Iqueue Project
Testing Document
Software Engineering for Automation (2022-2023)

Giacomelli Gianluca, 10615105
Gottardini Andrea, 10617000
Veronese Niccolò Enrico, 10620278

Professors: Rossi Matteo Giovanni
Lestingi Livia

Summary

The following documentation helps to explain the different testing benchmarks applied to the Iqueue software. For each of the most important components of the software, some test sets are developed in order to guarantee their robustness to possible wrong inputs from the users and to assure that the outputs of the system are anyway correct. With testing, we have followed a continuous process of validation and verification, that allowed us to improve our code robustness and compliance with specifications.

We want to remark on two important traits of testing: the independence of its validity from the chosen programming language and the fact that it cannot guarantee the absence of errors but permits detecting their presence. As Dijkstra stated, "*Program testing can be used to show the presence of bugs, but never to show their absence*".

Good testing should behold the following features:

1. Help isolate errors.
2. Be repeatable.
3. Be accurate.

1. White Box Testing

We choose to test the code related to the booking view, since it is a crucial component of our application. To ease the readability of the code, we chose to add proper comments. In addition, the relative application pages will be reported.

Test sets employed in the white-box testing will be identified based on the following criteria:

- Statement Coverage
- Edge Coverage
- Condition Coverage
- Path Coverage

Then, in each test for every coverage criteria, a coverage-based test evaluation will be performed to assess the test validity:

$$Percentage\ of\ coverage = \frac{Number\ of\ items\ covered}{Total\ number\ of\ items} \times 100$$

1.1 Booking view

We choose to test the code related to the booking view, thus the code that allows a customer to book a slot of a certain shop, because this is one of the most important features of our application. In addition, the relative application pages will be reported.

Software tested

Iqueue – booking view

Version

Latest

Goal

To test if significant parts of the program structure are inadequate.

Code

```python
def Booking_view(request, selected_category):
    #Loading of the Django forms into variables
    Shop_and_day_form = forms.Shop_and_day_selectionForm()
    TimeSlot_form = forms.TimeSlot_selectionForm()

    #Filtering of the shops based on the category that the customer has selected in
    the previous view
    shops = Shop.objects.filter(category=selected_category)

    #Identification of the actual queue and the rating of every shop of the
    selected category to populate the map with them
    queues = []
    reviews = []

    for shop in shops:
        timeslot1 = TimeSlot.objects.filter(shop=shop)
        num_av_slots, _, _ = shop.checkQueue(timeslot1)
        shop.queue = num_av_slots + shop.queue_no_app
        shop.save()
        queues.append(shop.queue)
        reviews.append(shop.rating)


    #Identification of the shop addresses and names to populate the map properly
    addresses = [shop.address for shop in shops]
    names = [shop.name for shop in shops]


    #Condition that verifies if form is sent
    if request.method == 'POST':

        #Condition that verifies if the first form is sent, thus the one of the
         shop selection to inspect its time slots
```

```python
if 'btnform1' in request.POST:
    # Recover the information of the selected shop and date
    shop_ids = request.POST.get('shop_ids')
    date = request.POST.get('date')
    # Checking on the validity of the date
    if  date < date.today():
        return render(request, 'ErrorPastBooking.html')
    #Finding out the associeted objects
    shop_sel = Shop.objects.get(ids=shop_ids)
    timeslots_shop_sel = TimeSlot.objects.filter(shop=shop, date=date,
    available=True)

    #Returning the webpage where now the available slots (if there are) are
    showed
    context2={'shops': Shop.objects.filter(category=selected_category),

    'timeslots': timeslots,
    'Shop_and_day_form': Shop_and_day_form, 'TimeSlot_form': TimeSlot_form,
    'addresses': addresses, 'names': names, 'queues': queues, 'reviews':
    reviews}

    return render(request, 'booking.html',context=context2)

#Condition that verifies if the second form is sent, thus the one of the
time-slot selection to make the reservations if possible
if 'btnform2' in request.POST and 'selected_slot' in request.POST:
    #Recovering the information of the actual customer of its selected time
    slot
    idc = request.session.get('idc', '')
    selected_timeslot_id = request.POST.get('selected_slot')
    #Identification of the associated timeslot object, its slots and its
    associeted shop
    timeslot = get_object_or_404(TimeSlot, id=selected_timeslot_id)

    slot = Slot.objects.filter(TimeSlot=timeslot,
    available=True).first()
    shop = timeslot.shop
    #Making the reservation for the customer: association of the slot to
    him and the slot is now unavailable
    slot.available = False
    slot.idc = idc
    slot.save()

    #Checking if the availability of the time slot: if all of the slots are
    unavailable, the it is also their timeslot
    if not Slot.objects.filter(available=True, TimeSlot=timeslot).exists():
        timeslot.available = False
        timeslot.save()
```

```python
#Creation of the QR to keep track of the reservation and rendering of
 its html webpage

… Code …


context3={'qr_code_img': qr_code_img_str,
          'shop':
          Shop.objects.filter(category=selected_category).values(),
          'QR': qr, 'addresses': addresses,
          'Shop_and_day_form': Shop_and_day_form,
           'TimeSlot_form': TimeSlot_form}

return render(request, 'qr.html',context=context3)




Context1 = {
    'shops': Shop.objects.filter(category=selected_category),
    'Shop_and_day_form': Shop_and_day_form,
    'TimeSlot_form': TimeSlot_form,
    'addresses': addresses,
    'names': names,
}

return render(request, 'booking.html', context=context1)
```

Control Flow graph

For simplicity, the rows associated with the setting of the context will not be reported in the control flow graph and the buttons realized in the html template that allows changing the webpage will not be considered; The for loops associated with the compilation of the addresses and names lists will be reported as statement blocks and not conditional ones since they are just employed to compile the two lists.
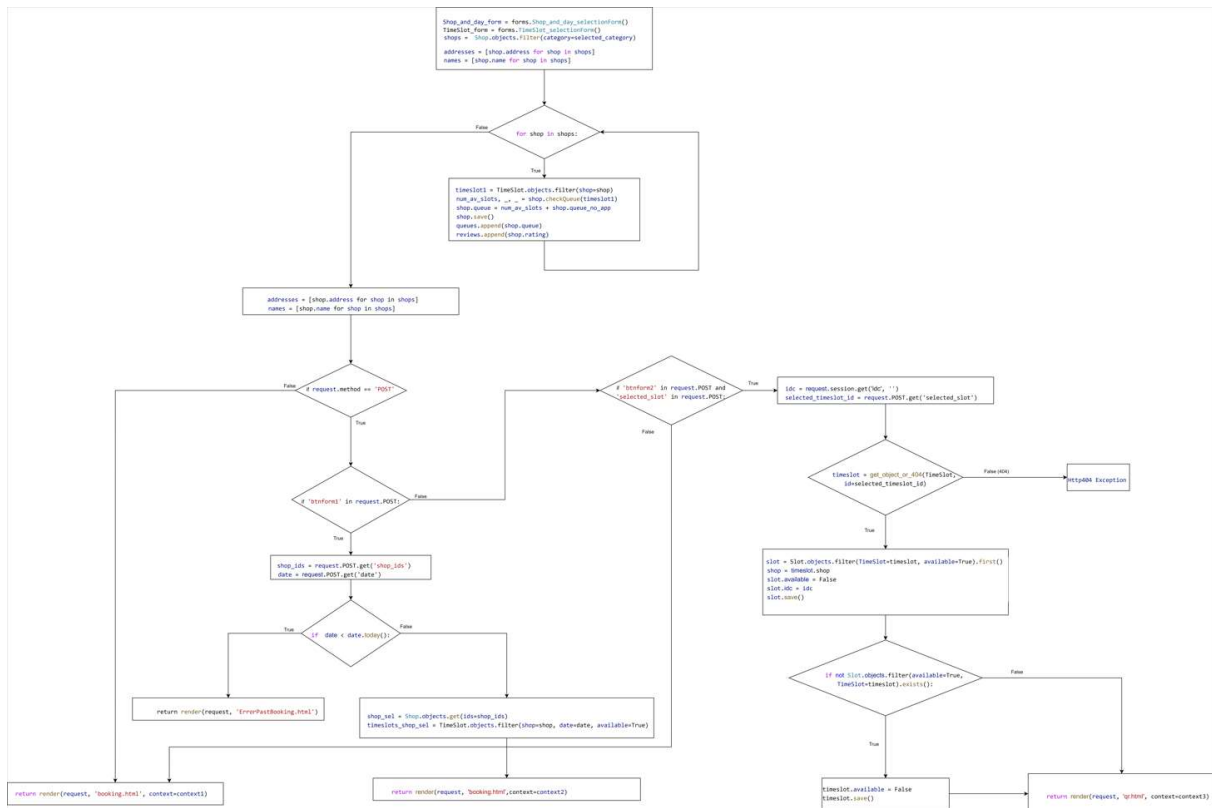

Figure 1: Control Flow graph Booking view

(A better image is in the folder White_box-Control flow graph)

Here to identify what **btnform1** and **btnform2** are, the screenshots of the application are reported. In order to send the form with **btnform1** the customer must have selected a date from the aside selection field. The webpage has been realized so that the **btnform2** can be pressed only after that the **btnform1** has been pressed at least once and the **btnform1** and **btnform2** cannot be sent together.
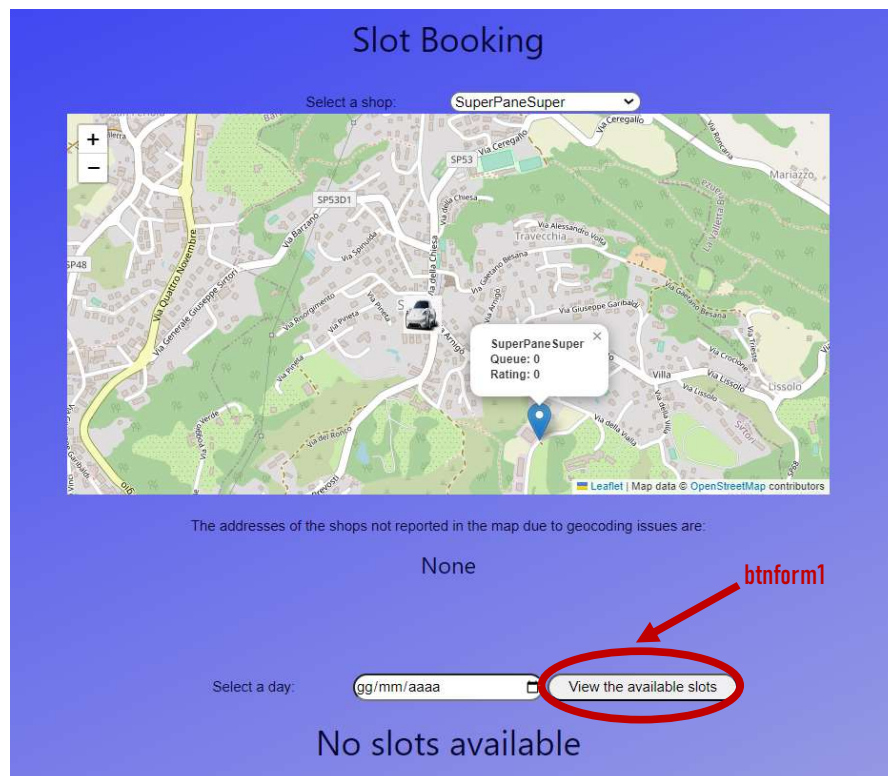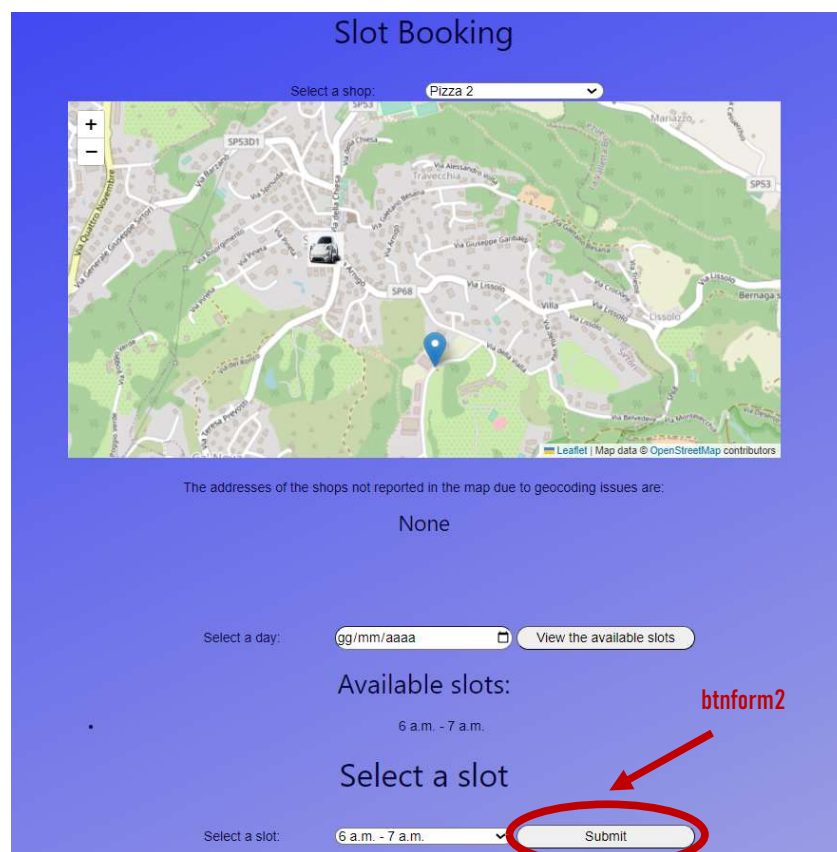
Figure 2: Screenshoot 1 (booking)

Figure 3: Screenshoot 2 (booking)

If no shops are provided by the initial query, the webpage appears as follows thanks to the code in the html template. In this case, btnform1 and btnform2 cannot be pressed.

Figure 4: No shops registeres (booking)

Before dealing with the different control flow coverage criteria, a description of the inputs of the test cases is provided:

- **selected_category**: it is the category of shops selected by the customer.
- **date**: it is the day the costumer wants to make its reservation.
- **shop_ids**: it is the ID code associated with the shop selected by the customer. It is described by a string of 36 characters.
- **btnform1**: button that sends the first form (see above). True means that the button has been pressed, false that it has not been pressed.
- **btnform2**: button that sends the second form (see above). True means that the button has been pressed, false that it has not been pressed.
- **selected_slot**: id assigned by Django to the TimeSlot object to record it in the database.

Since the webpage and so the booking view will be reloaded when we reach the `return render(request,'booking.html', context=… )` statement, we consider ending the test case when we reach it.
As `date.today()`will be considered the 14.06.2023.

Statement Coverage

Test set:

{ < selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2024, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2023, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268954>}

In this case are covered all the statements apart from the situation where `timeslot = get_object_or_404(TimeSlot, id=selected_timeslot_id)` returns the HTTP 404 exception.
This situation cannot be reached since the selectable timeslot is always registered in the Django database thanks to the webpage settings.

Considering the percentage of coverage:

$$Percentage\ of\ coverage|_{statement} = \frac{Number\ of\ statements\ covered}{Total\ number\ of\ statements} \times 100$$

We have in this case:

Number of statements covered = 54

Total number of statements = 55 (including the HTTP 404 exception)

Percentage of coverage = 98%

Edge Coverage

Test set:

{ < selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2024, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2023, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268954>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268975>}

In this case, all the edges are covered apart from these two:

1. The one associated with the HTTP 404 exception of `timeslot = get_object_or_404(TimeSlot, id=selected_timeslot_id)`. As said previously, this situation cannot be reached since the selectable timeslot is always registered in the Django database thanks to the webpage settings.

2. The one associated with the false condition of `if 'btnform2' in request.POST and 'selected_slot' in request.POST` because to reach this condition, the btnform2 must be pressed (so the first statement is always true if we reach this condition) and when we press this button a selected_slot is always selected due to the webpage settings.

Considering the percentage of coverage:

$$Percentage\ of\ coverage|_{edges} = \frac{Number\ of\ edges\ covered}{Total\ number\ of\ edges} \times 100$$

We have in this case:

Number of edges covered = 20

Total number of edges = 22 (including the HTTP 404 exception)

Percentage of coverage = 91%

Condition Coverage

Test set:

{< selected_category = Others, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2024, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2023, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268954>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268975>}

In the first test case, we have considered a situation where there are no shops of category Others.

In this case, all conditions are covered apart from these:

1. The one associated with the HTTP 404 exception of `timeslot = get_object_or_404(TimeSlot, id=selected_timeslot_id)`.

2. The two associated with the false condition of `if 'btnform2' in request.POST and 'selected_slot' in request.POST` because to reach this condition, the btnform2 must be pressed (so the first statement is always true if we reach this condition): due to this, this condition will be never false. In addition, when we press this button a selected_slot is always selected due to the webpage settings and so we are never in a situation where the first statement is true and the second is false.

Considering the percentage of coverage:

$$Percentage\ of\ coverage|_{condition} = \frac{Number\ of\ Conditions\ covered}{Total\ number\ of\ conditions} \times 100$$

We have in this case:

Number of conditions covered = 10

Total number of conditions = 13 (including the HTTP 404 exception)

Percentage of coverage = 77%

Path Coverage

Test set:

{< selected_category = Others, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2024, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = TRUE, date = 17.05.2023, shop_ids = a7b8da0f-b6ba-4f69-9b4e-119181849180, btnform2 = FALSE, selected_slot = None>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268954>,

< selected_category = Bakery, btnform1 = FALSE, date = None, shop_ids = None, btnform2 = TRUE, selected_slot = 268975>}

Since the for loop is iterated in the same way with the different shop category selection, it will be considered for simplicity two categories in the test-cases: category others where we have no shops and category bakery where we have shops.

Considering the percentage of coverage:

$$Percentage\ of\ coverage|_{path} = \frac{Number\ of\ paths\ covered}{Total\ number\ of\ paths} \times 100$$

We have in this case:

Number of paths covered = 6

Total number of paths = 8 (including the HTTP 404 exception)

Percentage of coverage = 75%

Overall results (Booking view)

Based on the performed testing, we can conclude that the booking view is significatively robust: this is due not only to the here reported view code but also to the implementation of the webpage template. The items that are not covered by the test set above are not an issue thanks to the well-structure of the webpage template.

Author

Gianluca Giacomelli

We remember that the condition coverage and the path coverage are finer than the edge coverage that is finer than the statement coverage. When a criteria is finer than another this means that for any test sets satisfying the first there is a subset of it that also satisfies the latter. This is reflected by the test sets implemented by us.

A weakness of all of the white-box techniques is that the focus on the code, but they do not tackling directly on the specifications. To realize a complete and exhaustive testing campaign, black-box techniques will be considered.

2. Black Box Testing

The goal of black box testing is to test what the program is supposed to do and hence, the test cases are derived from the software specifications. In the following examples the main test sets of the Iqueue application are checked, each one with different test cases in order to see not only the outputs when standard inputs are applied but also when the preconditions are violated. In this way, the robustness of the code is guaranteed and specifically in Iqueue application this property is fundamental since it is characterized by many input user commands.

In this case, we will consider programs as functional software so described by an input-output relationship as follow:

- Takes some values in input
- Performs computations
- Returns some value in output

2.1 Test set: Account Registration

Software tested

Iqueue – registration view

Version

Latest

Goal

To show that the registration code guarantees that the user can register in the Iqueue application and it is robust with respect to different inputs from the users.

2.1.1 Test Case 1: Standard case

Goal

To show that the account registration code will allow to register a new user if all the inputs are correct.

Input

Birthday: 01/02/1999, Email: niccoloenrico.veronese@mail.polimi.it (not already used)

Expected output

Registration successfully completed.

Actual output

Registration successfully completed and a new account is created.

Result

The code allows the user to register a new account.

2.1.2   Test Case 2: Birthday after the current date

Goal

To show that the account registration code will not allow to insert an invalid birthday

Input

Birthday: 10/7/2050

Expected output

"The birthday cannot be in the future"

Actual output

Error

Oops! The birthday cannot be in the future! Please go back.

Figure 5: Error for a future birthday

Result

The code is robust with respect to a user who tries to put a future birthday

2.1.3   Test Case 3: Wrong email

Goal

To prove that the system will not allow to put an incorrect mail in the registration

Input

Email field: niccolo.veronese.it

Expected output

Impossible to create the account without a correct email field

Actual output

Figure 6: actual output in case of a wrong email

Result

The code shows good robustness with respect to wrong emails.

2.1.4 Test Case 4: Account with the same email address of another user

Goal

To guarantee that two user do not have the same email address

Input

Email address: gianluca99.giacomelli@gmail.com (already existing)

Expected output

Validation error: "the given email is already registered"

Actual output

Figure 7: Error in case the user subscribes with al already registered email.

Result

The Iqueue prevents users to register with an already existing email.

Overall results (Account Registration)

The registration component is robust with respect to many wrong input parameters from the users such as a birthday date which is in the future, a wrong email address or even worse an email address which is already used.

14

Author

Niccolò Veronese

2.2   Test set: Shop registration

Goal

To show that the registration shop code is robust with respect to different inputs from the users and that it will allow to register a new shop in the Iqueue app.

2.2.1   Test Case 1: standard case

Goal

To show that the shop registration code will allow to register a new shop.

Input

Opening time 08:00 and Closing time 18:00

Expected output

Registration successfully completed.

Actual output

Registration successfully completed.

Result

The code allows the user to register a new shop.

2.2.2   Test Case 2: opening time after the closing time

Goal

To show that the shop registration code will not allow to insert an invalid opening time

Input

Opening time 09:00 and Closing time 08:00

Expected output

"The opening time must be before the closing time"

Actual output
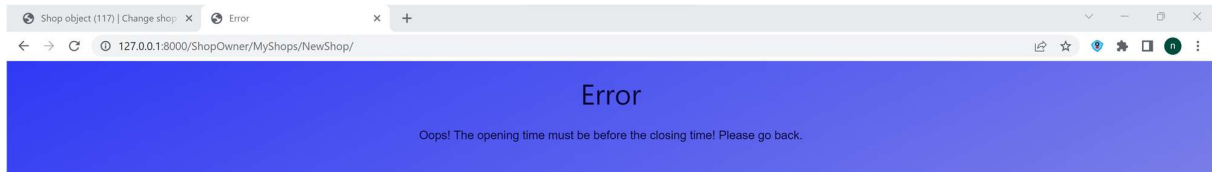
Figure 8: Error in case the Shop Owner inserts an opening time above the closing time.

Result

The code is robust with respect to a user who tries to put a wrong opening/closing time

2.2.3   Test Case 3: Negative max number of clients

Goal

To prove that the system will not allow to put a negative max number of clients

Input

Max number = -2

Expected output

The code will not allow to enter a negative number.

Actual output

The code do not write the minus symbol and it doesn't allow to go below zero with the arrow of the button

Result

The code shows good robustness with respect to negative number of clients.

2.2.4   Test Case 4: Negative slot durations

Goal

To guarantee that the shop owner must not enter a negative slot duration

Input

Negative slot duration

Expected output

The code does not write negative numbers

Actual output

The code does not write negative numbers in that field

Result

The Iqueue prevents users to register the shops with a negative slot duration

2.2.5 Test Case 5: Time slot greater than the total opening time

Goal

To guarantee that the shop owner must not enter a time slot which is greater than the total opening time of the shop

Input

Opening time 10.30, Closing time 12.00, Slot duration 100 min

Expected output

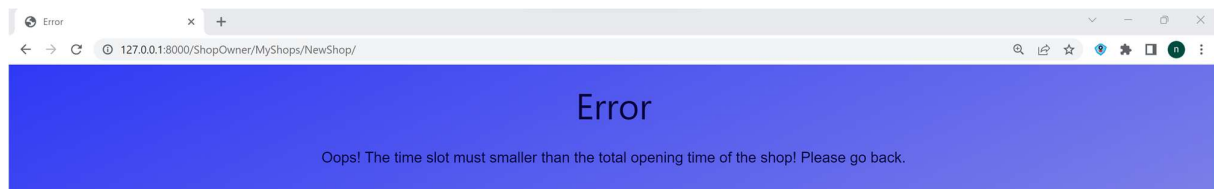The code will not allow to register that Shop

Actual output

Figure 9: Error when a too greater time slot is registered

Result

The Iqueue prevents users to register the shops with a time slot greater than the total opening time of the shop.

Overall results (Shop registration)

The shop registration component is robust with respect to many wrong input parameters from the users such as a opening time which is after the closing time, negatives max numbers of clients and negatives time slots.

Author

Niccolò Veronese

2.3 Test set: Booking and QR reading

Software tested

Iqueue – Booking view, qr view

Latest

To show that the booking and the qr reading code is robust with respect to different inputs from the users.

2.3.1 Test Case 1: scan of a QR code

Goal

To show that the Iqueue app is able to scan a QR code for a reservation

Input

Qr code

Expected output

Scan successfully completed and the reservation disappears.

Actual output

Scan successfully completed and if enters into the myReservations page

Figure 10: MyReservations view when the QR code has been scanned. It is supposed that only one reservations was created.

Result

The code allow the user to see its reservations and once the QR code has been scanned the reservation will disappear.

2.3.2 Test Case 2: scan of an already used qr code

Goal

To show that the Iqueue app will not allow to scan two times the same qr code

Input

Qr code already used

Expected output

Impossible to scan the Qr code

Actual output
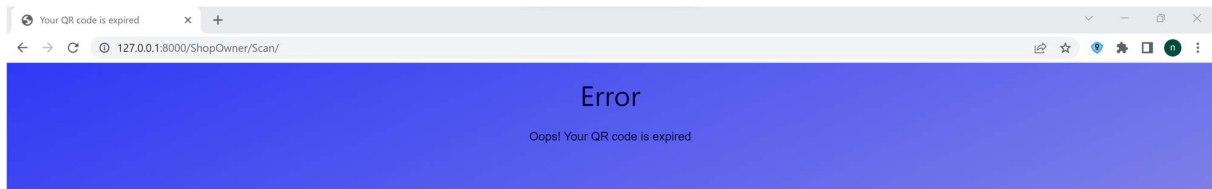
Figure 11: Error in case the Customer tries to scan twice the same QR code

Result

The code will not allow to scan two times the same Qr code.

2.3.3 Test Case 3: Selection of a time slot in the past

Goal

To prove that the system will not allow to book a passed time slot

Input
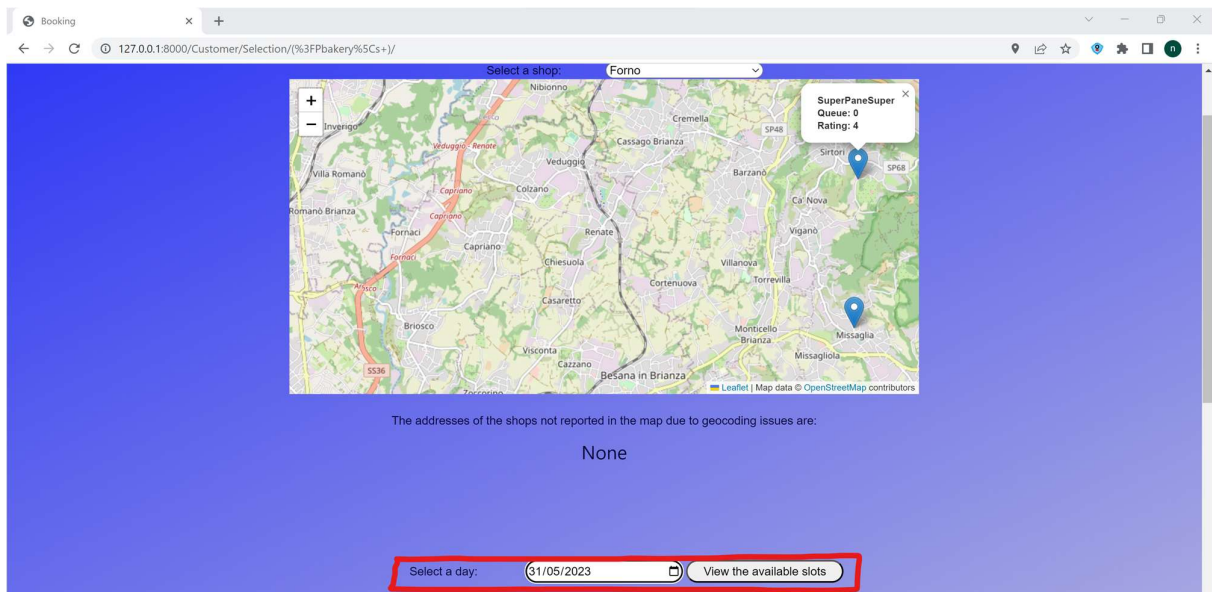
Time slot in the past: 31/05/2023

Figure 12: Input with past time slot.

Expected output
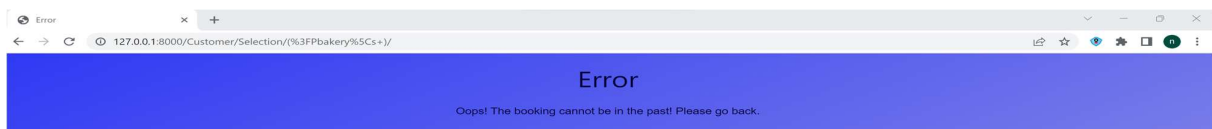
The code will not allow to book a time slot in the past.

Actual output

Figure 13: Error for a passed booking

Result

The code shows good robustness with respect to slots in the past since it shows that no slots are available for that date.

2.3.4   Test Case 4: Time Slot too far in the future

Goal

To guarantee that the customer cannot book a time slot more than one year in advance

Input

Booking for the 14/02/2025

Expected output

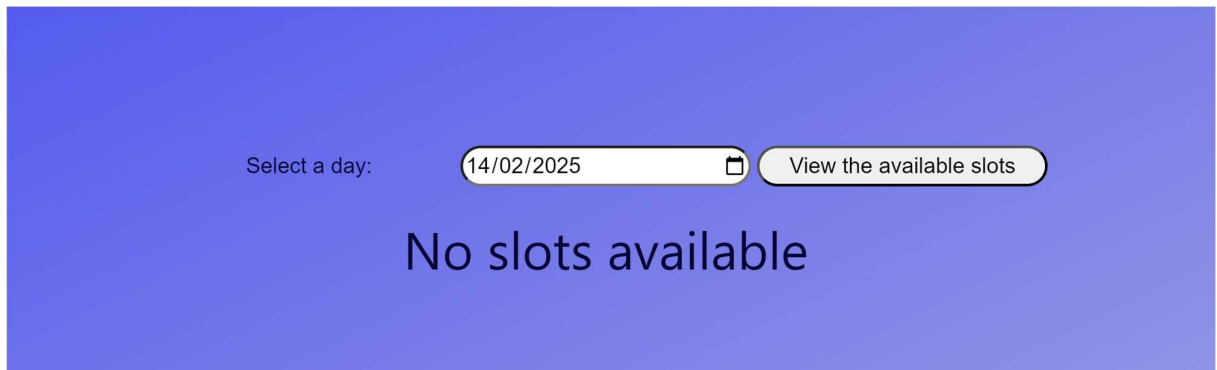The code will not show the available slots

Actual output

Figure 14: No available slots too far in the future (more than 52 weeks from the Shop creation)

Result

The Iqueue prevents users to book a slot that is more than one year in advance

Overall results (Booking and qr code reading)

The booking and QR reading components are robust with respect to many wrong input parameters from the users such as booking of time slots in the past or too far in the future and already used QR.

Author

Niccolò Veronese

2.4 Test set: Queue

Iqueue – ShopQueueList

Latest

To show that the Iqueue app is able to correctly show the number of people in queue in front of a shop, considering also the ones without the Iqueue application.

2.4.1 Test Case 1: Booking for the 15th of June, no customer without Iqueue and actual date.

Goal

To show that the Iqueue app can show to the Customer its position in the queue for the selected time slot.

Input

Booking for the 15th of June at the shop Niccolò Enrico Veronese and with no other customers in line on that day in that time slot.

Expected output

The position of the queue is 1.

Actual output

The position in the queue for the selected time slot is 1. Since this test case is developed on the 13th of June the queue in front of the Shop in that moment remains 0.
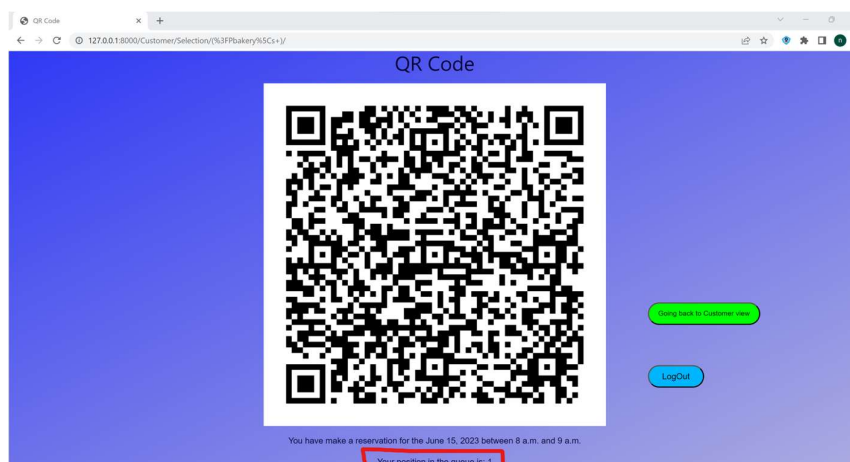

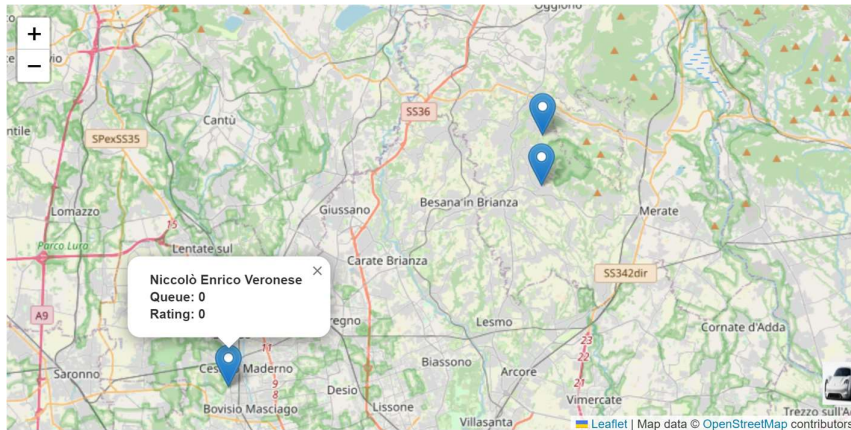*Figure 15: QR code with the position in the queue*

Figure 16: Map with the current queue in front of the Shop. Since the booking it is generated for a date different from the actual date it is correct that the queue remains 0

Result

The code allows the user to see its number in the queue for that time slot.

2.4.2   Test Case 2: Booking for the 15th of June, no customer without Iqueue and forced date

Goal

To show that the Iqueue app can increment the number of people in queue in real time.

Input

Booking for the 15th of June at the shop Niccolò Enrico Veronese and with no other customers in line on that day in that time slot, the date now is forced to be the 15th of June.

Expected output

The queue in front of the shop will change to 1.

Actual output

The customer will see from the map that the shop has now 1 people in queue and the shop owner can see the same number of queue from its MyShops page.
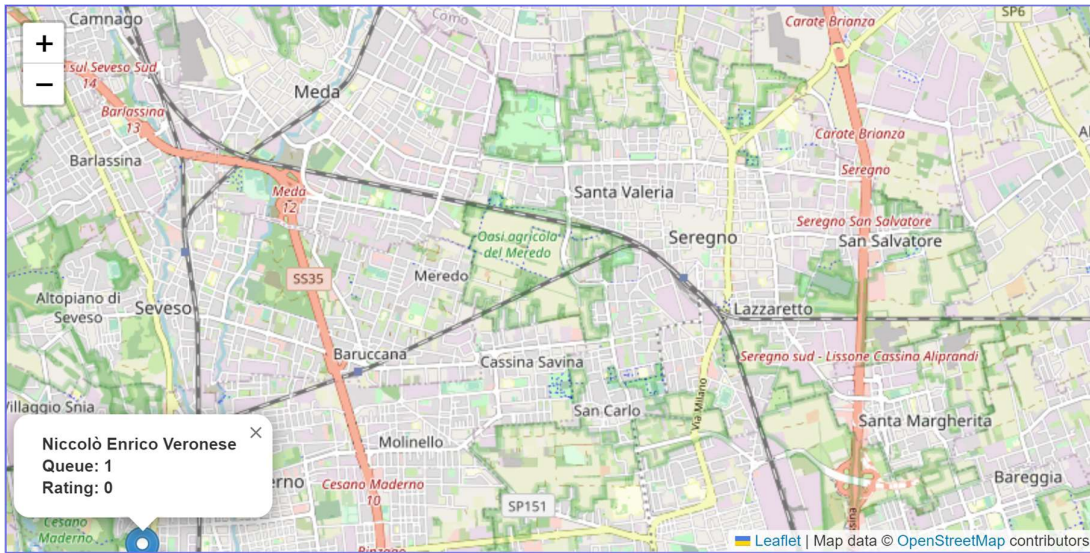
Figure 17: Updated queue in front of the shop in the map.

Figure 18: Updated queue in front of the shop in the MyShops view.

Result

The queue in front of the shop will change correctly in real time. In this case the real time has been simulated by forcing the date.

2.4.3   Test Case 3: Booking for the 15th of June, adding a customer without Iqueue and forced date

Goal

To show that the Iqueue app can increment the number of people in queue in real time, considering also possible customers without the Iqueue app.

Input

Booking for the 15th of June at the shop Niccolò Enrico Veronese and with 1 other customer who enters in the shops without the Iqueue app.

Expected output

The queue in front of the shop will change to 2 both in the map and in the MyShop view.

Actual output

The customer will see from the map that the shop has now 2 people in queue and the shop owner can see the same number of queue from its MyShops page.
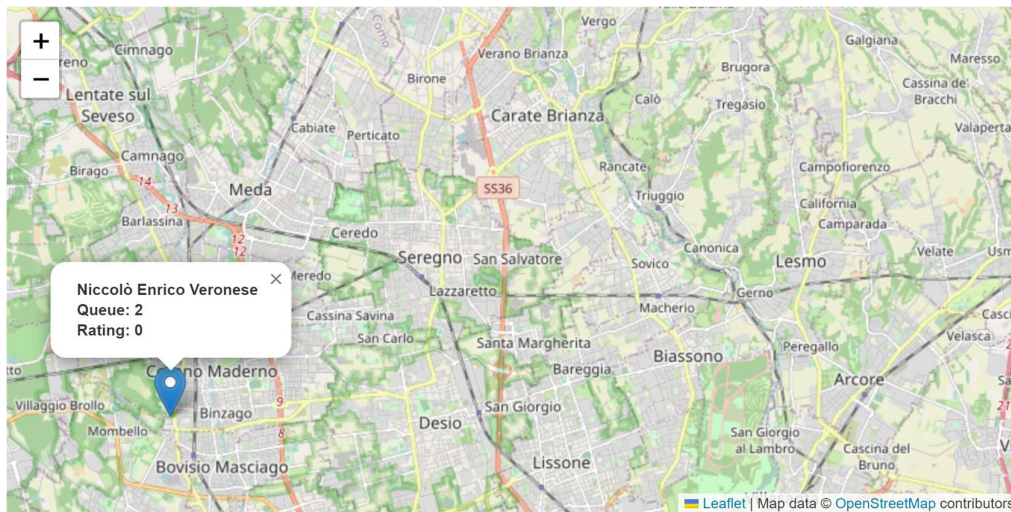

Figure 19: Updated queue in the map considering the customer without Iqueue app
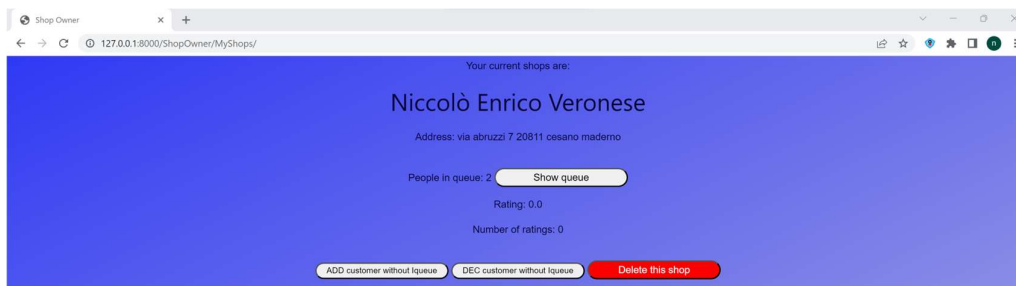

Figure 20: Updated queue in the My Shop view considering the customer without Iqueue app

Result

The software can correctly tracking the number of people in queue, also considering the ones without the Iqueue app.

2.4.4 Test Case 4: Scanning of the QR code and forced date

Goal

To show that the Iqueue app can decrement the number of people in queue in real time, considering also possible customers without the Iqueue app.

Input

Scanning the QR code corresponding to the booking for the 15th of June at the shop Niccolò Enrico Veronese and with 1 other customer who enters in the shops without the Iqueue app.

Expected output

The queue in front of the shop will change to 1 both in the map and in the MyShop view for the Shop Owner.

Actual output

The customer will see from the map that the shop has now 1 people in queue (only the Customer without the Iqueue app) and the shop owner can see the same number of queue from its MyShops page.
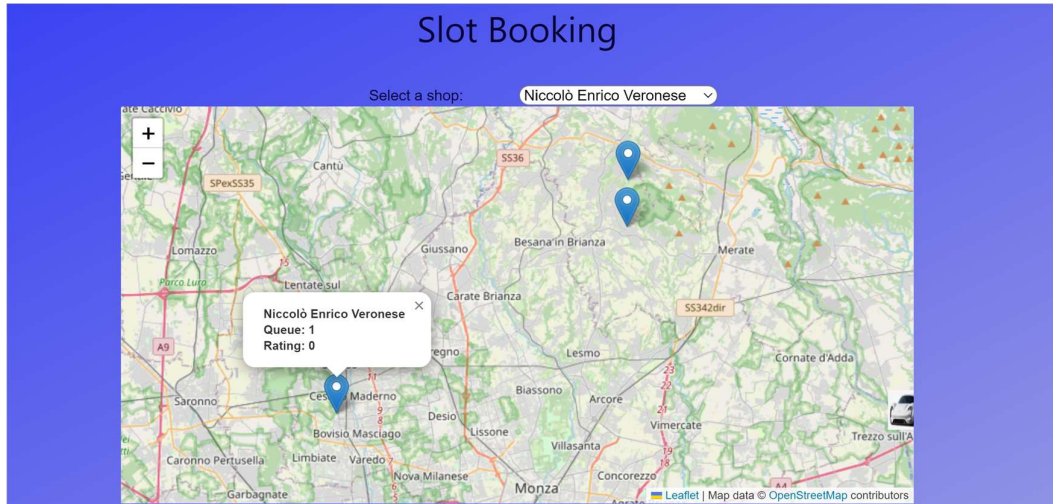

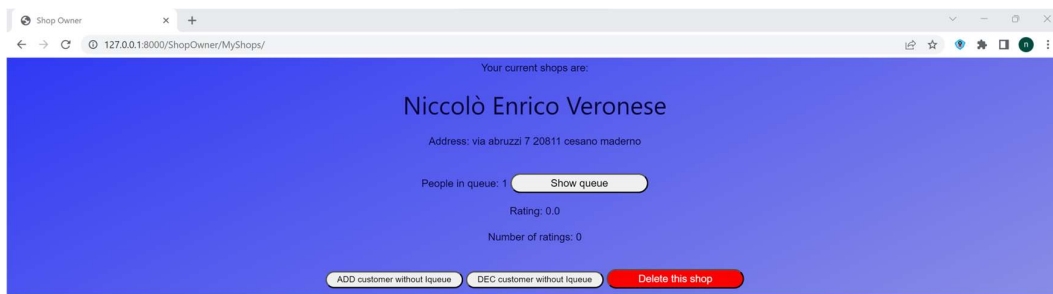*Figure 21: Update of the queue in the map when scanning the QR code*

Figure 22: Update of the queue in the My Shop view when scanning the QR code

Result

The software can correctly track the number of people in queue when a QR code has been scanned, also considering the ones without the Iqueue app.

Overall results (Queue)

The queue component is able to correctly track the number of people in queue in front of the shop in real time and considering also the ones without the application.

Author

Niccolò Veronese

2.5   Test set: Product

Iqueue – Product view, scan product, purchase list, wishlist

Version

Latest

Goal

To show that the Iqueue app is able to register a product linked to a Shop, scan it using another QR code and define a purchase list and a wishlist with the products.

2.5.1   Test Case 1: Product registration

Goal

To show that the Iqueue app can allow the Shop Owner to register a product for his shop.

Input

Focaccia ligure - Price: 10.0 - Quantity: 5 – Discount: 20%

Pizza margherita - Price: 5.0 - Quantity: 1 – Discount: 30%

Expected output

Focaccia ligure - Price: 10.0 - Quantity: 5 – Discount: 20%

Pizza margherita - Price: 5.0 - Quantity: 1 – Discount: 30%
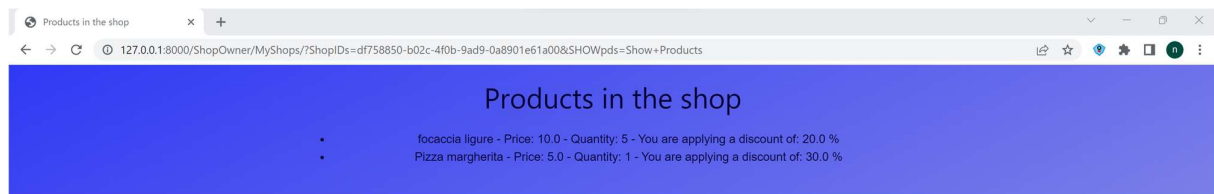
Actual output

Figure 23: Products registered in the Shop.

Result

The code allows the user to correctly register the products for the Shop, without giving the possibility to insert negative values.

2.5.2   Test Case 2: Wishlist

Goal

To show that the Iqueue app can allow the Customer to generate its Wishlist with the products of the Shops

Input

Search for "focaccia" and for "pizza"

Expected output

The software should show the products inserted before, since it is case insensitive, and give the possibility to add them to a wishlist.

Actual output

Figure 24: Product "focaccia ligure" found when searching for "focaccia"

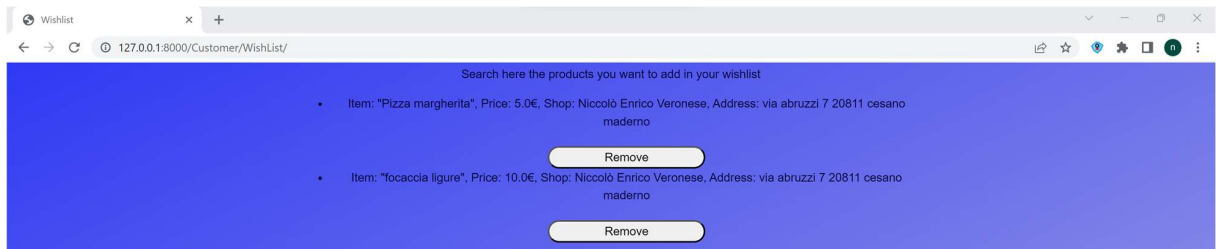Figure 25: Product "Pizza margherita" found when searching for "pizza"

Figure 26: Wishlist of the Customer

Result

The code allows the user to correctly search the products, without problems with different cases, and to add them in a wishlist.

2.5.3 Test Case 3: Product scanning and purchase list

Goal

To show that the Iqueue app can allow the Shop Owner to scan a product bought by the Customer and that the same product will be added to the purchase list of the Customer.

Input

The Customer will buy the product "Pizza Margherita", that is available in the Shop in 2 quantities
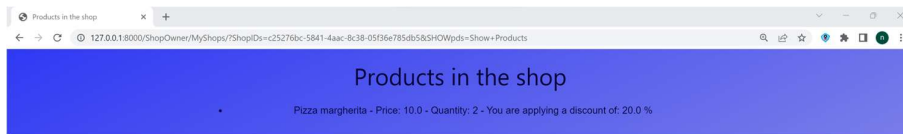
Figure 27: List of the available products in the Shop at the beginning.

Figure 28: QR code linked with the Product "Pizza Margherita"

Expected output

The product quantity will change to 1 and the product "pizza margherita" will be added to the purchase list of the Customer.

Actual output

The product "Pizza margherita" is added to the purchase list of the Customer and the quantity in the Shop is reduced to 1.

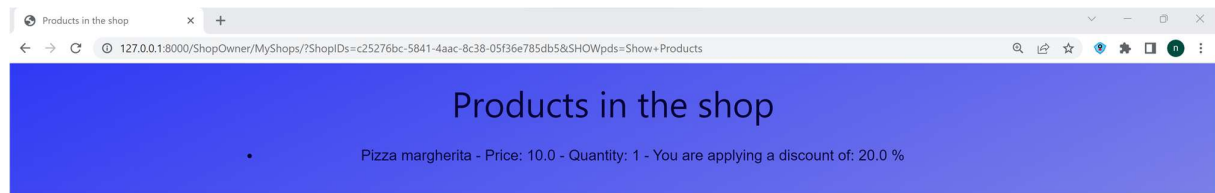*Figure 29: Purchase list of the Customer after the scanning*

Figure 30: List of the products after the scanning.

Result

The code allows the user to correctly scan the products for the Shop, and it adds them in the purchase list of the Customer.

Overall results (Product)

The product component is able to correctly let the Shop Owner add products related to the Shops. In addition to that, the software guarantees that the Customer can add his products in a wishlist, without being sensible to case issues. Finally, it can track the number of sold products in real time and add also the results in the purchase list of the Customer.

Author

Niccolò Veronese

2.6   Test set: Advertisement

Software tested

Iqueue – Advertisement view

Version

Latest

Goal

To show that the Iqueue app is able to register an advertisement for the Shop.

2.6.1   Test Case 1: Advertisement registration

Goal

To show that the Iqueue app can allow the Shop Owner to generate an advertisement for his shop.

Input

Advertisement for the Shop "Panificio Fumagalli"

Select which shop do you want to advertise. You can only advertise shops that are not advertised yet.

To see which shops have you already advertised, go back to MyShops.

Panificio Fumagalli

How long do you want to advertise your shop?

1 month

The advertisement period starts from now.

Record

This is a free trial. Iqueue team plans to make a charge upon advertising your shop in the future.

Figure 31: Advertisement for the Shop "Panificio Fumagalli"

Expected output

The Shop has been advertised and the image of the Shop appears in the initial page.

Actual output

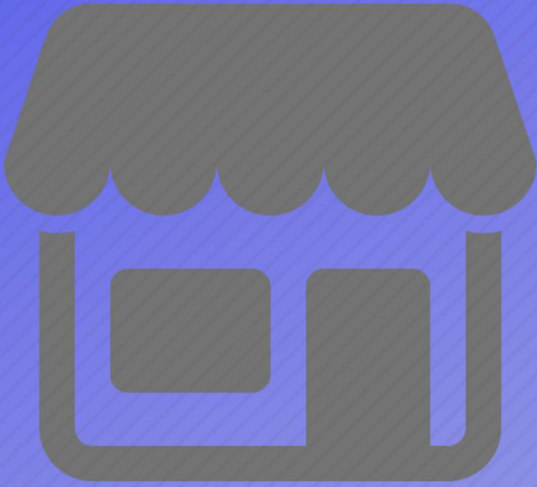# Select Your Role

Shop Owner   Customer

Welcome back Niccolò Enrico !

Your Customer ID: 6950ec7b-2232-4caf-958e-7f75c7d3f658

Your Shop Owner ID: 045ab391-74a2-46b2-98a2-0c994e07a68e

Advertisement:

Panificio Fumagalli

Location: via abruzzi 7 20811 cesano maderno

Figure 32: Advertised Shop in the homepage

Result

> The code allows the user to correctly generate the advertisement for the Shop and the image with the shop name and address appears in the initial page of all the users.

Overall results (Product)

The advertisement component is able to generate an advertisement for the Shop and it shows the promotional image in the initial page.

Author

Niccolò Veronese

3. Conclusions

The testing phase assures that the code of the Iqueue app meets the requirements and it is robust with respect to the inputs of the users since all the main components of the system have been tested considering both an ideal situation and strange/wrong inputs. Since the booking view is crucial for the Iqueue app, we chose to perform white-box testing on it.

The robustness of the system is a crucial aspect of the Iqueue application because it is characterized by many pages in which the user plays an active role. Although the results of the testing guarantee a high level of robustness, as for all the testing cases, these do not correspond to the absolute certainty of zero bugs inside the application. However, thanks to the developed testing, we can confidently say that the Iqueue application is robust and it is ready to be further developed.