901] ████████████████████████████████████ |
0.039 [24] ■  |
0.058 [7]   |
0.077 [8]   |
0.095 [9]   |
0.114 [9]   |
0.133 [10]  |
0.152 [11]  |
0.171 [10]  |
0.189 [10]  |

Latency distribution:
10% in 0.0069 secs
25% in 0.0087 secs
50% in 0.0112 secs
75% in 0.0148 secs
90% in 0.0165 secs
95% in 0.0196 secs
99% in 0.1541 secs

Details (average, fastest, slowest):
DNS+dialup:  0.0004 secs, 0.0014 secs, 0.1893 secs
DNS-lookup:   0.0002 secs, 0.0000 secs, 0.0110 secs
req write:    0.0000 secs, 0.0000 secs, 0.0061 secs
resp wait:   0.0150 secs, 0.0014 secs, 0.1768 secs
resp read:   0.0000 secs, 0.0000 secs, 0.0007 secs
```

```
Status code distribution:  
[200] 2000 responses
```

4.1.2 customer/customer-search-dish

```
hey -n 2000 -c 100 -m POST \  
-H "Content-Type:application/json" \  
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhdCI6MTc0Mzk1ODYwNywiZXh  
.It7Pjc9AHKs0IiX_dSkYsXu-fb14e06i7IgYMS-1x8" \  
-d '{"dishNameToSearch": "paneer"}' \  
http://localhost:4000/customer/customer-search-dish

Summary:  
Total: 0.4378 secs  
Slowest: 0.1093 secs  
Fastest: 0.0134 secs  
Average: 0.0215 secs  
Requests/sec: 4568.6700

Total data: 956000 bytes  
Size/request: 478 bytes

Response time histogram:  
0.013 [1] |  
0.023 [1757] ██████████████████████████████████████████████████████████████████████████ |  
0.033 [142] ████ |  
0.042 [0]  
0.052 [0]  
0.061 [0]  
0.071 [0]  
0.081 [0]  
0.090 [30] ■ |  
0.100 [37] ■ |  
0.109 [33] ■ |
```



```
Latency distribution:  
10% in 0.0145 secs  
25% in 0.0154 secs  
50% in 0.0168 secs  
75% in 0.0201 secs  
90% in 0.0238 secs  
95% in 0.0819 secs  
99% in 0.1040 secs
```



```
Details (average, fastest, slowest):  
DNS+dialup: 0.0003 secs, 0.0134 secs, 0.1093 secs  
DNS-lookup: 0.0001 secs, 0.0000 secs, 0.0026 secs
```

```
req write: 0.0000 secs, 0.0000 secs, 0.0013 secs
resp wait: 0.0210 secs, 0.0134 secs, 0.0995 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0010 secs
```

```
Status code distribution:
[200] 2000 responses
```

4.1.3 /customer/customer-fetch-reservations

```
hey -n 2000 -c 100 -m POST \
-H "Content-Type: application/json" \
-H "Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhlhdCI6MTc0Mzk1ODYwNywiZXh
._It7Pjc9AHKs0IiX_dSKYsXu-fb14e06i7IgYMS-1x8" \
http://localhost:4000/customer/customer-fetch-reservations
```

Summary:

```
Total: 0.4234 secs
Slowest: 0.0937 secs
Fastest: 0.0036 secs
Average: 0.0209 secs
Requests/sec: 4723.5470
```

```
Total data: 38000 bytes
Size/request: 19 bytes
```

Response time histogram:



Latency distribution:

```
10% in 0.0140 secs
25% in 0.0155 secs
50% in 0.0177 secs
75% in 0.0199 secs
90% in 0.0265 secs
95% in 0.0555 secs
99% in 0.0882 secs
```

Details (average, fastest, slowest):

```
DNS+dialup: 0.0004 secs, 0.0036 secs, 0.0937 secs
DNS-lookup: 0.0002 secs, 0.0000 secs, 0.0047 secs
req write: 0.0000 secs, 0.0000 secs, 0.0018 secs
resp wait: 0.0204 secs, 0.0036 secs, 0.0817 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0009 secs
```

```
Status code distribution:
[200] 2000 responses
```

4.1.4 /customer/customer-add-dish

```
hey -n 2000 -c 100 -m POST \
-H "Content-Type: application/json" \
-H "Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciiIsImhdCI6MTc0Mzk1ODYwNywiZXh
._It7Pjc9AHKs0IiX_dSkYsXu-fb14e06i7IgYMS-1x8" \
-d '{"dish_id":2}' \
http://localhost:4000/customer/customer-add-dish

Summary:
Total: 0.6073 secs
Slowest: 0.1299 secs
Fastest: 0.0079 secs
Average: 0.0302 secs
Requests/sec: 3293.0897

Total data: 215549 bytes
Size/request: 107 bytes

Response time histogram:
0.008 [1] |
0.020 [393] ██████████ | 
0.032 [1209] ████████████████████ | 
0.045 [177] █████ | 
0.057 [71] ███ | 
0.069 [49] ███ | 
0.081 [0] | 
0.093 [0] | 
0.106 [17] █ | 
0.118 [55] ███ | 
0.130 [28] █ | 

Latency distribution:
10% in 0.0193 secs
25% in 0.0204 secs
50% in 0.0238 secs
75% in 0.0297 secs
90% in 0.0472 secs
95% in 0.1028 secs
```

```
99% in 0.1183 secs

Details (average, fastest, slowest):
DNS+dialup: 0.0009 secs, 0.0079 secs, 0.1299 secs
DNS-lookup: 0.0007 secs, 0.0000 secs, 0.0147 secs
req write: 0.0000 secs, 0.0000 secs, 0.0011 secs
resp wait: 0.0291 secs, 0.0079 secs, 0.1075 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0009 secs

Status code distribution:
[200] 1900 responses
[201] 100 responses
```

4.1.5 /customer/customer-remove-dish

```
hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VtIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhlhdCI6MTc0Mzk1ODYwNywiZXh
._It7Pjc9AHKs0IiX_dSkYsXu-fb14e06i7IgYMS-1x8" \
-d '{"dish_id":2}' \
http://localhost:4000/customer/customer-remove-dish

Summary:
  Total:      0.6347 secs
  Slowest:    0.1409 secs
  Fastest:    0.0059 secs
  Average:    0.0316 secs
  Requests/sec: 3150.8933

  Total data: 208000 bytes
  Size/request: 104 bytes

Response time histogram:
 0.006 [1]      |
 0.019 [388]  ██████████ | 
 0.033 [1165] ████████████████████ | 
 0.046 [225] ████████ | 
 0.060 [13]   |
 0.073 [69]  ███ | 
 0.087 [28]  █ | 
 0.100 [32]  █ | 
 0.114 [39]  █ | 
 0.127 [2]   |
 0.141 [38]  █ | 

Latency distribution:
```

```

10% in 0.0187 secs
25% in 0.0197 secs
50% in 0.0236 secs
75% in 0.0315 secs
90% in 0.0650 secs
95% in 0.0923 secs
99% in 0.1342 secs

Details (average, fastest, slowest):
DNS+dialup: 0.0010 secs, 0.0059 secs, 0.1409 secs
DNS-lookup: 0.0008 secs, 0.0000 secs, 0.0176 secs
req write: 0.0000 secs, 0.0000 secs, 0.0013 secs
resp wait: 0.0303 secs, 0.0059 secs, 0.1155 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0039 secs

Status code distribution:
[200] 2000 responses

```

4.1.6 /customer/customer-view-order-history

```

hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhlhdCI6MTc0Mzk1ODYwNywiZXh
._It7Pjc9AHKs0Iix_dSKYsXu-fb14e06i7igYMS-1x8" \
http://localhost:4000/customer/customer-view-order-history

Summary:
Total: 0.3258 secs
Slowest: 0.0645 secs
Fastest: 0.0015 secs
Average: 0.0160 secs
Requests/sec: 6139.1492

Total data: 110000 bytes
Size/request: 55 bytes

Response time histogram:
0.002 [1] |
0.008 [30] ■ |
0.014 [1369] ████████████████████████████ |
0.020 [285] ████████ |
0.027 [126] █████ | |
0.033 [89] █████ | |
0.039 [0] |
0.046 [15] |
0.052 [21] ■ |
0.058 [39] ■ |
0.064 [25] ■ |

```

```
Latency distribution:
```

```
10% in 0.0106 secs  
25% in 0.0117 secs  
50% in 0.0129 secs  
75% in 0.0147 secs  
90% in 0.0247 secs  
95% in 0.0396 secs  
99% in 0.0619 secs
```

```
Details (average, fastest, slowest):
```

```
DNS+dialup: 0.0002 secs, 0.0015 secs, 0.0645 secs  
DNS-lookup: 0.0001 secs, 0.0000 secs, 0.0024 secs  
req write: 0.0000 secs, 0.0000 secs, 0.0021 secs  
resp wait: 0.0157 secs, 0.0015 secs, 0.0573 secs  
resp read: 0.0000 secs, 0.0000 secs, 0.0009 secs
```

```
Status code distribution:
```

```
[200] 2000 responses
```

4.1.7 /customer/customer-view-favourites

```
hey -n 2000 -c 100 -m POST \  
-H "Content-Type:application/json" \  
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhdCI6MTc0Mzk1ODYwNywiZXh  
.It7Pjc9AHKs0IiX_dSkYsXu-fb14e06i7IgYMS-1x8" \  
http://localhost:4000/customer/customer-view-favourites
```

```
Summary:
```

```
Total: 0.3882 secs  
Slowest: 0.1590 secs  
Fastest: 0.0007 secs  
Average: 0.0184 secs  
Requests/sec: 5151.8619
```

```
Total data: 52000 bytes  
Size/request: 26 bytes
```

```
Response time histogram:
```



```

0.159 [9]      |

Latency distribution:
10% in 0.0095 secs
25% in 0.0139 secs
50% in 0.0159 secs
75% in 0.0185 secs
90% in 0.0269 secs
95% in 0.0296 secs
99% in 0.1137 secs

Details (average, fastest, slowest):
DNS+dialup: 0.0003 secs, 0.0007 secs, 0.1590 secs
DNS-lookup: 0.0001 secs, 0.0000 secs, 0.0031 secs
req write: 0.0000 secs, 0.0000 secs, 0.0010 secs
resp wait: 0.0179 secs, 0.0007 secs, 0.1498 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0005 secs

Status code distribution:
[200] 2000 responses

```

4.1.8 /customer/customer-wallet

```

hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lciIsImhdCI6MTc0Mzk2MjQzMiwZXh
.L9c90JZqCzLeNYQRu61Nt6yeFHRKBDICuYKLJ5NT_E_k" \
http://localhost:4000/customer/customer-wallet

Summary:
Total: 1.0402 secs
Slowest: 0.2956 secs
Fastest: 0.0038 secs
Average: 0.0499 secs
Requests/sec: 1922.6818

Total data: 158000 bytes
Size/request: 79 bytes

Response time histogram:
0.004 [1]      |
0.033 [259]    |
0.062 [1222]   |
0.091 [404]    |
0.121 [68]     |
0.150 [23]     |
0.179 [4]      |
0.208 [1]      |

```

```

0.237 [4]      |
0.266 [7]      |
0.296 [7]      |

Latency distribution:
10% in 0.0324 secs
25% in 0.0344 secs
50% in 0.0403 secs
75% in 0.0635 secs
90% in 0.0718 secs
95% in 0.0992 secs
99% in 0.1599 secs

Details (average, fastest, slowest):
DNS+dialup: 0.0003 secs, 0.0038 secs, 0.2956 secs
DNS-lookup: 0.0001 secs, 0.0000 secs, 0.0037 secs
req write: 0.0000 secs, 0.0000 secs, 0.0008 secs
resp wait: 0.0496 secs, 0.0038 secs, 0.2879 secs
resp read: 0.0000 secs, 0.0000 secs, 0.0010 secs

Status code distribution:
[200] 2000 responses

```

4.1.9 /customer/customer-view-profile

```

hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. \
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUI0iJjdXN0b21lciIsImhdCI6MTc0Mzk2MjQzMiwZXh \
.L9c90JZqCzLeNYQRu61Nt6yeFHRKBDICuYKLJ5NT_E_k" \
http://localhost:4000/customer/customer-view-profile

Summary:
Total: 8.3892 secs
Slowest: 1.7591 secs
Fastest: 0.0357 secs
Average: 0.4030 secs
Requests/sec: 238.4019

Total data: 794000 bytes
Size/request: 397 bytes

Response time histogram:
0.036 [1]      |
0.208 [156]    |
0.380 [466]    |
0.553 [1315]   |
0.725 [7]       |
0.897 [10]      |

```

```
1.070 [7] |  
1.242 [10] |  
1.414 [9] |  
1.587 [8] |  
1.759 [11] |
```

Latency distribution:

```
10% in 0.2326 secs  
25% in 0.3762 secs  
50% in 0.3915 secs  
75% in 0.4422 secs  
90% in 0.4668 secs  
95% in 0.4803 secs  
99% in 1.4128 secs
```

Details (average, fastest, slowest):

```
DNS+dialup: 0.0002 secs, 0.0357 secs, 1.7591 secs  
DNS-lookup: 0.0001 secs, 0.0000 secs, 0.0031 secs  
req write: 0.0000 secs, 0.0000 secs, 0.0011 secs  
resp wait: 0.4026 secs, 0.0357 secs, 1.7505 secs  
resp read: 0.0000 secs, 0.0000 secs, 0.0006 secs
```

Status code distribution:

```
[200] 2000 responses
```

4.1.10 /customer/customer-fetch-top5

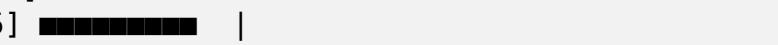
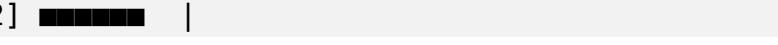
```
hey -n 2000 -c 100 -m POST \  
-H "Content-Type: application/json" \  
-H "Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJ1c2VyX2lkIjoyNiwidXNlclR5cGUiOiJjdXN0b21lcIIsImhdCI6MTc0Mzk2MjQzMiwZXh  
.L9c90JZqCzLeNYQRu61Nt6yeFhrKBDICuYKLJ5NTE_k" \  
http://localhost:4000/customer/customer-fetch-top5
```

Summary:

```
Total: 0.7536 secs  
Slowest: 0.1182 secs  
Fastest: 0.0229 secs  
Average: 0.0372 secs  
Requests/sec: 2654.0399
```

```
Total data: 2182000 bytes  
Size/request: 1091 bytes
```

Response time histogram:

```
0.023 [1] |  
0.032 [1257]  |  
0.042 [285]  |  
0.052 [192]  |
```

```
0.061 [66] ■ |  
0.071 [15] |  
0.080 [56] ■ |  
0.090 [49] ■ |  
0.099 [20] ■ |  
0.109 [30] ■ |  
0.118 [29] ■ |
```

Latency distribution:

```
10% in 0.0250 secs  
25% in 0.0261 secs  
50% in 0.0288 secs  
75% in 0.0381 secs  
90% in 0.0603 secs  
95% in 0.0841 secs  
99% in 0.1116 secs
```

Details (average, fastest, slowest):

```
DNS+dialup: 0.0003 secs, 0.0229 secs, 0.1182 secs  
DNS-lookup: 0.0002 secs, 0.0000 secs, 0.0038 secs  
req write: 0.0000 secs, 0.0000 secs, 0.0008 secs  
resp wait: 0.0368 secs, 0.0229 secs, 0.1094 secs  
resp read: 0.0000 secs, 0.0000 secs, 0.0004 secs
```

Status code distribution:

```
[200] 2000 responses
```

4.1.11 /customer/customer-fetch-dishes

```
hey -n 2000 -c 100 -m POST \  
-H "Content-Type: application/json" \  
-H "Cookie: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUIoIJjdXN0b21lciIsImhdCI6MTc0Mzk2MjQzMiwZXh  
.L9c90JZqCzLeNYQRu61Nt6yeFHRKBDICuYKLJ5NTE_k" \  
http://localhost:4000/customer/customer-fetch-dishes
```

Summary:

```
Total: 0.4597 secs  
Slowest: 0.0728 secs  
Fastest: 0.0073 secs  
Average: 0.0228 secs  
Requests/sec: 4350.8618
```

```
Total data: 4722000 bytes  
Size/request: 2361 bytes
```

Response time histogram:

```
0.007 [1] |  
0.014 [70] ■ |
```

0.020	[1219]	
0.027	[235]	
0.033	[233]	
0.040	[142]	
0.047	[0]	
0.053	[10]	
0.060	[29]	
0.066	[40]	
0.073	[21]	

Latency distribution:

```
10% in 0.0148 secs  
25% in 0.0165 secs  
50% in 0.0184 secs  
75% in 0.0250 secs  
90% in 0.0348 secs  
95% in 0.0507 secs  
99% in 0.0692 secs
```

Details (average, fastest, slowest):

DNS+dialup:	0.0003	secs,	0.0073	secs,	0.0728	secs
DNS-lookup:	0.0001	secs,	0.0000	secs,	0.0035	secs
req write:	0.0000	secs,	0.0000	secs,	0.0009	secs
resp wait:	0.0223	secs,	0.0073	secs,	0.0634	secs
resp read:	0.0000	secs,	0.0000	secs,	0.0008	secs

Status code distribution:

[2001 2000 responses]

4.1.12 /customer/customer-fetch-menu

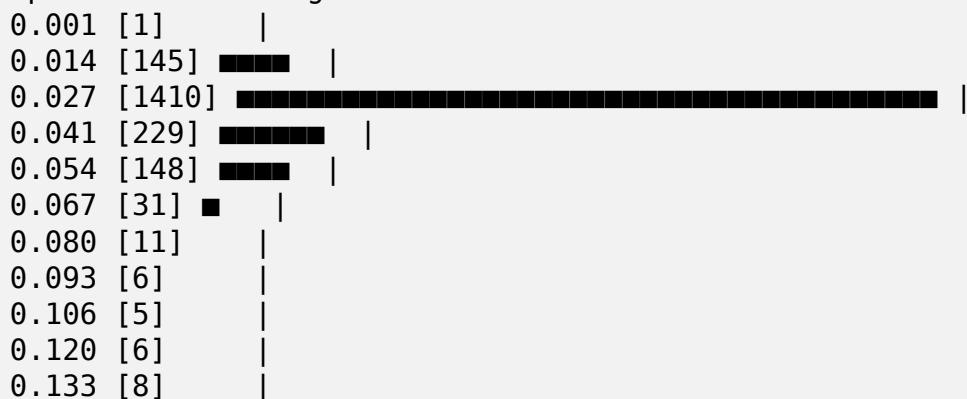
```
hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUI0iJjdXN0b21lciIsImlhdCI6MTc0Mzk2MjQzMjIwZTQ.
L9c90JZqCzLeNYQRu61Nt6yeFhrKBDICuYKLJ5NTE_k" \
-d '{"canteen_id":2}' \
http://localhost:4000/customer/customer-fetch-menu

hey -n 2000 -c 100 -m POST \
-H "Content-Type:application/json" \
-H "Cookie:_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJlc2VyX2lkIjoyNiwidXNlclR5cGUI0iJjdXN0b21lciIsImlhdCI6MTc0Mzk2MjQzMjIwZTQ.
L9c90JZqCzLeNYQRu61Nt6yeFhrKBDICuYKLJ5NTE_k" \
-d '{"canteen_id":2}' \
http://localhost:4000/customer/customer-fetch-menu
```

```
Slowest:      0.1329 secs
Fastest:      0.0010 secs
Average:      0.0238 secs
Requests/sec: 4097.3986
```

```
Total data:   1248000 bytes
Size/request: 624 bytes
```

Response time histogram:



Latency distribution:

```
10% in 0.0152 secs
25% in 0.0168 secs
50% in 0.0188 secs
75% in 0.0251 secs
90% in 0.0408 secs
95% in 0.0475 secs
99% in 0.0925 secs
```

Details (average, fastest, slowest):

```
DNS+dialup:    0.0008 secs, 0.0010 secs, 0.1329 secs
DNS-lookup:    0.0007 secs, 0.0000 secs, 0.0142 secs
req write:     0.0000 secs, 0.0000 secs, 0.0019 secs
resp wait:     0.0228 secs, 0.0010 secs, 0.1119 secs
resp read:     0.0000 secs, 0.0000 secs, 0.0020 secs
```

Status code distribution:

```
[200] 2000 responses
```