1.INTRODUCTION

Online Mens-Hut is a website in which the user sitting infront of the computer can access the mens products through online. It includes registration of customers, storing their details into the system. The system has the facility to give an unique id for every order. It includes a menu from where we can select a product and add how much quantity of a product we needed. Then the system adds the items to cart and payment mode for this system is cash on delivery, where the payment is made at the time of delivery rather than in advance. The interface is very user-friendly. The data is well protected for personal use and makes the data processing very fast.

2.SYSTEM ANALYSIS

2.1 LITERATURE SURVEY

The scope of **"Online Mens-Hut"** is to display the details of items and orders of a logged in user. It contains another admin page where he can add, delete, update and view items. Here the admin is going to maintain the details of orders that are placed, cancelled and finished.

2.2 EXISTING SYSTEM

The Existing System of buying products has several disadvantages. It requires lots of time to travel to the particular shop to buy the products. Since everyone is leading busy life now a days, time means a lot to everyone. Also there are expenses for travelling from house to shop. Moreover the shop from where would like to buy something may not be open 24*7*365. Hence we have to adjust our time with the shopkeeper's time or vendor's time.

Disadvantages:

1. It is less user-friendly.
2. User must go to shop and select products.
3. Time consuming.

2.3 PROPOSED SYSTEM

In order to overcome these, we have e-commerce solution, i.e one place where we can get all required products online. The proposed system helps in building a website to buy, sell products online using internet connection. Purchasing of goods online, user can choose different products based on categories, prices and hence covering the disadvantages of the existing system and making the buying easier and helping the vendors to reach wider market.

2.4 SYSTEM REQUIREMENTS

Software Requirements:

Operating System	:	Windows 8
Language	:	Python 3.7
Framework	:	Flask
Database	:	SQLITE 3
Browser	:	Any Browser with GUI

Hardware Requirements:

Processor	:	Intel(R) Core(TM) i-5 6200U
Input devices	:	Mouse, Keyboard
RAM	:	8GB

3.DESIGN

3.1 MODULE DESCRIPTION

The system consists of 2 modules. They are :-

1.Administrator

2.User

3.1.1Administrator Module

Administrator can manage hole site from admin Dashboard. He can manage the things like add, update, delete and view item details. He can view item details, placed, cancel, and finished orders of user. He finishes the order once the payment is done by the customer. He can view the Door Delivery details of that placed, cancelled, and finished orders also.

3.1.2 User Module

- User register with their details
- Login to website
- Add products to cart and their quantity
- View cart once the selection is over
- Give Delivery Details
- Place Order
- View Placed orders
- Cancel order
- View Cancel orders
- Update their details
- Logout

3.2 DATABASE DESIGN

Design creates a representation or model, provides details about software data structure, architecture, interfaces and components that are necessary to implement a system. Design is the perfect way to accurately translate user's requirement in to a finished software product.

Database Tables:

3.2.1 Admin Table:

Field name	Data Type	Constraint	Description
admin_email	Text	Primary Key	Usernme of Admin
admin_pswd	Text	Not Null	Password of Admin

Table 3.2.1: admin_details Table

3.2.2 Customer Table:

Field name	Data Type	Constraint	Description
cus_name	Text	Not Null	Name of customer
cus_email	Text	Unique	Email address of customer
cus_phn_no	Integer	primary key	Username of customer
cus_pswd	Integer	Not Null	Password of customer
cus_dr_num	Text	Not Null	Home address of customer
cus_street	Text	Not Null	Street of customer
cus_town	Text	Not Null	Town of customer
cus_city	Text	Not Null	City of customer
cus_state	Text	Not Null	State of customer
cus_pin	Integer	Not Null	Pincode of the city

Table 3.2.2: customer Table

3.2.3 Product Table:

Field name	Data Type	Constraint	Description
pr_id	Integer	Primary Key	Product id
pr_name	Text	Not Null	Product name
pr_cat_name	Text	Not Null	Name of the catalogue
pr_img	Text	Not Null	Image of the product
pr_desc	Text	Not Null	Description of the product
pr_price	Integer	Not Null	Price of the product
pr_qty	Integer	Not Null	Number of product

Table 3.2.3: product Table

3.2.4 Cart Table:

Field name	Data Type	Constraint	Description
cart_or_id	Integer	Foreign key, Composite key	Order id
cart_pr_id	Integer	Foreign key, Composite key	Ordered Product id
Cart_quantity	Integer	>0	Count of items

Table 3.2.4: cart Table

3.2.5 Order Details Table:

Field name	DataType	Constraint	Description
or_id	Integer	primary key	order id
or_cus_phn_no	Integer	Foreign Key	customer phone number
or_date	Text	Not Null	ordered date
or_status	Text	Not Null	Pending/placed/Cancelled/finished

Table 3.2.5: orders Table

3.2.6 Door details Table:

Field name	Data Type	Constraint	Description
do_or_id	Integer	Foreign key	Order id
do_name	Text	Not Null	Receiver name
do_ph_no	Integer	Not Null	Receiver number
do_address	Text	Not Null	delivery address

Table 3.2.6: doordetails Table

3.2.7 Payment Table:

Field Name	Data Type	Constraint	Description
Pay_or_id	Integer	Foreign Key	Order id
Pay_type	Text	Not Null	Payment type
Pay_card_no	Integer	Not Null	Card Number
Pay_cus_name	Text	Not Null	Name of person
Pay_date	Text	Not Null	Date
Pay_amount	Real	Not Null	Amount paid

Table 3.2.6: payment Table

3.3 SYSTEM DESIGN:

3.3.1 Data flow diagram:

A **data-flow diagram** (DFD) is a way of representing a flow of a data of a process or a system (usually an information system) The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops.

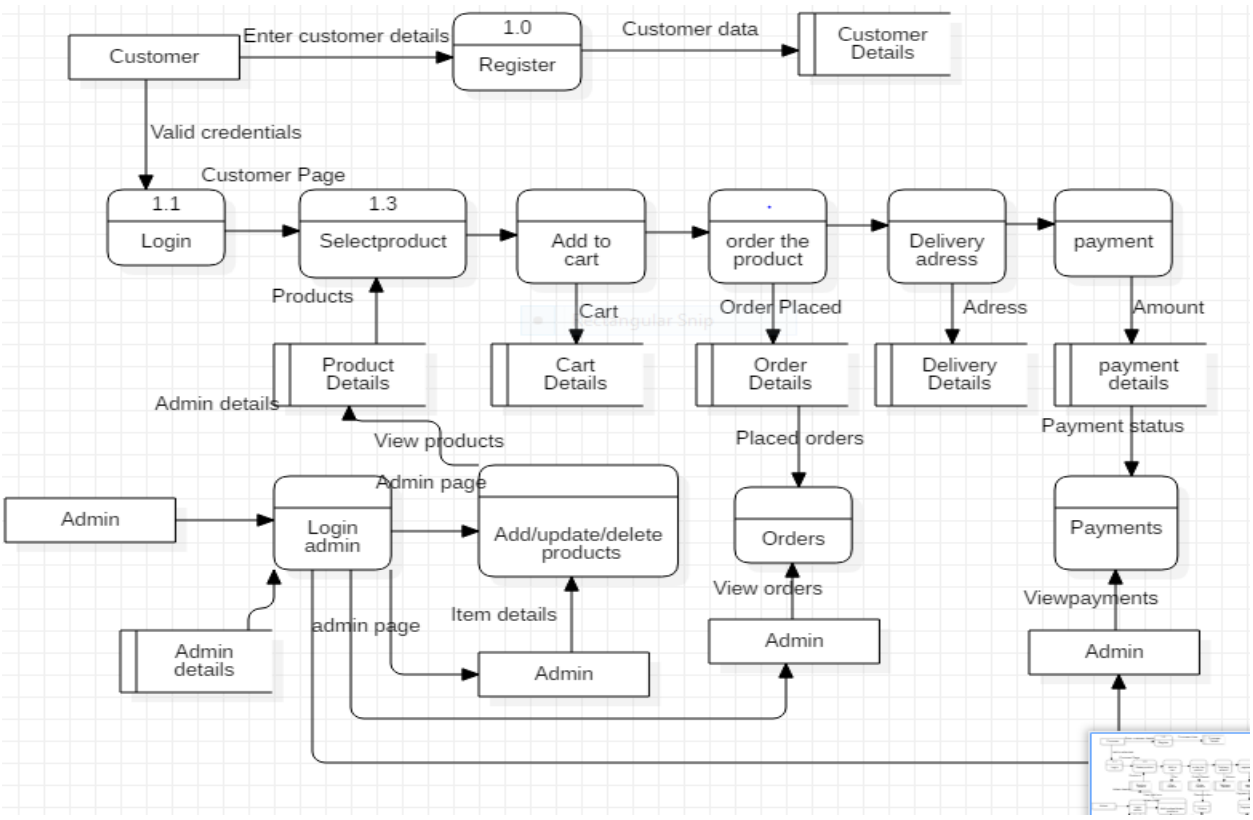


Figure Data Flow Diagram for Customer,Admin

4.IMPLEMENTATION

4.1 FLASK:

If you decide you want to build a web application, and you would like to develop it in Python, you'll probably want a so-called web framework. There are a lot of repetitive and boring parts of building backend logic, user interface, and hooking everything up to the Internet so that users can navigate your app in their browser. A web framework aims to implement all the functionality common to most web applications, such as mapping URLs to chunks of Python code.

Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. “Micro” does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The “micro” in microframework means Flask aims to keep the core simple but extensible. Flask won’t make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. By default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to your application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs.

4.2 FLASK vs DJANGO:

Django and Flask are hugely popular among Python programmers. Django is a full-stack web framework for Python, whereas Flask is a lightweight and extensible Python web framework. Developers have to build web applications by availing the built-in features provided by the web framework. On the other hand, Flask is a micro but extensible web frameworks. Django is a full-stack Python web framework. Also, it is developed based on batteries included approach. The batteries included in Django makes it easier for Django developers to accomplish common web development tasks like user authentication, URL routing and database schema migration. Also, Django accelerates custom web application development by providing built-in template engine, ORM system, and bootstrapping tool. On the other hand, Flask is a simple, lightweight, and minimalist web framework. It lacks some of the built-in features provided by Django. But it helps developers to keep the core

of a web application simple and extensible. Unlike Flask, Django makes it easier for users to handle common project administration tasks by providing a ready-to-use admin framework. It further generates the functional admin module automatically based on project models. Flask is developed based on Jinja2 template engine. As a fully-featured template engine for Python, Jinja2 is also inspired by Django's template system. It enables developers to accelerate development of dynamic web applications by taking advantage of an integrated sandboxed execution environment and writing templates in an expressive language. Django allows developers to take advantage of a robust ORM system. The developers can use the ORM system to work with widely used databases like MySQL, Oracle, SQLite and PostgreSQL. Also, the ORM system enables developers to perform common database operations without writing lengthy SQL queries. Unlike Django, Flask does not provide a built-in ORM system. It requires developers to work with databases and perform database operations through SQLAlchemy. Flask requires developers to each project as a single application. But the developers have option to add multiple models and views to the same application. On the other hand, Django allows developers to divide a project into multiple applications.

4.3 VIRTUAL ENVIRONMENT:

Use a virtual environment to manage the dependencies for your project, both in development and in production. The more Python projects you have, the more likely it is that you need to work with different versions of Python libraries, or even Python itself. Newer versions of libraries for one project can break compatibility in another project. Virtual environments are independent groups of Python libraries, one for each project. Packages installed for one project will not affect other projects or the operating system's packages.

4.4 INSTALLATION OF FLASK:

Create a project folder and a venv folder within:

```
Mkdir myproject
cd myproject
python3 -m venv   venv
```

On Windows:

```
py -3 -m venv venv
```

If you needed to install virtualenv because you are on an older version of Python, use the following command instead:

```
virtualenv venv
```

On Windows:

\Python27\Scripts\virtualenv.exe venv

Activate the environment

Before you work on your project, activate the corresponding environment:

. venv/bin/activate

On Windows:

venv\Scripts\activate

Your shell prompt will change to show the name of the activated environment. **Install Flask**

Within the activated environment, use the following command to install Flask:

pip install Flask

Living on the edge

If you want to work with the latest Flask code before it's released, install or update the code from the master branch:

pip install -U https://github.com/pallets/flask/archive/master.tar.gz

Install virtualenv

If you are using Python 2, the venv module is not available. Instead, install virtualenv.

On Linux, virtualenv is provided by your package manager:

Debian, Ubuntu

sudo apt-get install python-virtualenv

CentOS, Fedora

sudo yum install python-virtualenv

Arch sudo pacman -S python-virtualenv

If you are on Mac OS X or Windows, download get-pip.py, then:

sudo python2 Downloads/get-pip.py

sudo python2 -m

pip install virtualenv

On Windows, as an administrator:

\Python27\python.exe Downloads\get-pip.py

\Python27\python.exe -m pip install virtualenv

Advantages of Flask Framework:

- Extremely flexible
- Simple to learn and use

- Simple to learn and use
- Routing URLs is easy
- Small core and easily extensible
- Not async-friendly
- Limited features, support and documentation
- Lack of database/ORM/forms
- Truly limited in features

4.5 SAMPLE CODE:

Projectpython.py :

```
from flask import Flask, render_template,request,redirect,url_for
from datetime import datetime,date

import sqlite3
app=Flask(__name__)
e=""
p=""
@app.route('/home')
def home():
return render_template('home.html')
'"@app.route('/homepage')
def homepage():
return render_template('homepage.html')"'
@app.route('/aboutus')
def aboutus():
return render_template('aboutus.html')
@app.route('/contactus')
def contactus():
return render_template('contactus.html')


@app.route('/login')
def login():
return render_template('login.html')

@app.route('/register')
def register():
return render_template('customer_register.html')

@app.route('/result',methods = ['POST'])
def result():
conn=sqlite3.connect('ecommerce.db')
cur=conn.cursor()
m=request.form['name']
n=request.form['email']
o=request.form['pswd']
d=request.form['dr_no']
p=request.form['ph_no']
q=request.form['street']
r=request.form['city']
s=request.form['state']
t=request.form['pin']
cur.execute("insert into
customer(cus_name,cus_email,cus_pswd,cus_phn_no,cus_dr_no,cus_street,cus_city,cus_sta
```

```
te,cus_pin) values(?,?,?,?,?,?,?,?,?)",(m,n,o,p,d,q,r,s,t))
 conn.commit()
    return "sucessfull"

@app.route('/loginvalidate',methods=['POST'])
def loginvalidate():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    global e,p
    e=request.form['email']
    p=request.form['pswd']
    cur.execute("select count(*) from admin where ad_email=='"+e+"' and
ad_pswd=='"+p+"'")
    rows=cur.fetchone()
    if rows[0]==0:
        cur.execute("select count(*) from customer where cus_phn_no=='"+e+"' and
cus_pswd=='"+p+"'")
        res=cur.fetchone()
        if(res[0]==0):
            return "invalid user"
        else:
            cur.execute("delete from orders where or_cus_phn_no=='"+e+"' and
or_status=='pending' ")
            conn.commit()
            val=(e,date.today(),'pending')
            cur.execute("insert into orders(or_cus_phn_no,or_date,or_status) values(?,?,?)",val)
            conn.commit()
            cur.execute("select or_id from orders where or_cus_phn_no=='"+e+"' order by or_id
desc limit 1")
            res=cur.fetchone()
            global orderid
            orderid=res[0]
            return redirect(url_for('cuspage'))
    else:
        return render_template('admin_page.html'


@app.route('/customerpage')
def cuspage():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select cus_name from customer where cus_phn_no=='"+e+"' and
cus_pswd=='"+p+"'")
    name=cur.fetchone()
    return render_template('customer.html',name=name[0])


@app.route('/adminadd')
def adminadd():
    return render_template('admin_add.html')



@app.route('/additem', methods=['POST'])
def additem():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    pname=request.form['pn']
    pcat=request.form['cp']
    pimg=request.form['p3']
    pdesc=request.form['p4']
    pcost=request.form['p5']
    pqty=request.form['pq']
    val=(pname,pcat,pimg,pdesc,pcost,pqty)
    cur.execute("insert into product (pr_name,pr_cat_name,pr_img,pr_desc,pr_price,pr_qty)
values(?,?,?,?,?,?)",val)
    conn.commit()
    return "inserted"
```

```
@app.route('/updateform')
def updateform():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()

    cur.execute("select  pr_id from product  ")
    row=cur.fetchall()
    return render_template('updateitem.html',row=row)

@app.route('/adminupdate',methods=['POST'])
def adminupdate():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    pid=request.form['p1']
    cur.execute("select * from product where pr_id=='"+pid+"' ")
    rows=cur.fetchone()
    return render_template('admin_update.html',rows=rows)


@app.route('/updateitem',methods=['POST'])
def updateitem():
     conn=sqlite3.connect('ecommerce.db')
     cur=conn.cursor()
     pid=request.form['p1']
     pname=request.form['pn']
     pcat=request.form['cp']
     pimg=request.form['p3']
     pdesc=request.form['p4']
     pcost=request.form['p5']
     pqty=request.form['p6']
     cur.execute("update product set
pr_id='"+pid+"',pr_name='"+pname+"',pr_cat_name='"+pcat+"',pr_img='"+pimg+"',pr_desc
='"+pdesc+"',pr_price='"+pcost+"',pr_qty='"+pqty+"' where pr_id=='"+pid+"' ")
     conn.commit()
     return "updated items successfully"


@app.route('/editdelsaree')
def editdelsaree():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select  pr_id from product  ")
    row=cur.fetchall()
    return render_template('delitem.html',row=row)

@app.route('/delete_item', methods=['post'])
def delete_item():
    i=request.form['i']
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from product where pr_id=='"+i+"' ")
    rows=cur.fetchone()
    if(rows==None):
          return ("No records")
    else:
       return render_template("del_itemrecord.html",rows=rows)


@app.route('/delete_item_rec', methods=['post'])
def delete_item_rec():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    i=request.form['i']
    pname=request.form['pn']
    pcat=request.form['cp']
    pimg=request.form['p3']
    pdesc=request.form['p4']
    pcost=request.form['p5']
    pqty=request.form['p6']
    cur.execute("delete from product where pr_id == '"+i+"' ")
```

```
conn.commit()
cur.execute("select * from product")
    x=cur.fetchall()
    return render_template("item_rec_delete.html",x=x)

@app.route('/view_items/<catname>')
def view_items(catname):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from product where pr_name=='"+catname+"' ")
    x=cur.fetchall()
    return render_template("item_rec_delete.html",x=x)


@app.route('/adminorder')
def adminorder():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select or_id,pr_img,pr_name,pr_price,cart_quantity,or_date,or_status from
product p,orders o,cart c where o.or_id==c.cart_or_id and p.pr_id==c.cart_pr_id and
or_status=='placed' order by o.or_id desc")
    rows=cur.fetchall()
    return render_template('admin_order.html',rows=rows)


@app.route('/cancelorder/<iid>')
def cancelorder(iid):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("update orders  set or_status='cancelled' where or_id=='"+iid+"' ")
    conn.commit()
    return redirect(url_for('adminorder'))

@app.route('/finishorder/<iid>')
def finishorder(iid):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("update orders set or_status='finished' where or_id=='"+iid+"' ")
    conn.commit()
    return redirect(url_for('adminorder'))

@app.route('/getdetails/<iid>')
def getdetails(iid):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from doordetails where do_or_id=='"+iid+"' ")
    rows=cur.fetchone()
    return render_template('view_door_delivery.html',rows=rows)

@app.route('/adminfinish/')
def adminfinish():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select or_id,pr_img,pr_name,pr_price,cart_quantity,or_date,or_status from
product p,orders o,cart c where o.or_id==c.cart_or_id and p.pr_id==c.cart_pr_id and
or_status=='finished' order by o.or_id desc")
    rows=cur.fetchall()
    return render_template('admin_finish.html',rows=rows)

@app.route('/admincancel')
def admincancel():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select or_id,pr_img,pr_name,pr_price,cart_quantity,or_date,or_status from
product p,orders o,cart c where o.or_id==c.cart_or_id and p.pr_id==c.cart_pr_id and
or_status=='cancelled' order by o.or_id desc")
    rows=cur.fetchall()
    return render_template('admin_finish.html',rows=rows)
```

```
######
@app.route('/product/<catname>')
def product(catname):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from product where pr_name == '"+catname+"' ")
    x=cur.fetchall()
    return render_template('itemlist.html',x=x)


@app.route('/productquantityform/<ptid>')
def productquantityform(ptid):
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from product where pr_id== '"+ptid+"' ")
    x=cur.fetchone()
    return render_template('cart.html',rows=x)


@app.route('/addtocart',methods=['POST'])
def addtocart():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cpid=request.form['p1']
    cq=request.form['q']
    cat=request.form['cat']
    cur.execute("select count(*) from cart where cart_or_id=='"+str(orderid)+"' and
cart_pr_id=='"+cpid+"'")
    x=cur.fetchone()
    if(x[0]==0):
        cur.execute("insert into cart (cart_or_id,cart_pr_id,cart_quantity) values
(?,?,?)",(orderid,cpid,cq))
    else:
        cur.execute("update cart set cart_quantity=='"+cq+"' where
cart_or_id=='"+str(orderid)+"' and cart_pr_id=='"+cpid+"'")

    conn.commit()
    cur.execute("select * from product where pr_name == '"+cat+"'")
    x=cur.fetchall()
    return render_template('itemlist.html',x=x)


@app.route('/viewcart')
def viewcart():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select
pr_id,pr_name,pr_cat_name,pr_img,pr_price,cart_quantity,pr_price*cart_quantity as Total
from product p,cart c where cart_or_id=='"+str(orderid)+"' and cart_pr_id==pr_id ")
    rows=cur.fetchall()
    total=0
    for row in rows:
        total=total+row[6]
    return render_template('viewcart.html',rows=rows,total=total)


@app.route('/doordelivery')
def doordelivery():
    return render_template('door_delivery.html',orderid=orderid)


@app.route('/finish',methods=['POST'])
def finish():
    global orderid
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    o=request.form['o']
    n=request.form['n']
    m=request.form['m']
    a=request.form['a']
```

```
cur.execute("INSERT INTO doordetails (do_or_id,do_name,do_ph_no,do_address)
VALUES (?,?,?,?)" ,(o,n,m,a))
   cur.execute("update orders set or_status='placed' where or_id=='"+str(orderid)+"' ")
    conn.commit()
    val=(e,date.today(),'pending')
    cur.execute("insert into orders(or_cus_phn_no,or_date,or_status) values(?,?,?)",val)
    conn.commit()
    cur.execute("select or_id from orders where or_cus_phn_no=='"+e+"' order by or_id desc
limit 1")
    res=cur.fetchone()
    orderid=res[0]
    return "placed successfully"


@app.route('/placedorders')
def placedorders():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()    cur.execute("select
or_id,pr_img,pr_name,pr_price,cart_quantity,or_date,or_status from product p,customer
cu,orders o,cart c where o.or_id==c.cart_or_id and p.pr_id==c.cart_pr_id and
or_status=='placed' and cu.cus_phn_no==o.or_cus_phn_no and cu.cus_phn_no=='"+e+"'
order by o.or_id desc")
    rows=cur.fetchall()
    return render_template('placedorders.html',rows=rows)


@app.route('/usercancelorder/<iid>')
def usercancelorder(iid):
    conn=sqlite3.connect('ecommerce.db')

    cur=conn.cursor()
    cur.execute("update orders set or_status='cancelled' where or_id=='"+iid+"' ")
    conn.commit()
    return redirect(url_for('placedorders'))

@app.route('/ucancelorders')
def ucancelorders():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select or_id,pr_img,pr_name,pr_price,cart_quantity,or_date,or_status from
product p,orders o,cart c,customer cu where o.or_id==c.cart_or_id and
p.pr_id==c.cart_pr_id and cu.cus_phn_no==o.or_cus_phn_no and or_status=='cancelled'
and  cus_phn_no=='"+e+"' order by o.or_id desc")
    rows=cur.fetchall()
    return render_template('admin_finish.html',rows=rows)
#######

@app.route('/fetchuserdetails')
def fetchuserdetails():
    conn=sqlite3.connect('ecommerce.db')
    cur=conn.cursor()
    cur.execute("select * from customer where cus_phn_no=='"+e+"'")
    rows=cur.fetchone()
    return render_template('updateprofile.html',rows=rows

@app.route('/savedetails',methods=['POST'])
def savedetails():
     conn=sqlite3.connect('ecommerce.db')
     cur=conn.cursor()
     name=request.form['n1']
     email=request.form['n2']
     passwd=request.form['n3']
     address=request.form['n4']
     street=request.form['n5']
     city=request.form['n6']
     state=request.form['n7']
     pincode=request.form['n8']
     phoneno=request.form['n9']
     cur.execute("update customer set
cus_name='"+name+"',cus_email='"+email+"',cus_pswd='"+passwd+"',cus_dr_no='"+addre
```

```
ss+"',cus_street='"+street+"',cus_city='"+city+"',cus_state='"+state+"',cus_pin='"+pincode+"
',cus_phn_no='"+phoneno+"' where cus_phn_no=='"+e+"' ")
     conn.commit()
     return "updated successfully"


@app.route('/display')
def list():
   con = sqlite3.connect("ecommerce.db")
   con.row_factory = sqlite3.Row

   cur = con.cursor()
   cur.execute("select * from customer")

   rows = cur.fetchall();
   return render_template("display.html",rows = rows)

orderid=""
e=""


if __name__ == '__main__':
   app.run(debug = True)
```

5. TESTING AND TEST CASES

5.1 INTRODUCTION

One of the purposes of the testing is to validate and verify the system. Verification means checking the system to ensure that it is doing what the function is supposed to do and Validation means checking to ensure that system is doing what the user wants it to do.

No program or system design is perfect; communication between the user and the designer is not always complete or clear, and time is usually short. The result is errors and more errors. Theoretically, a newly designed system should have all the pieces in working order, but in reality, each piece works independently. Now is the time to put all the pieces into one system and test it to determine whether it meets the user's requirements. This is the best chance to detect and correct errors before the system is implemented. The purpose of system testing is to consider all the likely variations to which it will be subjected and then push the system to its limits. If we implement the system without proper testing then it might cause the problems.

1. Communication between the user and the designer.
2. The programmer's ability to generate a code that reflects exactly the system specification.
3. The time frame for the design.

Theoretically, a new designed system should have all the pieces in working order, but in reality, each piece works independently. Now is the time to put all the pieces into one system and test it to determine whether it meets the requirements of the user. The process of system testing and the steps taken to validate and prepare a system for final implementation are:

5.2 TESTING STRATEGIES

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at differing phases of software development are:

5.3 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration.

5.4 INTEGRATION TESTING

In Integration Testing, the different units of the system are integrated together to form the complete system and this type of testing checks the system as whole to ensure that it is doing what is supposed to do. The testing of an integrated system can be carried out top-down, bottom-up, or big-bang. In this type of testing, some parts will be tested with white box testing and some with black box testing techniques.

This type of testing plays very important role in increasing the systems productivity. We have checked our system by using the integration testing techniques.

VERIFICATION TESTING

In Verification Testing, the different units of system are verified together whether the exact output to the system is existing or not.

VALIDATION TESTNG

In Validation Testing, software validation is achieved through a series of black-box tests that demonstrate conformity with the requirements. After each validation test case has been conducted, one or two possible conditions exists:

1.The function or performance characteristics conform to specification and are accepted.

2.A deviation from specification is uncovered and a deficiency list is created.

5.5 SYSTEM TESTING

Apart from testing the system to validate the functionality of software against the requirements, it is also necessary to test the non-functional aspect of the system. Some examples of non-functional tools include tests to check performance, data security, usability/user friendliness, volume, load/stress that we have used in our system to test the various modules. System testing consists of the following steps:

1. Program(s) testing.
2. String testing.
3. System testing.
4. System documentation.
5. User acceptance testing.

5.6 TEST CASES

Testcase Table:

S.NO	Test case	Input	Expected behaviour	Actual behaviour	Result
1	User registration	Enter the fields	Home page	Homepage	success
2	User registration	If we miss entry fields	Fill the entries	Please fill out this field	success
3	User login	Enter valid credentials	Redirecting to customer page	Customer page	success
4	User login	Enter invalid credentials	Invalid User	Invalid user	Success
5	Admin added products	Added items	it will be seen in customerpage	customerpage	success
6	Adding the item to cart.	Enter Quantity of item and proceed.	Open cart Page	Cart Page	success
7	Placing the Order	Enter Door Details in CustomerPage	Placed Succesfully at Admin	Order placed at Admin Page	success

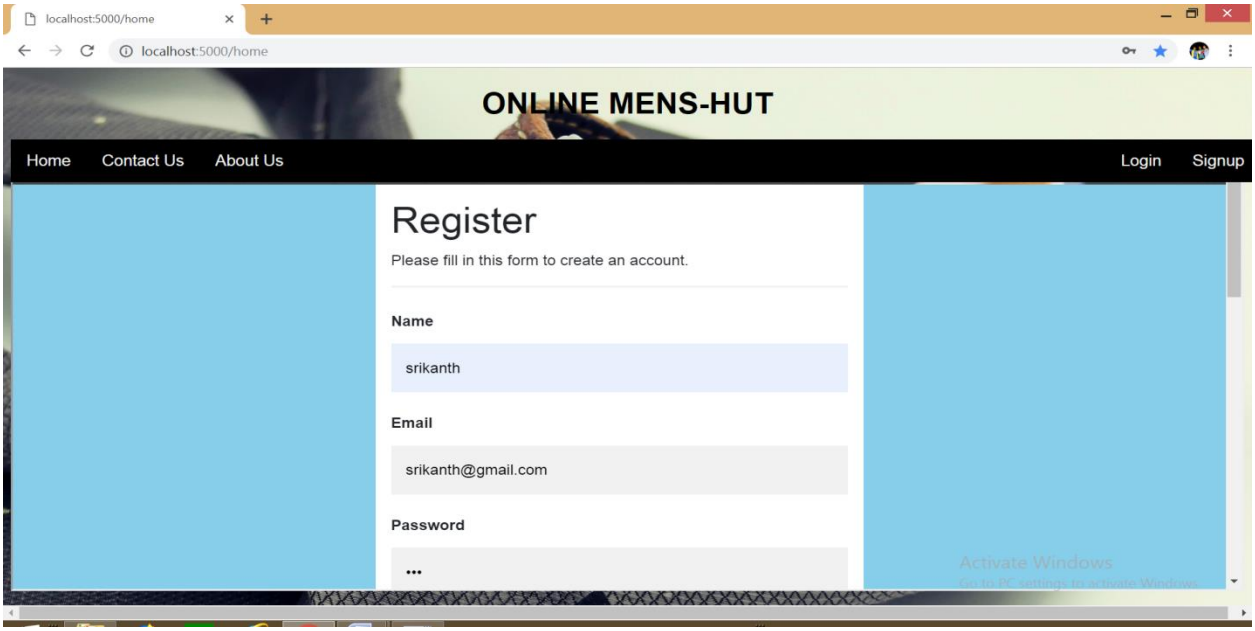
TEST CASE 1:

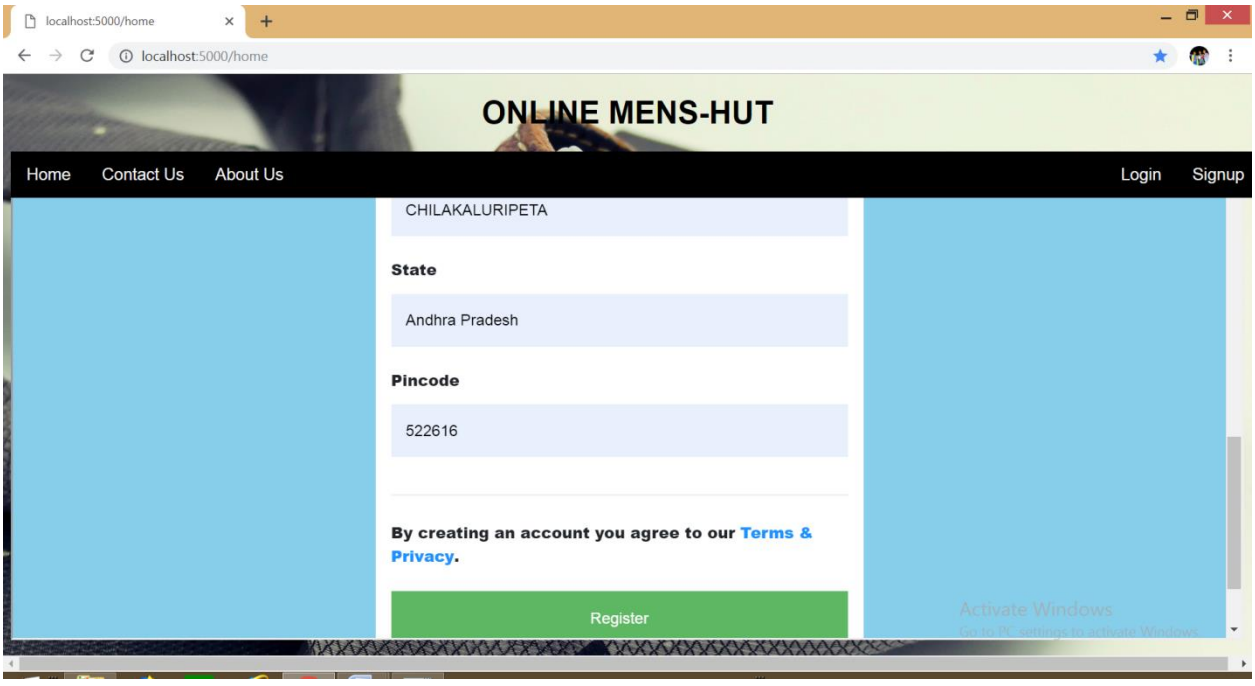
Input : Valid Data For Registration

Expected Behaviour : Home Page

Actual Behaviour : Home Page

Status : Success





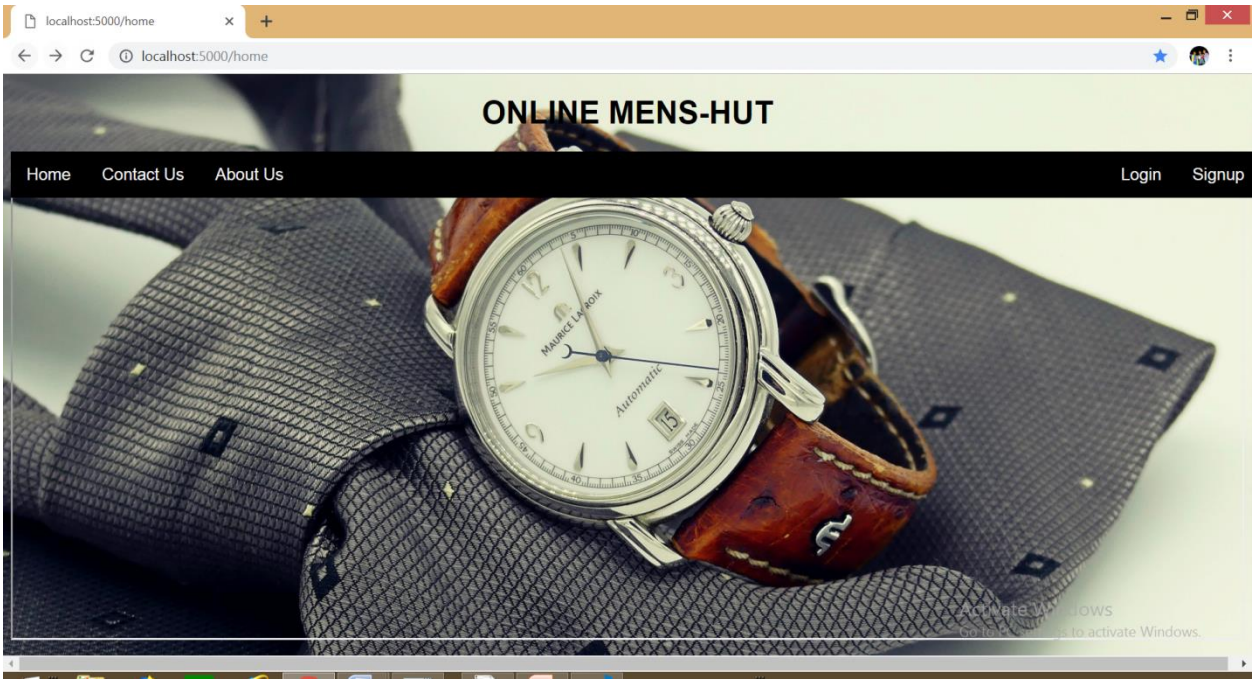


Figure1: 5.6.1: Test case for Valid data for registration

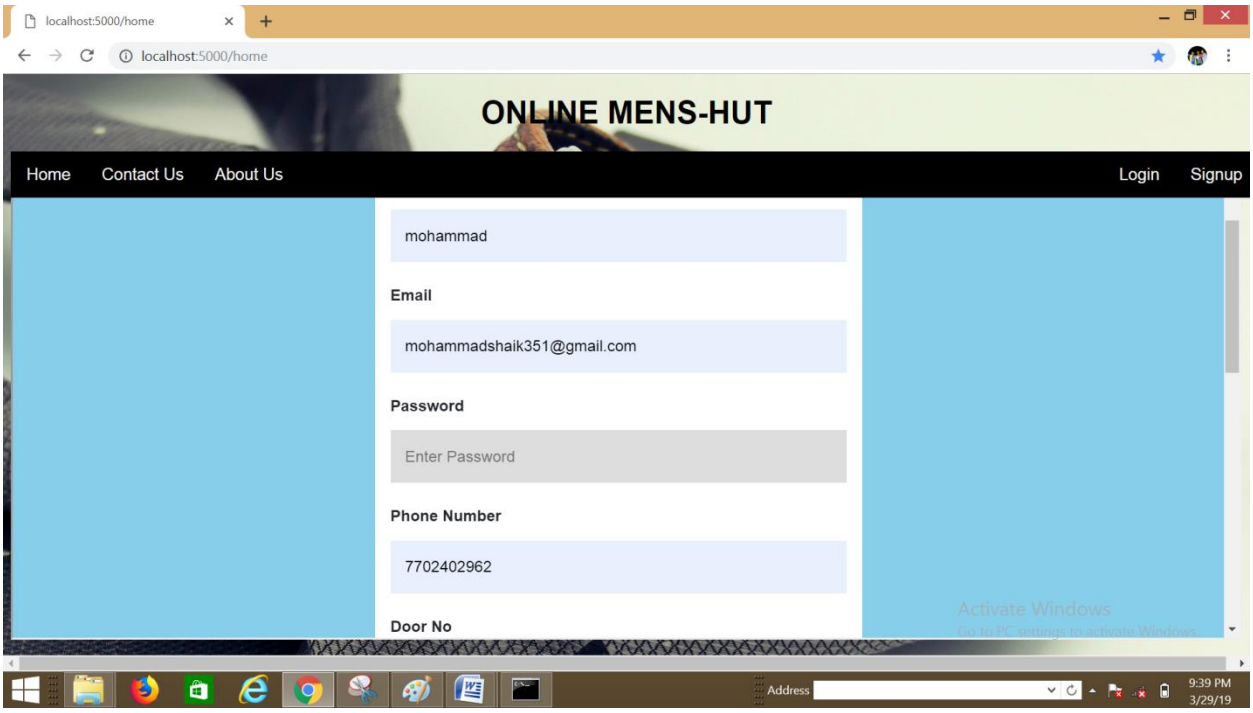
TEST CASE 2:

Input : Valid data User Registration

Expected Behaviour : Fill the entries

Actual Behaviour : Please fill out this field

Status : Success



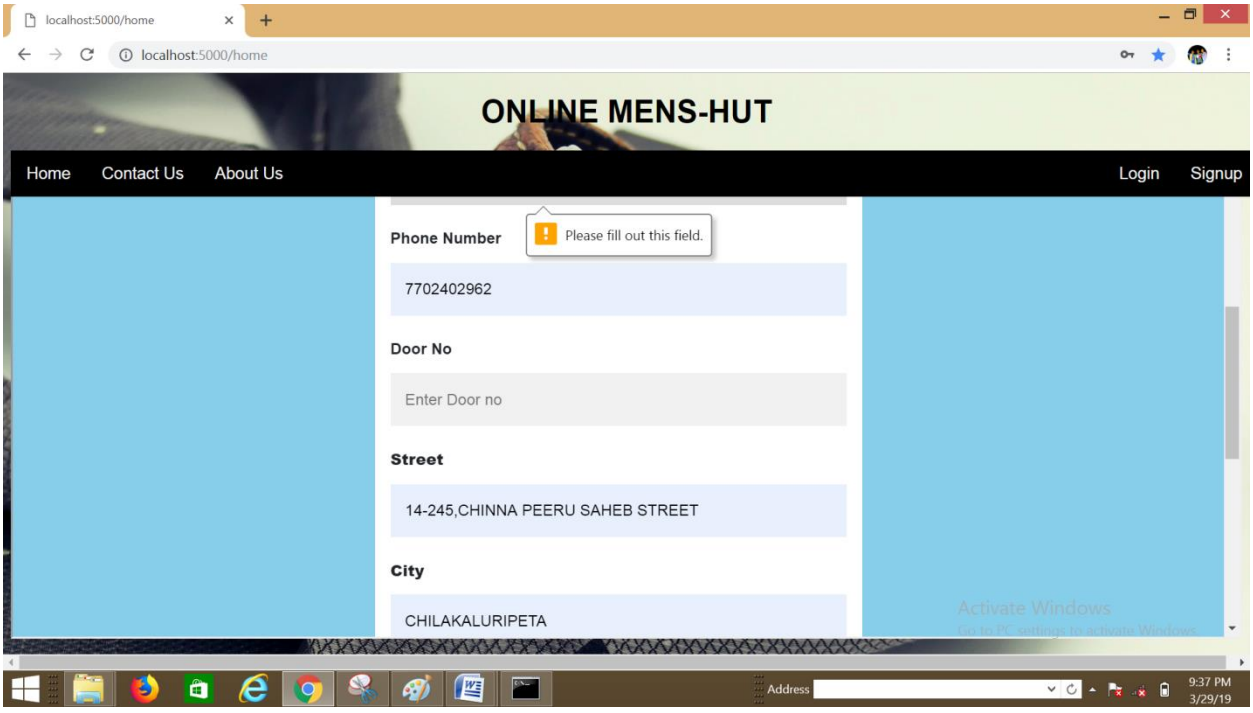


Figure2: 5.6.2: Test case for Valid fill data for Registration

TEST CASE 3:

Input : Valid data for User Login

Expected Behaviour : Customer Page

Actual Behaviour : Customer Page

Status : Success

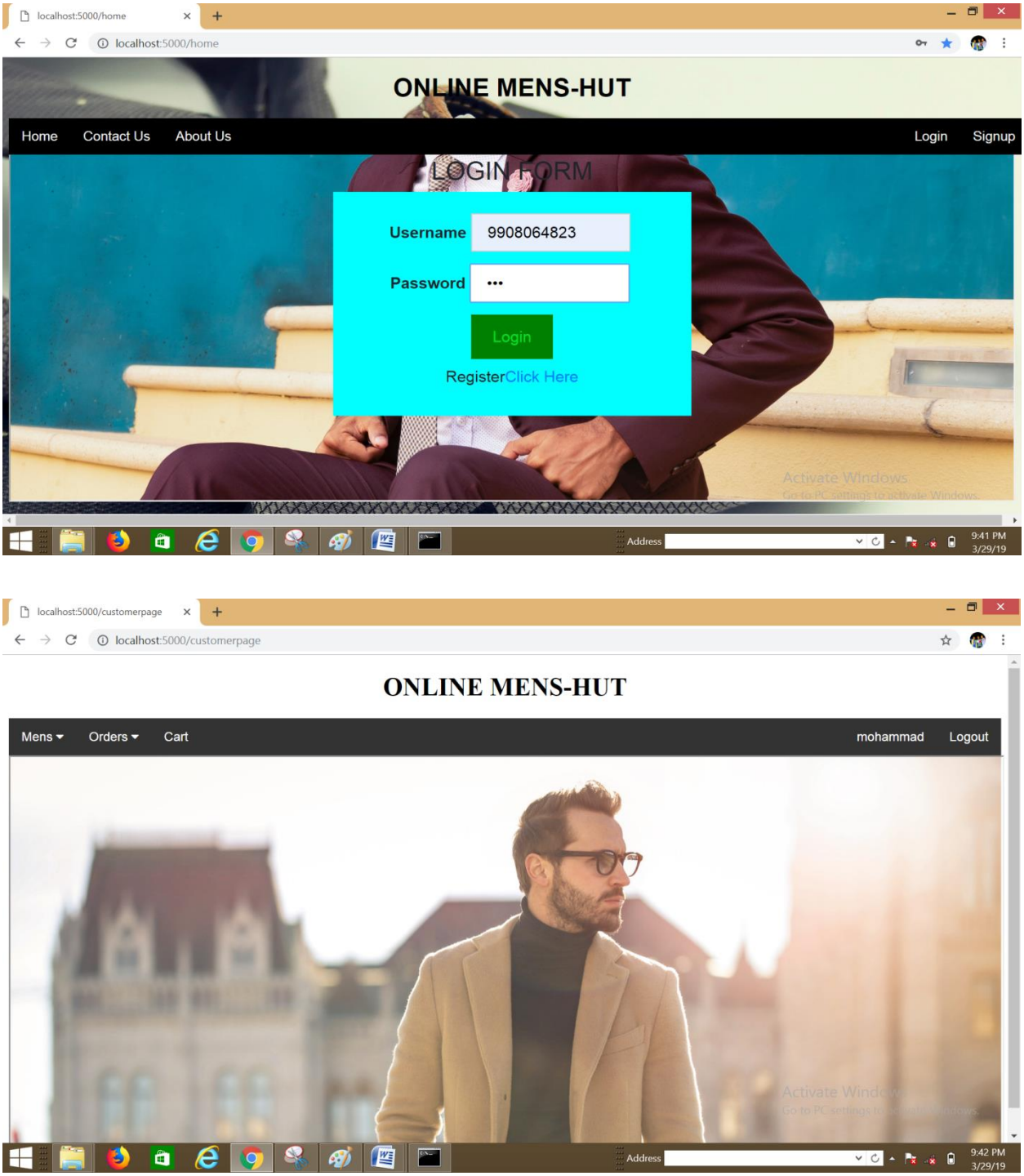


Figure3: 5.6.3: Test case for Valid Data for User Login

TEST CASE 4:

Input : Invalid credentials For User Login

Expected Behaviour : Invalid UserName

Actual Behaviour : Invalid User

Status : Success

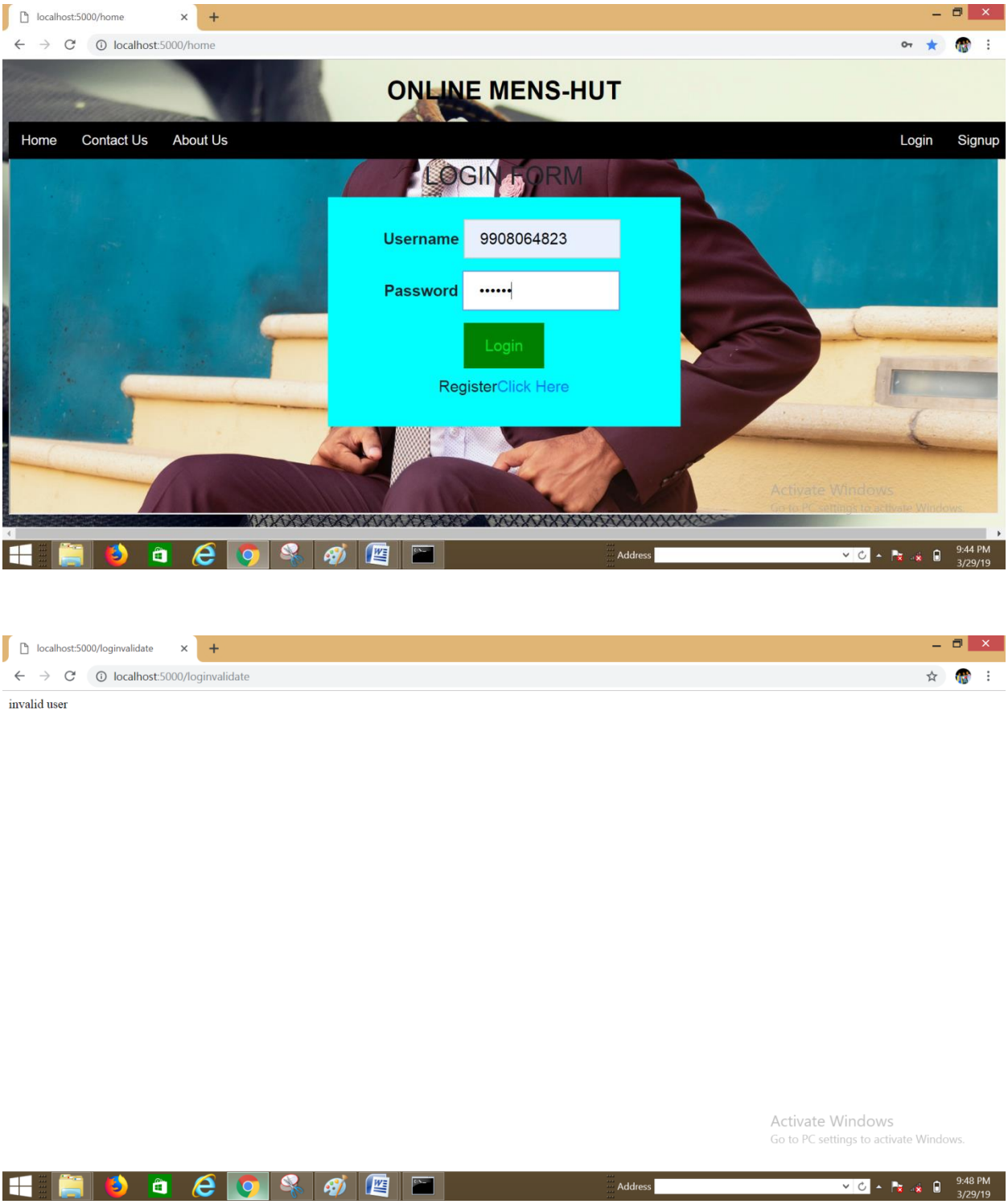


Figure4: 5.6.4: Test case for Invalid User Id for Login

TEST CASE 5:

Input : Products added by Admin

Expected Behaviour : User can view in Customer Page

Actual Behaviour : User can view in Customer Page

Status : Success

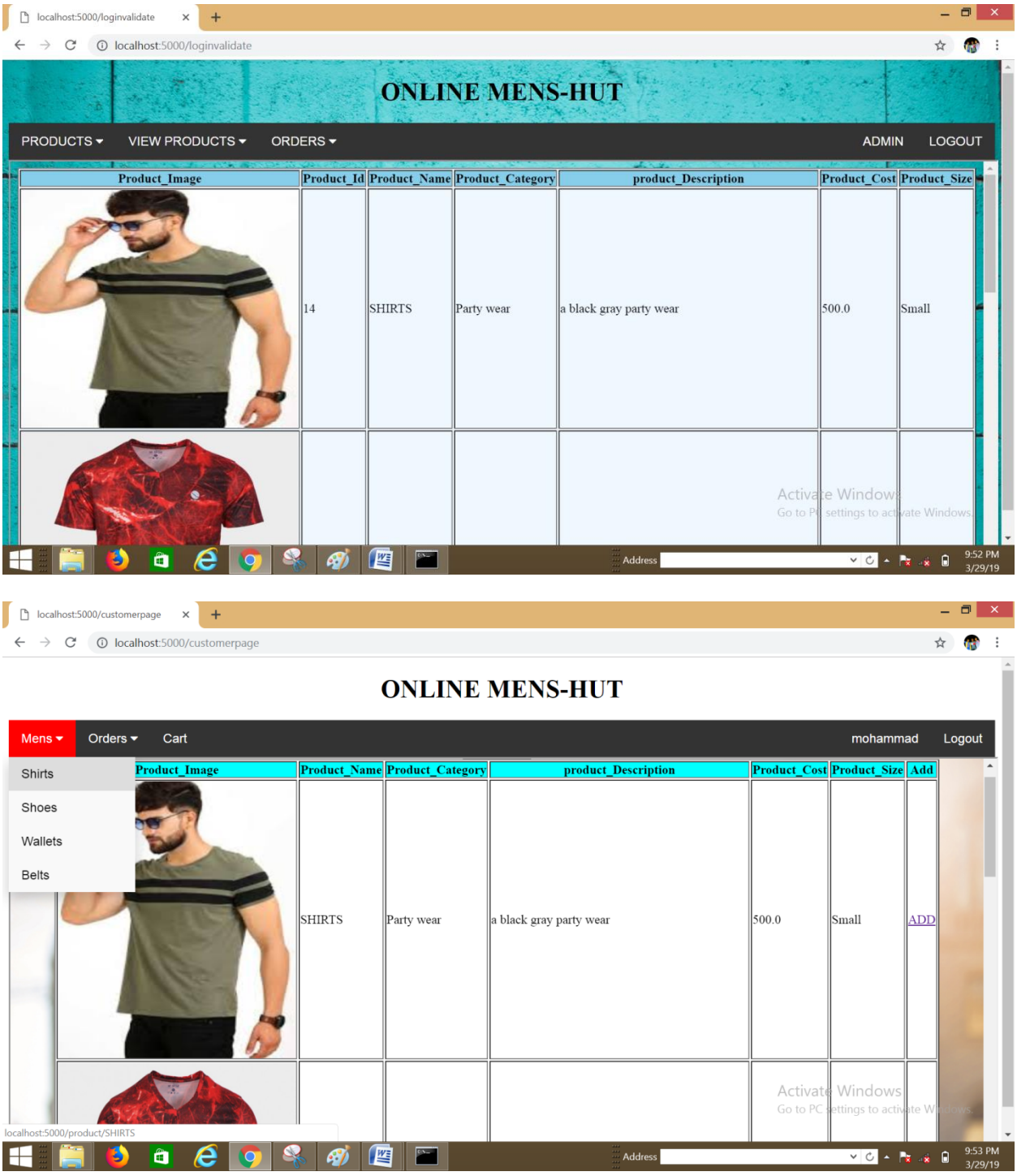


Figure5: 5.6.5: Test case for Products added by Admin can be viewed in User Page

TEST CASE 6:

Input : Valid Quantity for Cart Details

Expected Behaviour : Add to Cart Successful

Actual Behaviour : Add to Cart Successful

Status : Success

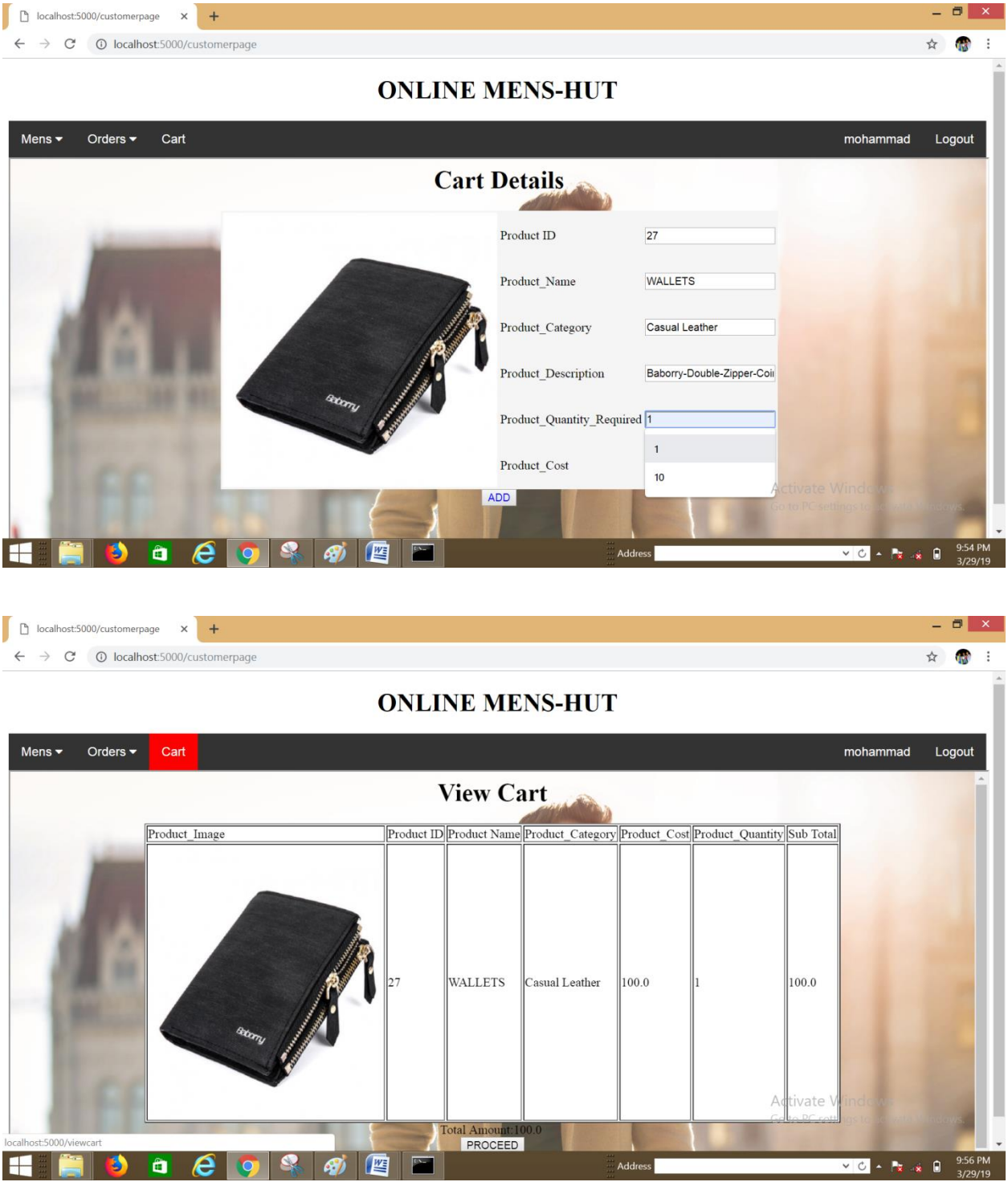


Figure6: 5.6.6: Test case Valid Quantity for Cart Details

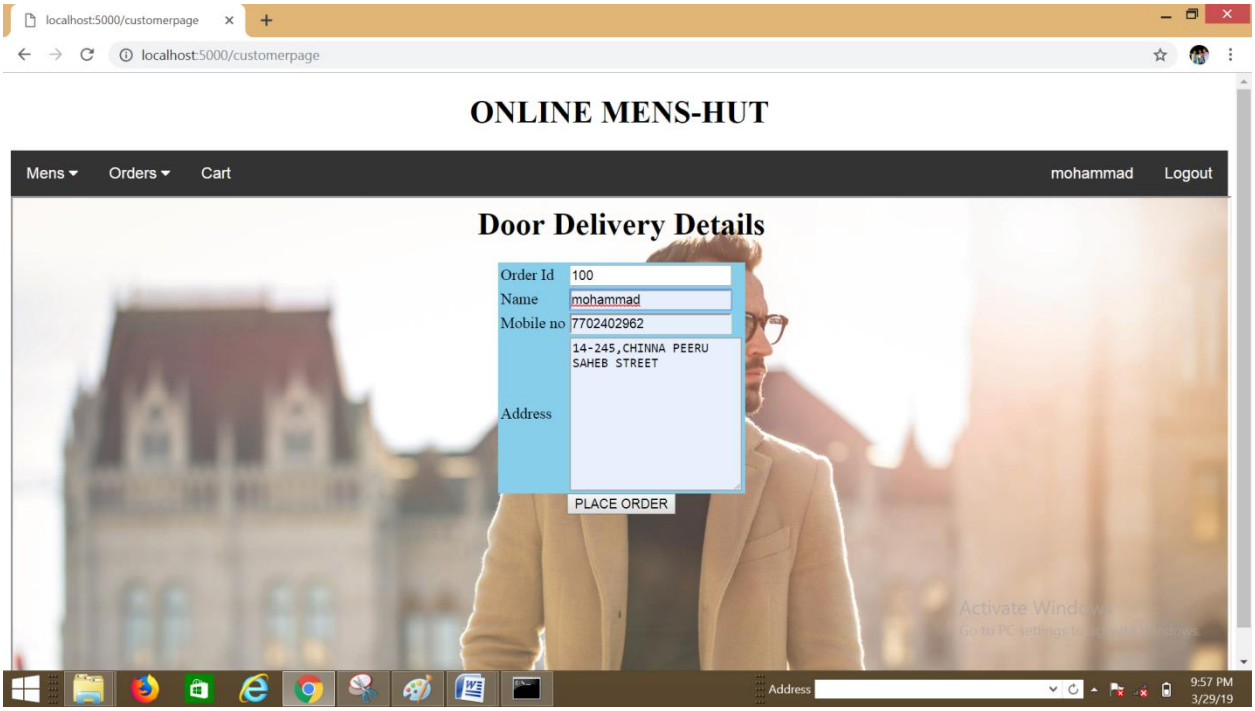
TEST CASE 7:

Input : Placing the order

Expected Behaviour : Order placed can be viewed in Adminpage

Actual Behaviour : Placed order Successfully

Status : Success



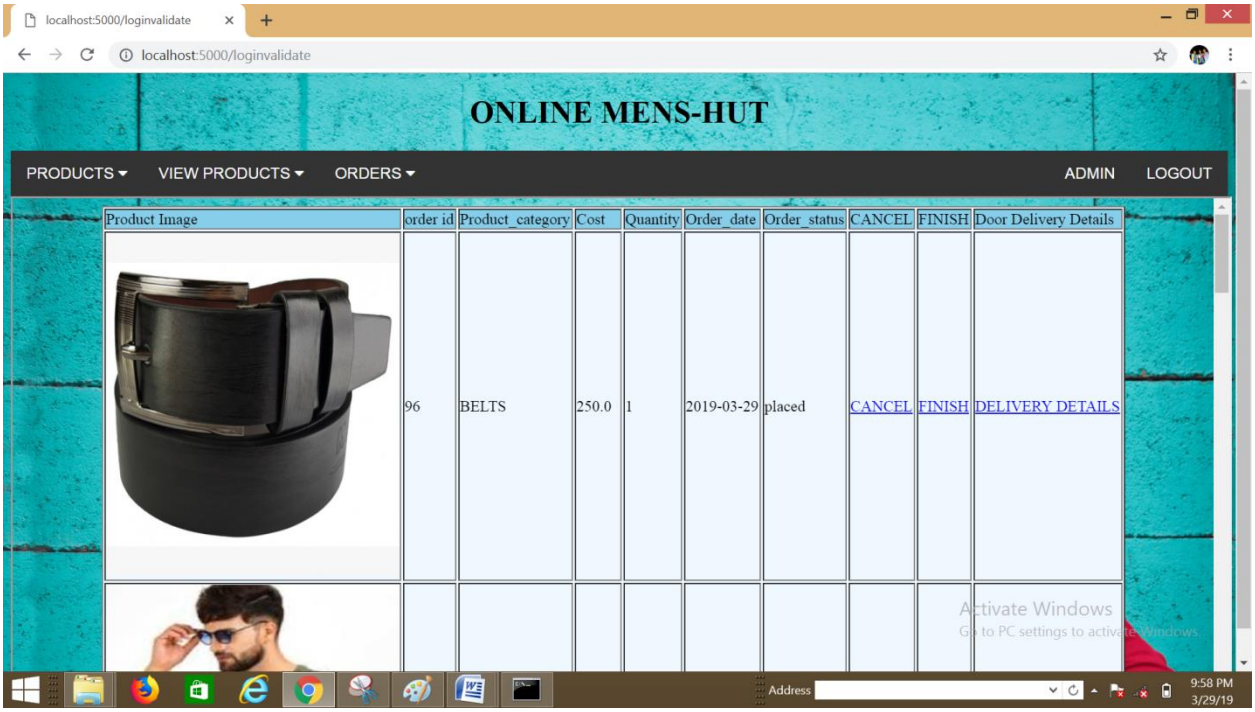


Figure7: 5.6.7: Test case for Placing the order

6.OUTPUT SCREENS

1. HOME PAGE:

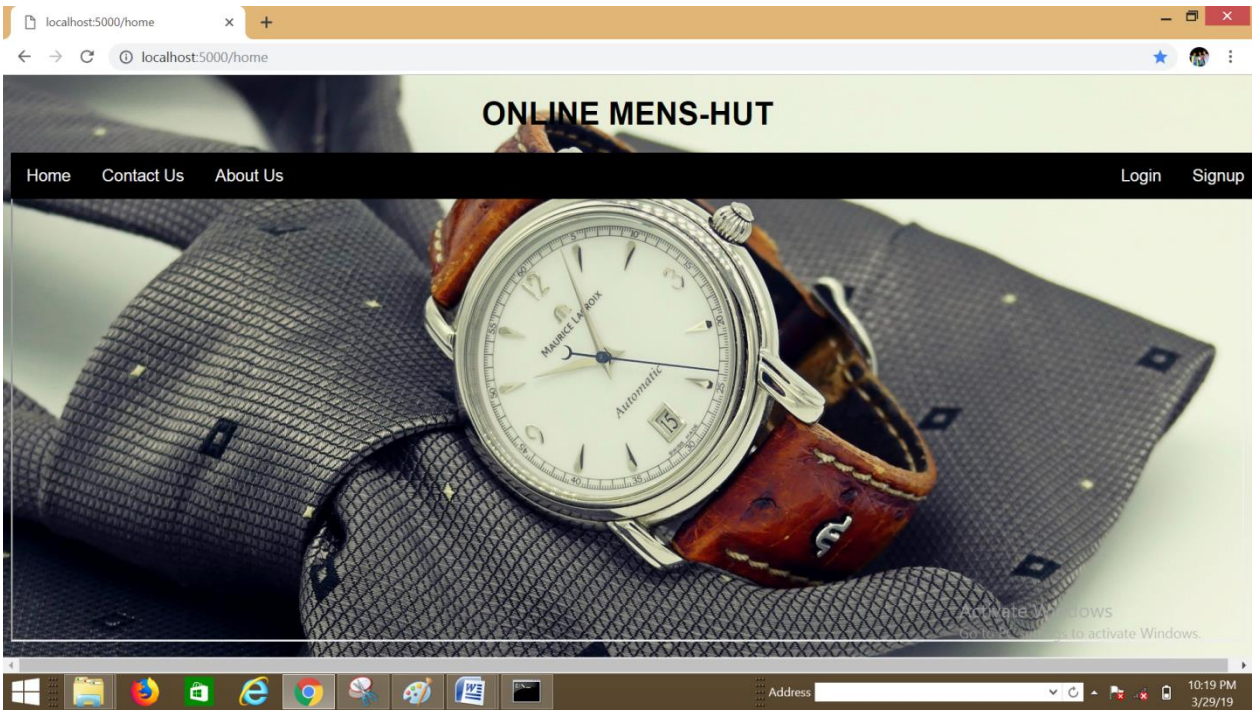


Figure11: 6.1: Home Page

2. CONTACT US PAGE:

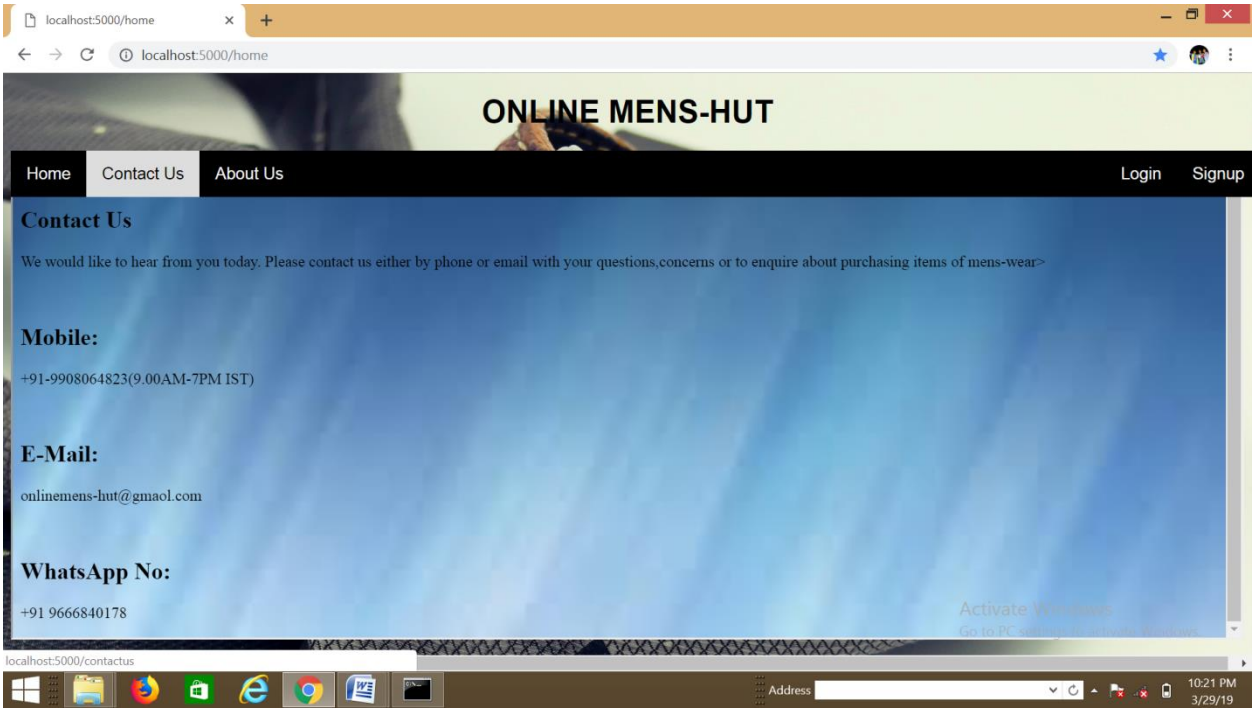


Figure12: 6.2: Contact Us Page

3.ABOUT US PAGE

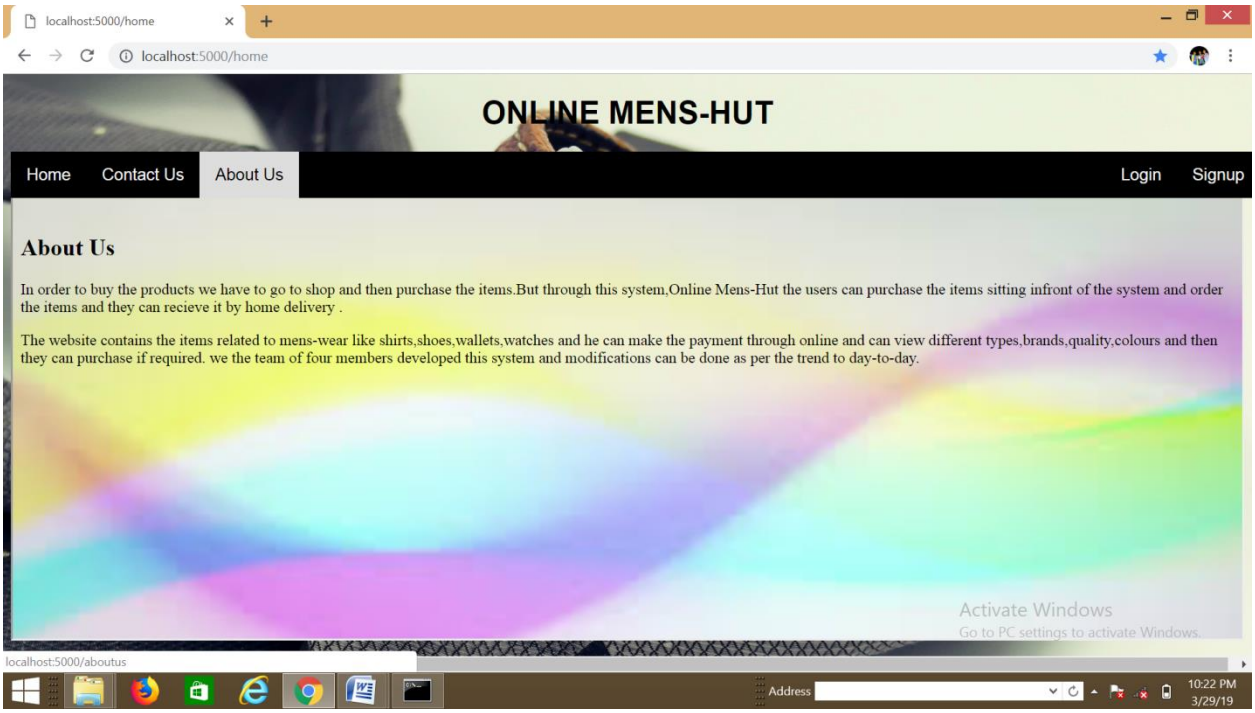
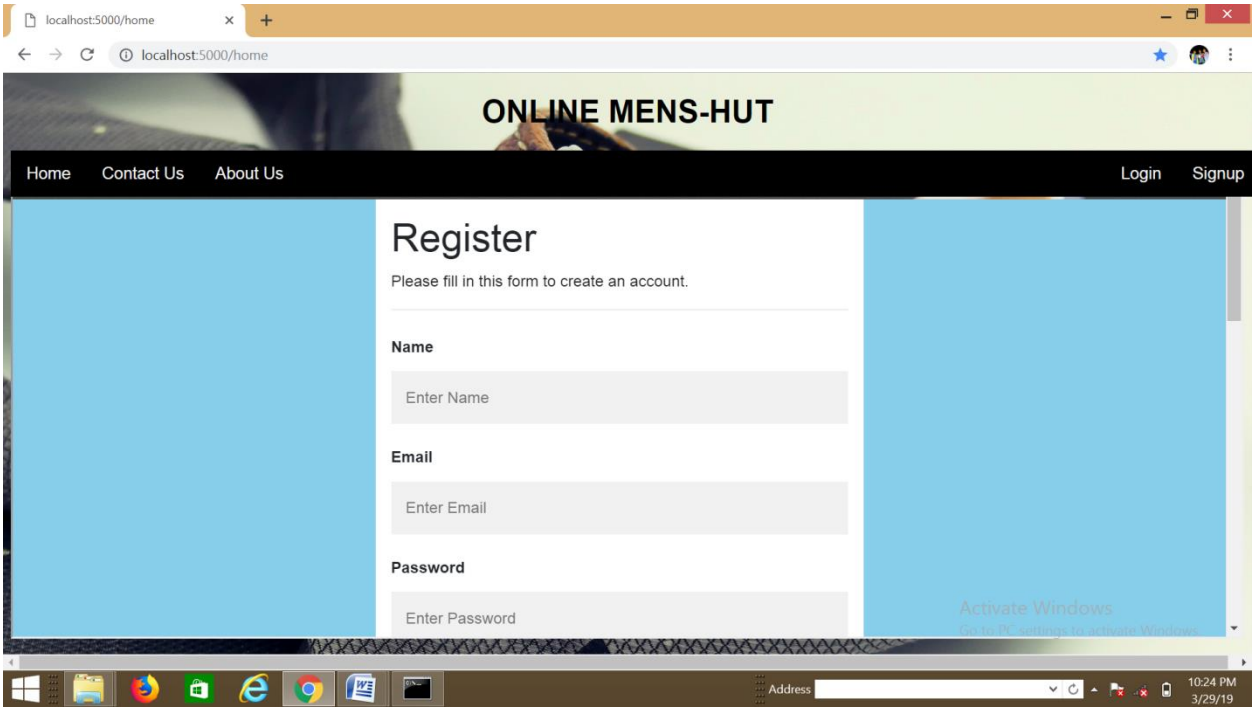


Figure13: 6.3:Aboutt Us Page

4. REGISTRATION PAGE:



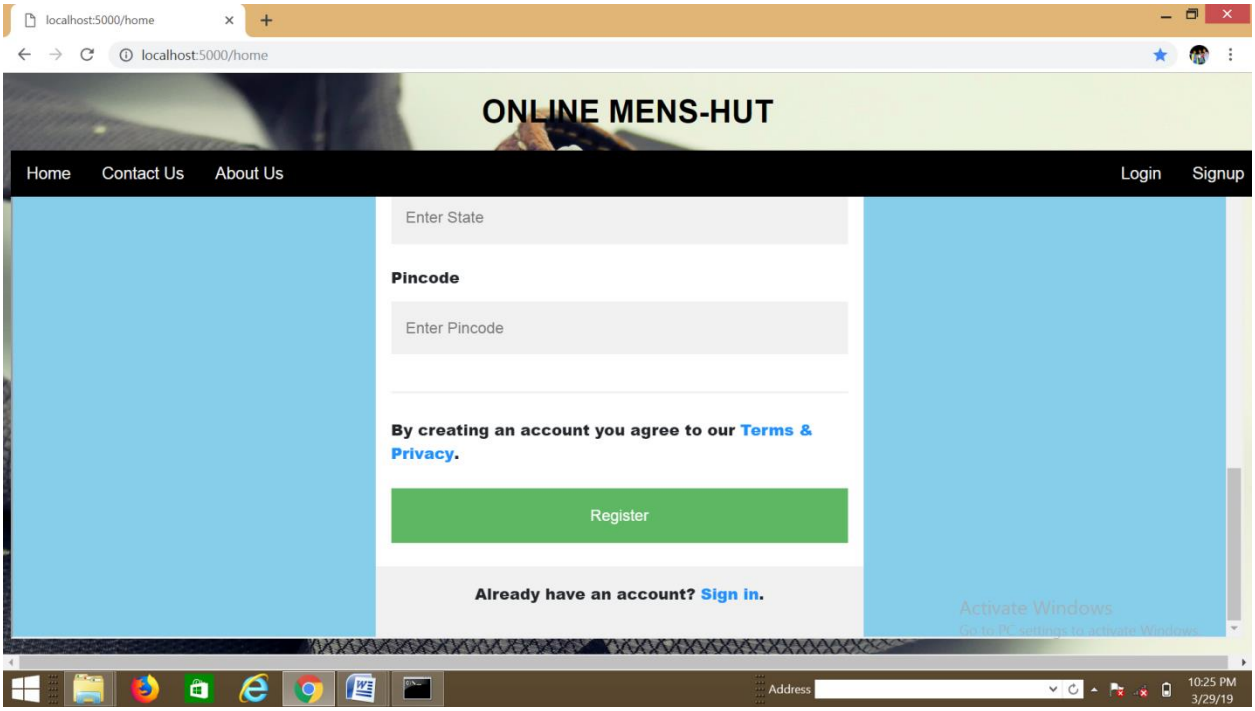


Figure14: 6.4: Registration Page

5. LOGIN PAGE:

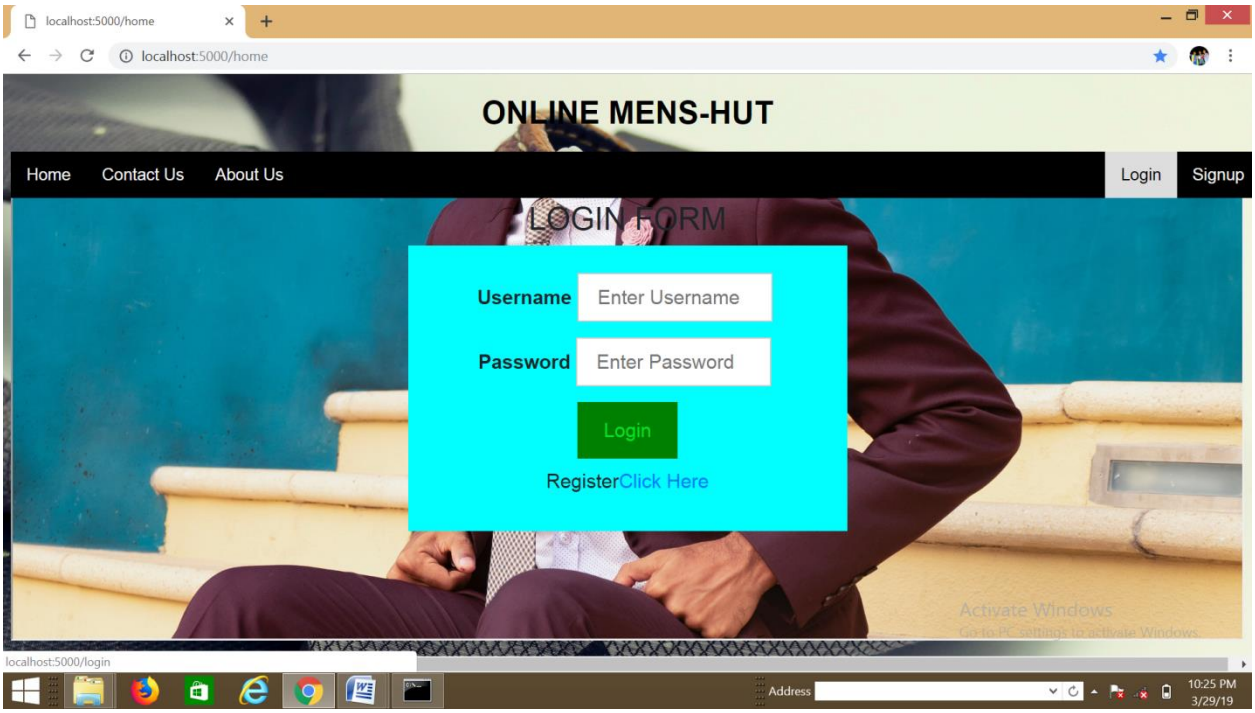


Figure15: 6.5: Login Page

6.CUSTOMER PAGE:

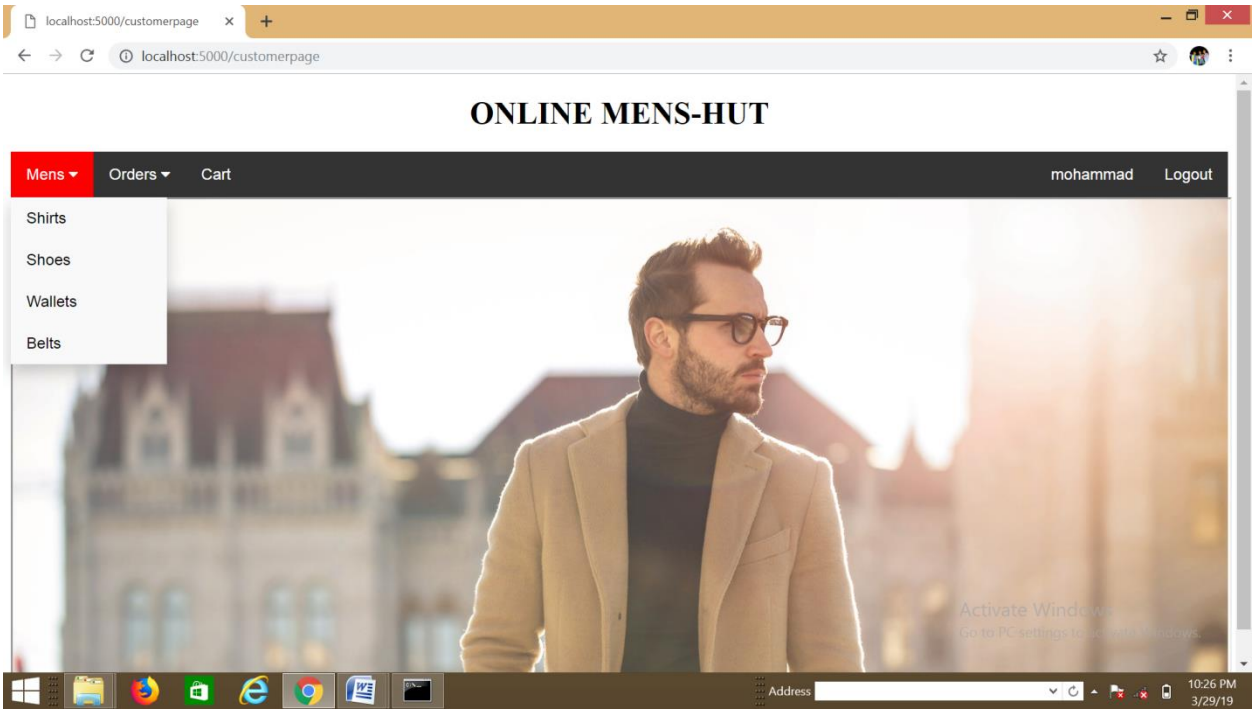
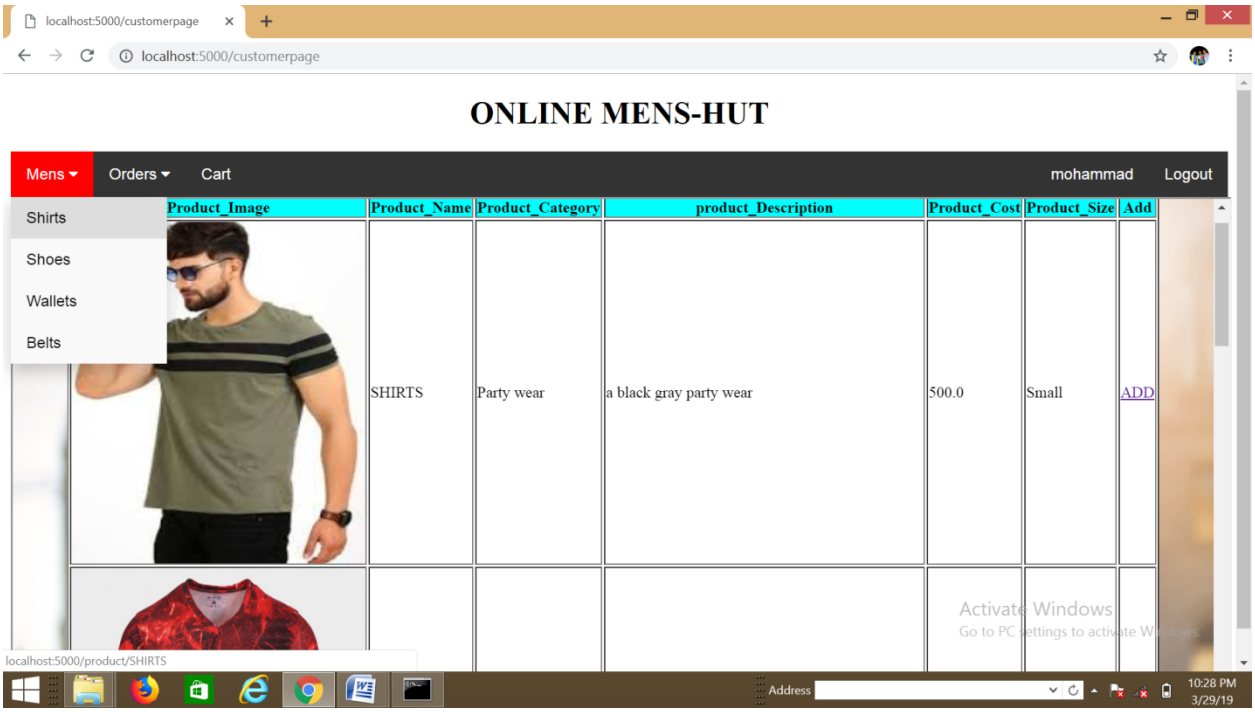


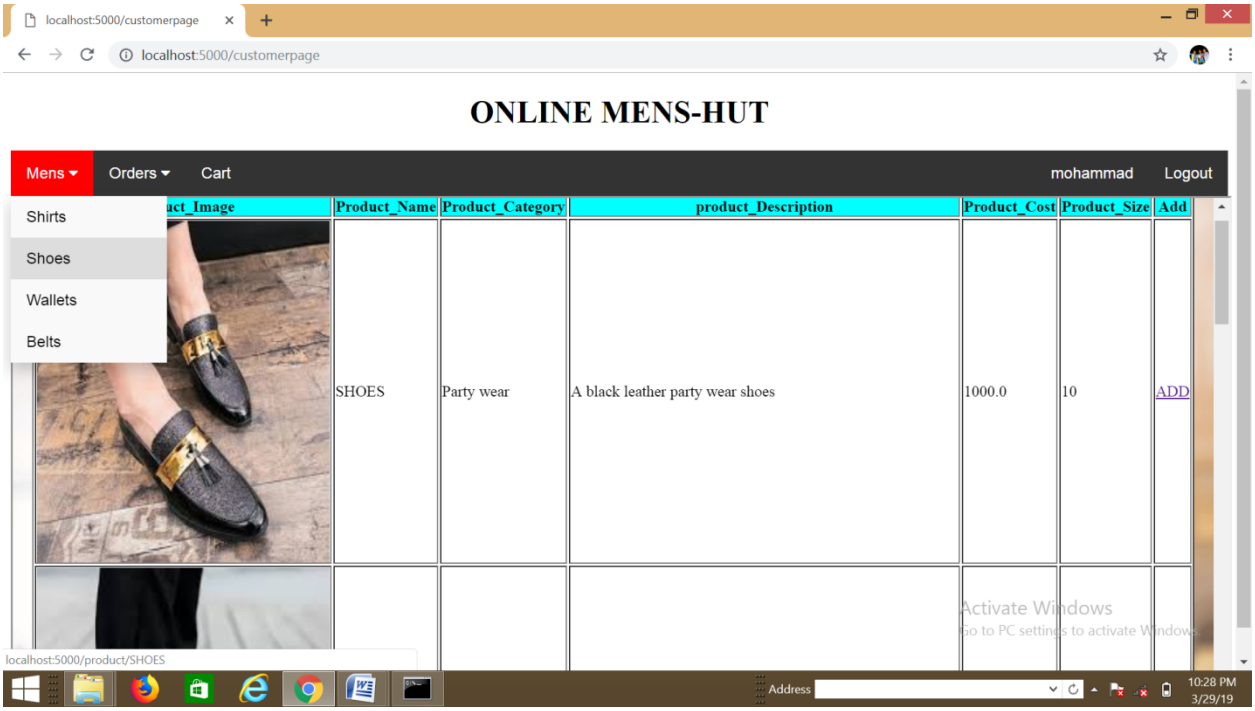
Figure16: 6.6: Customer Page

7. VIEW PRODUCTS PAGE:

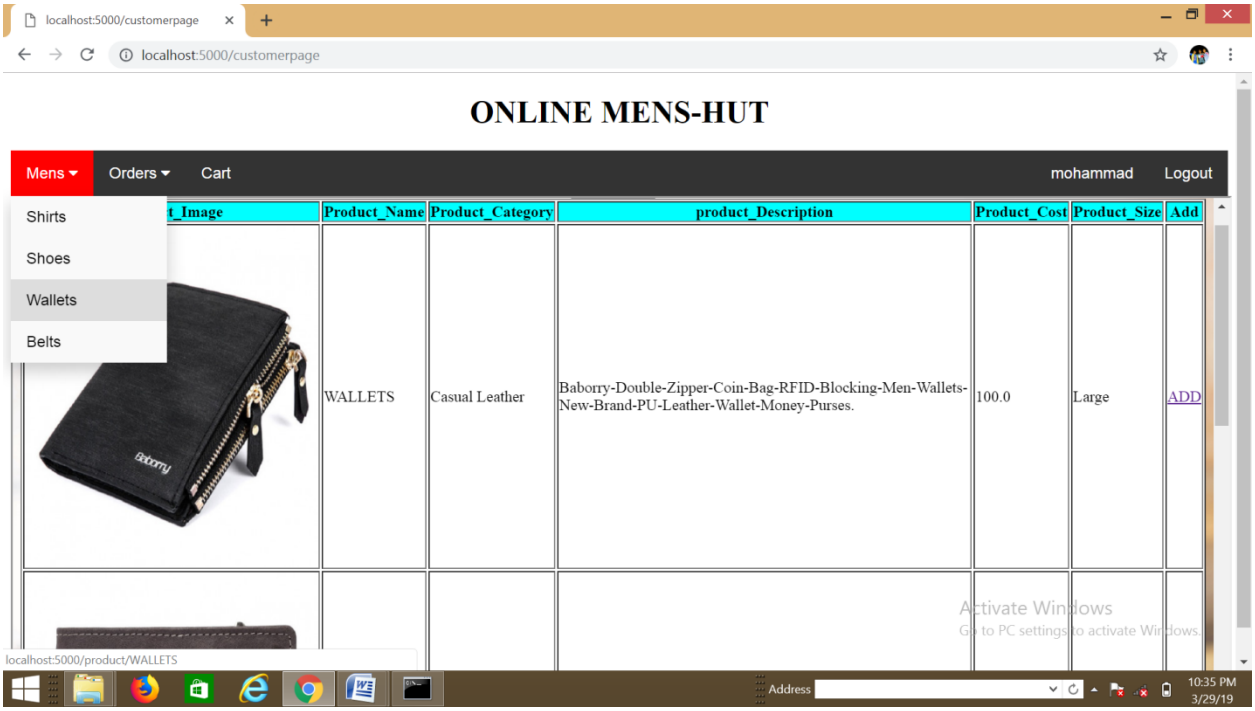
7.1 SHIRTS PAGE:



7.2 SHOES PAGE:



7.3 WALLETS PAGE:



7.4 BELTS PAGE

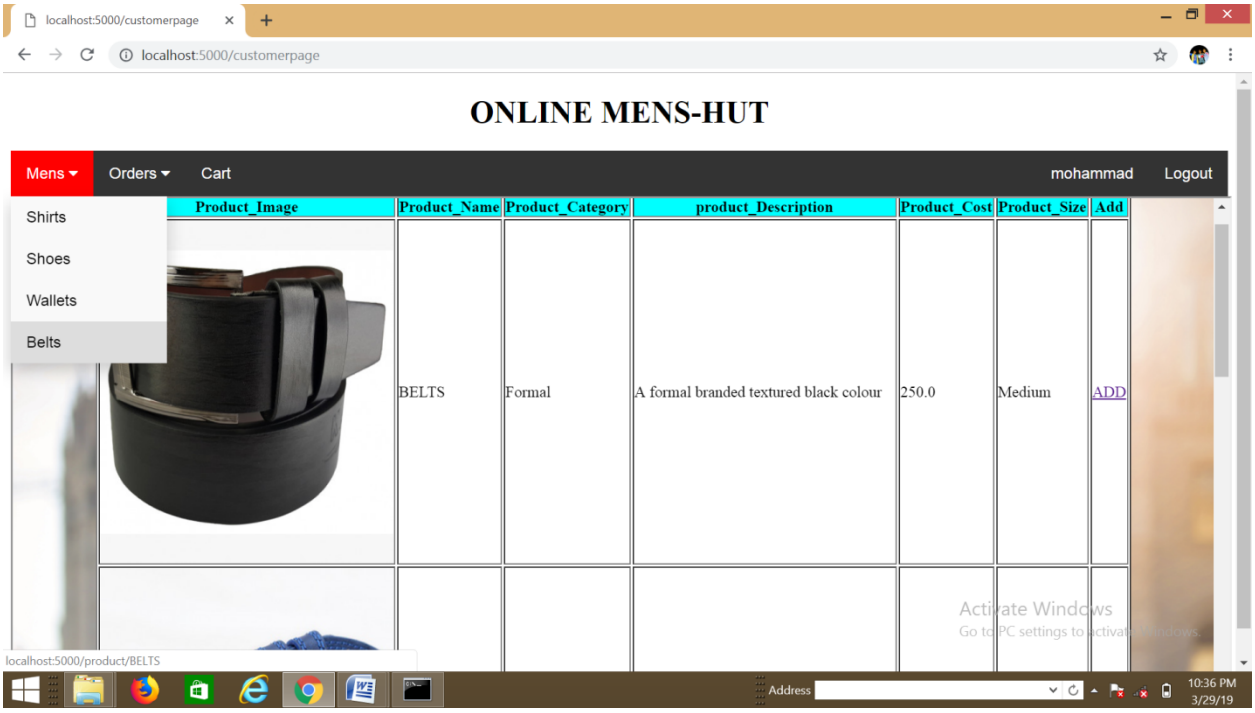


Figure17: 6.7: View Products Page

8. CART DETAILS PAGE:

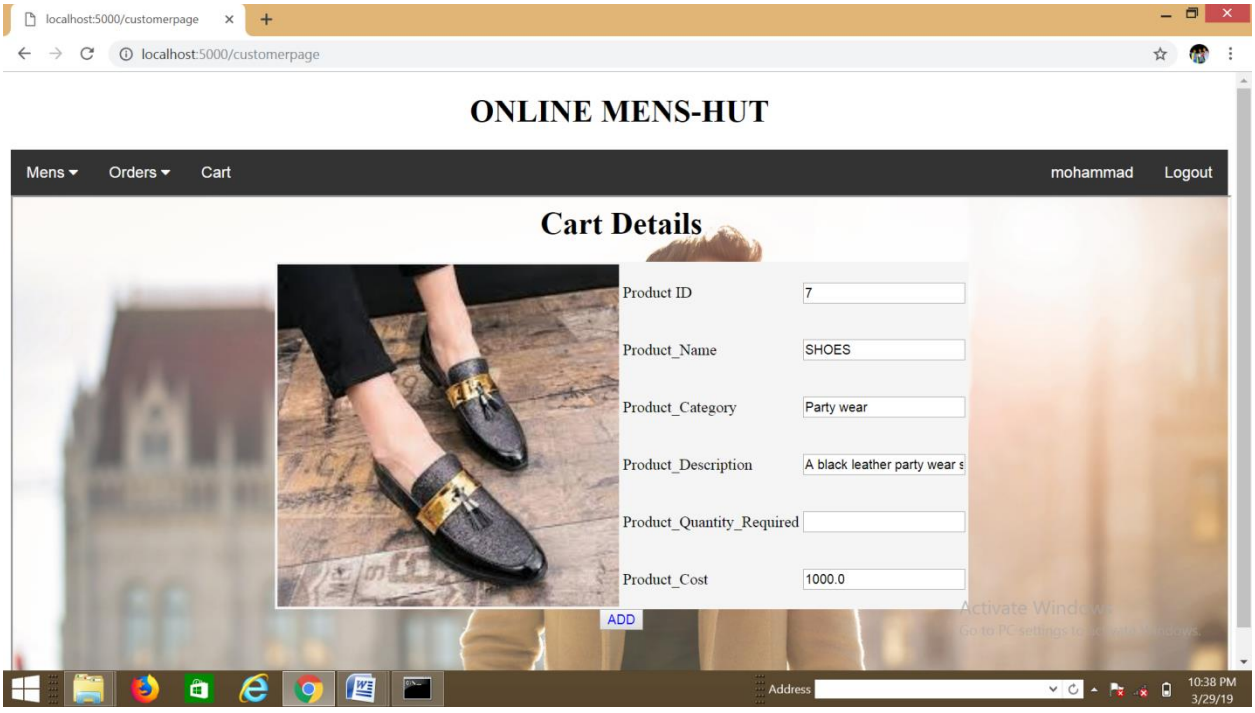


Figure18: 6.8: Cart Details Page

9. VIEW CART PAGE:

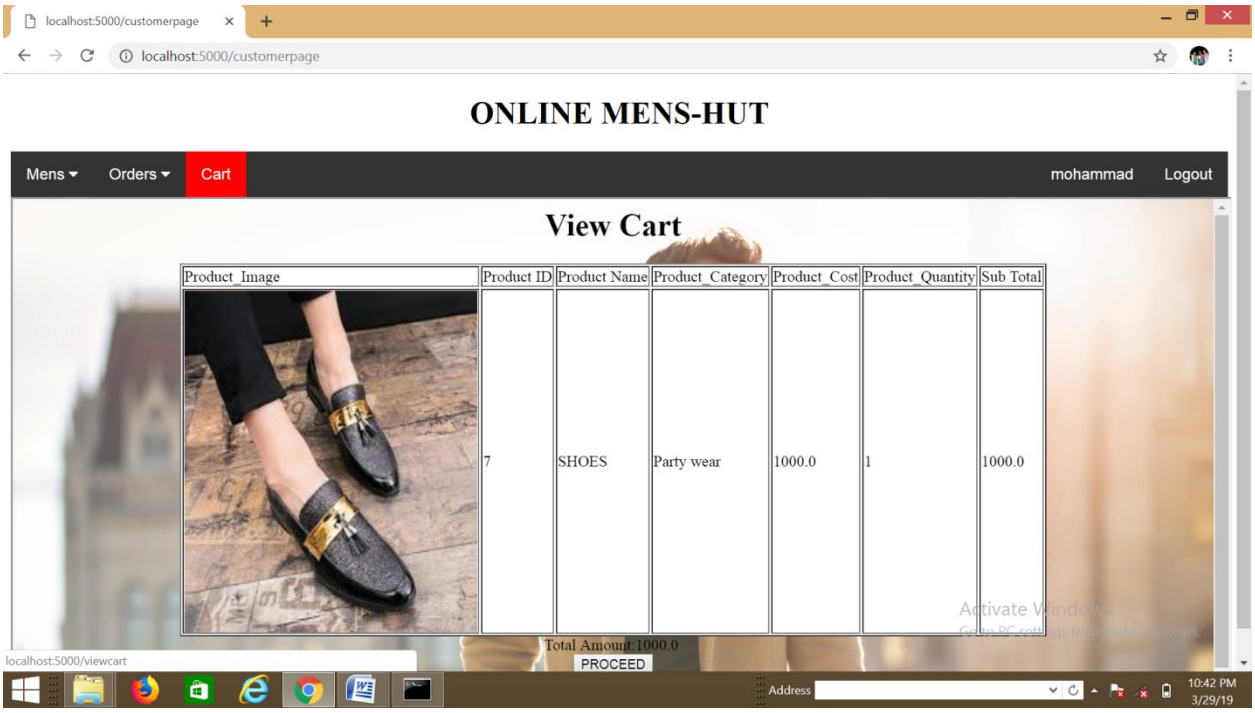


Figure19: 6.9: View Cart Page

10. DOOR DELIVERY DETAILS PAGE:

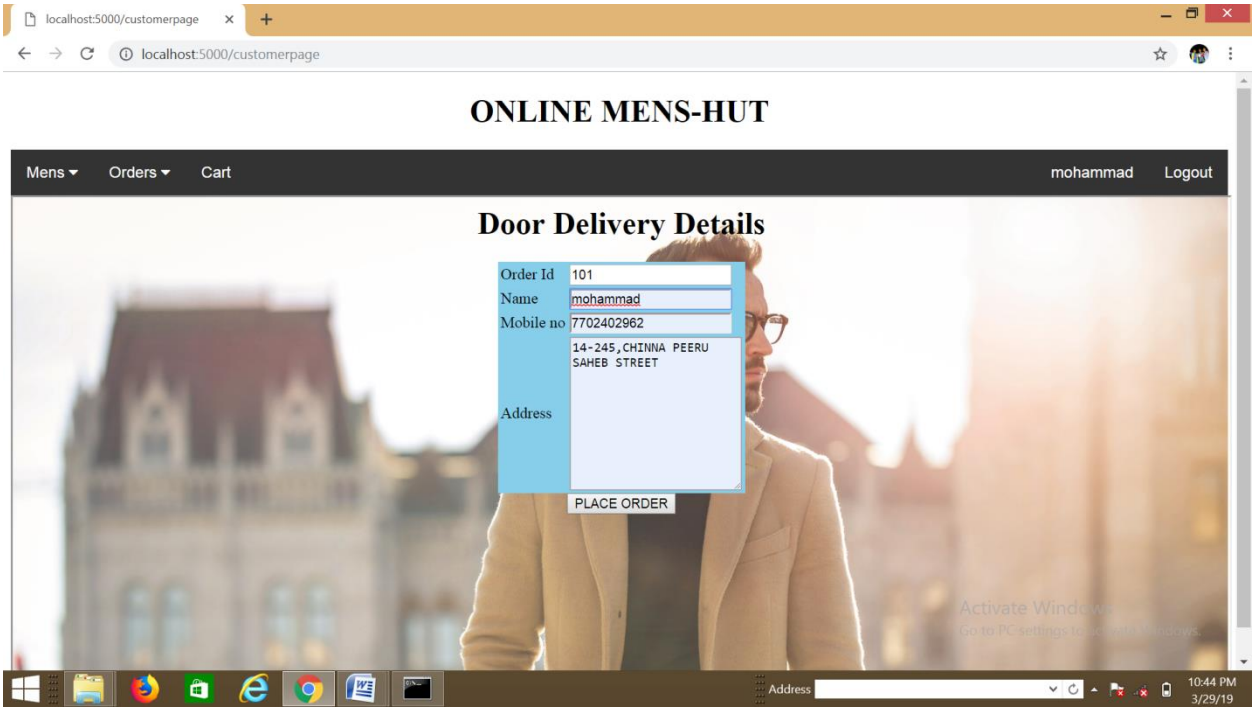


Figure20: 6.10: Door Delivery Details Page

11. PLACED ORDERS PAGE:

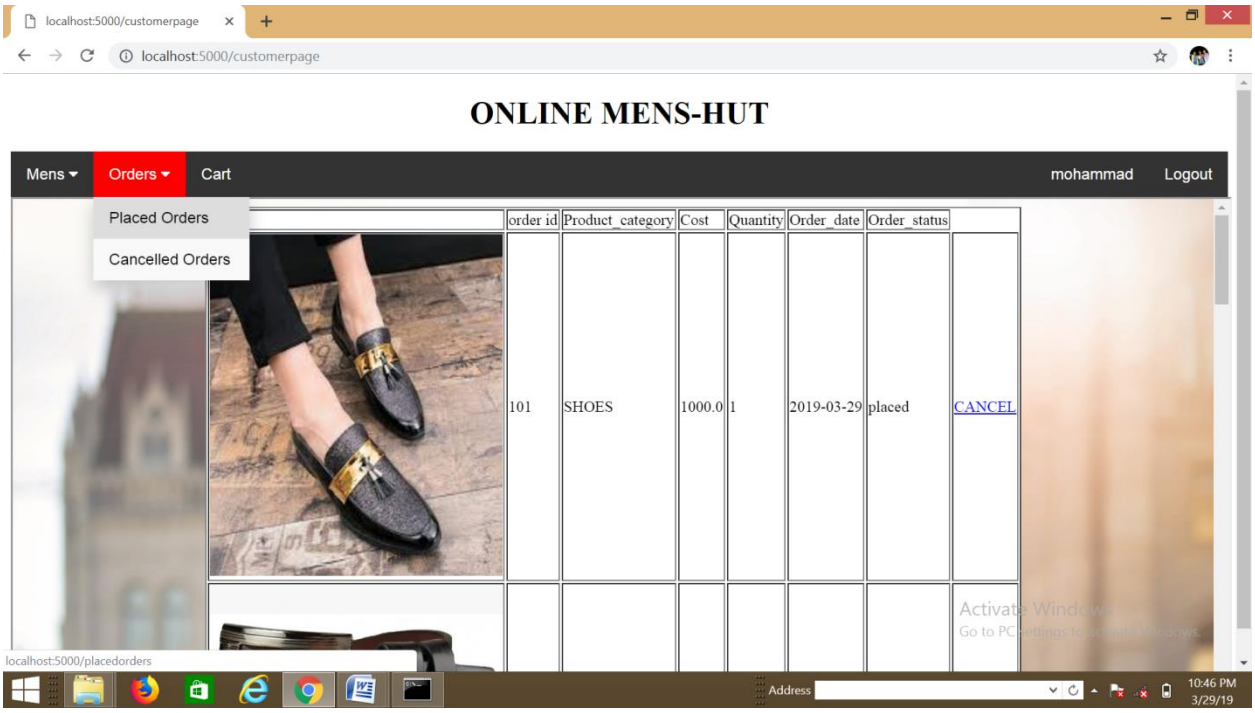


Figure21: 6.11: Placed Orders Page

12.CANCEL ORDERS PAGE:

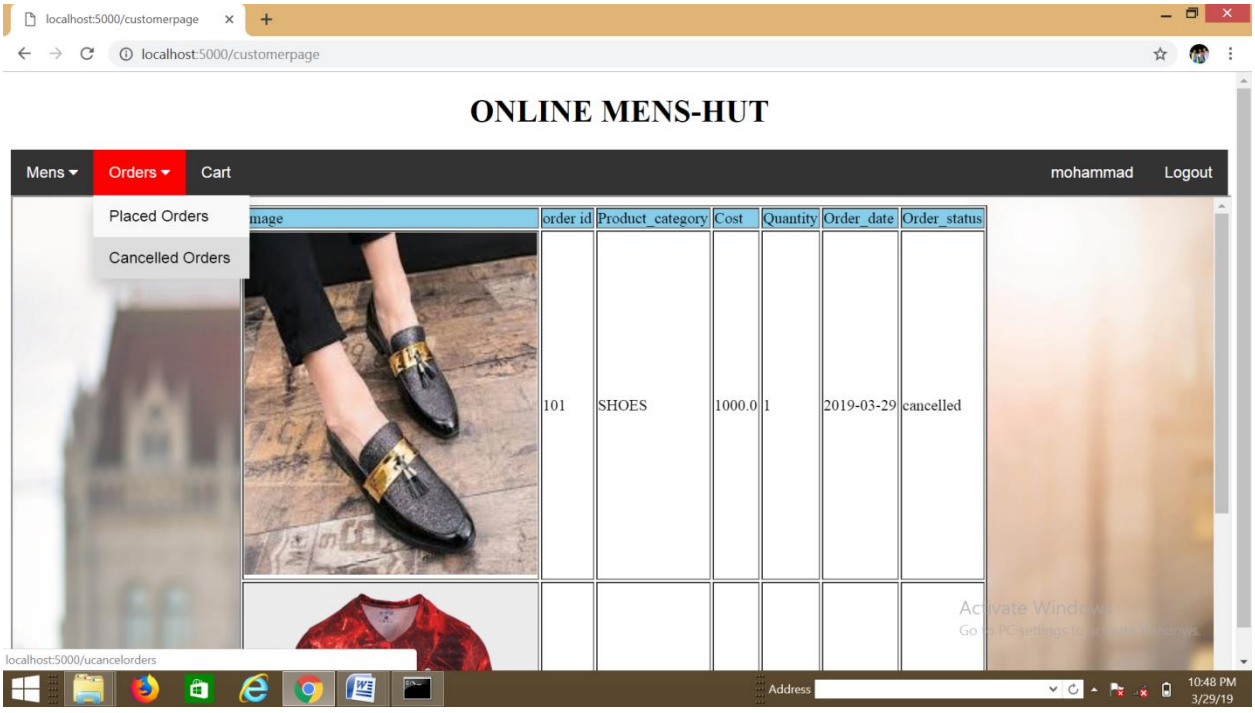


Figure22: 6.12: Cancel Orders Page

13. EDIT PROFILE PAGE:

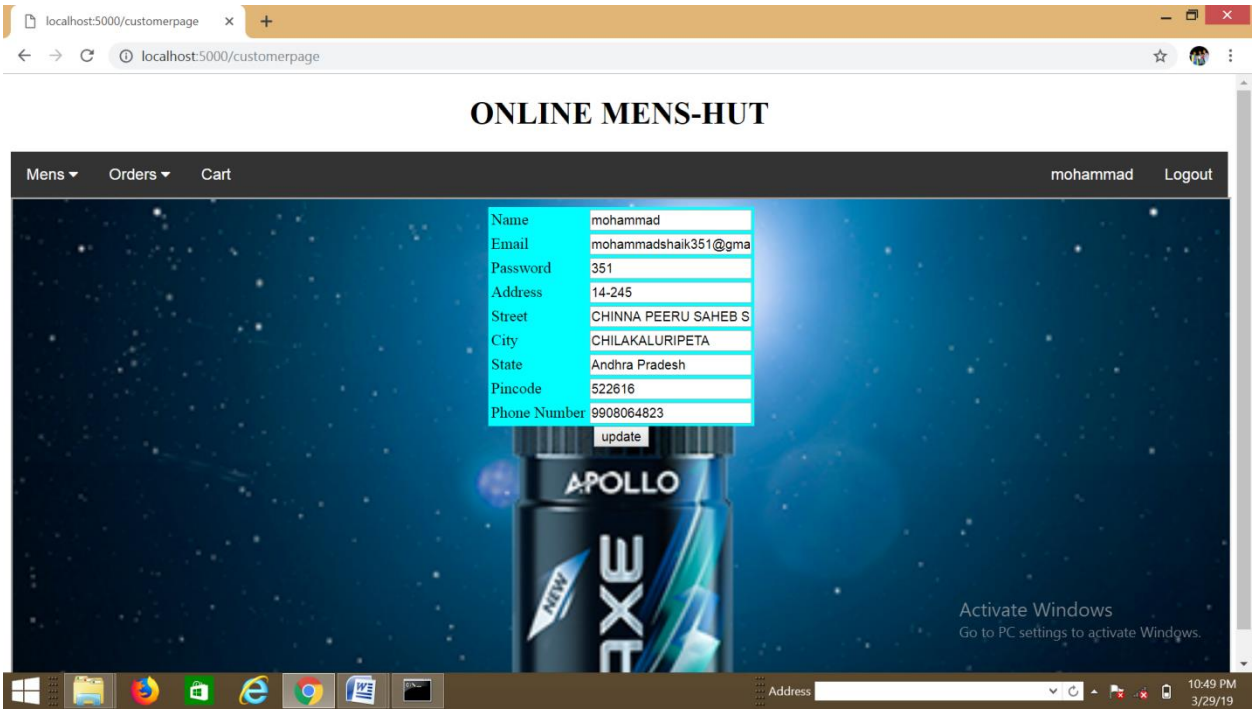


Figure23: 6.13: Edit Profile Page

14.ADMIN PAGE:

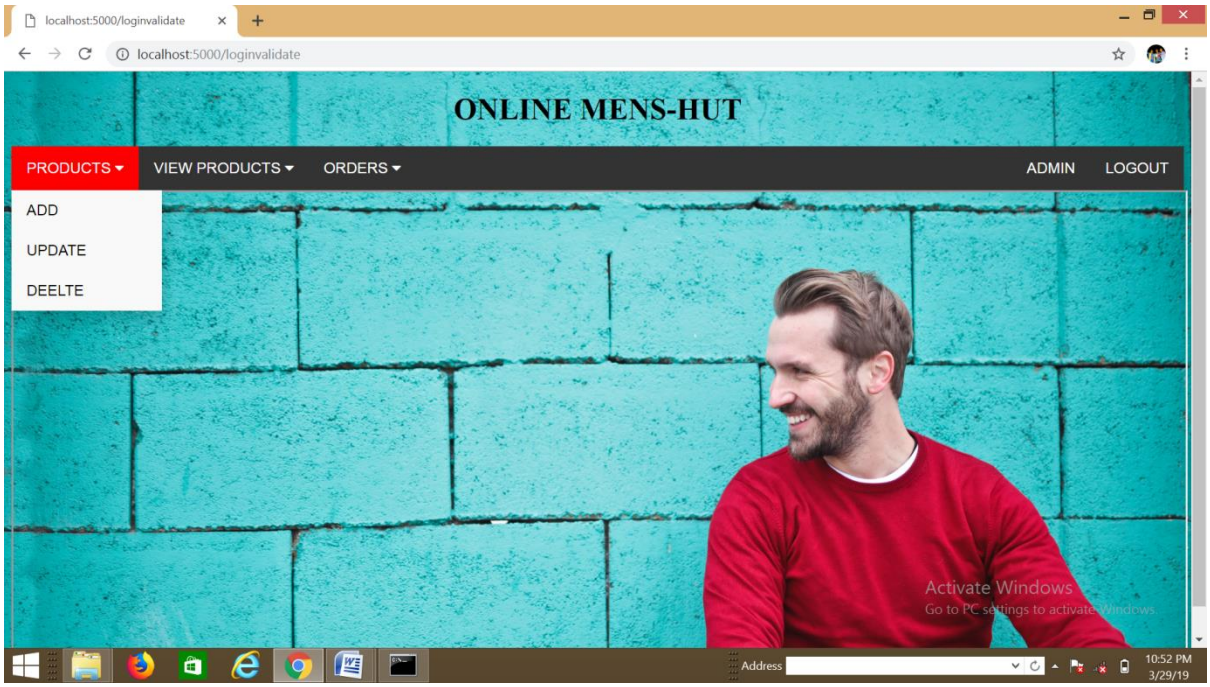


Figure24: 6.14: Admin Page

15. ADD ITEM PAGE:

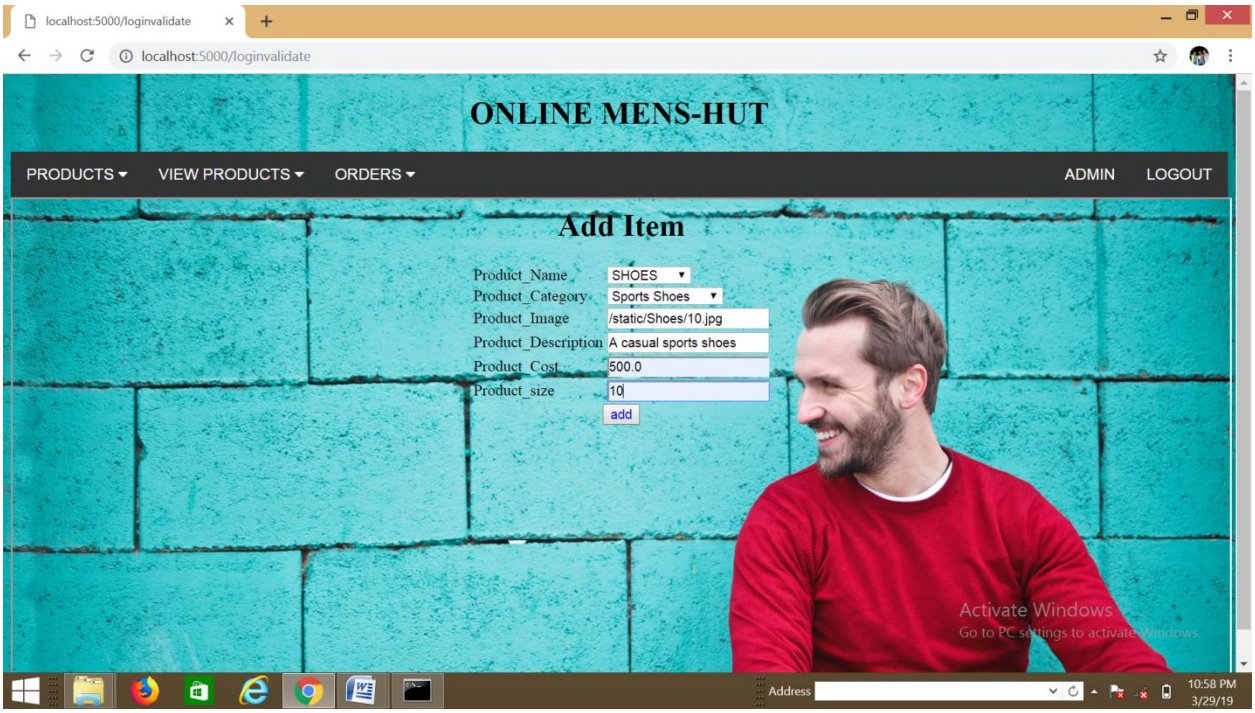
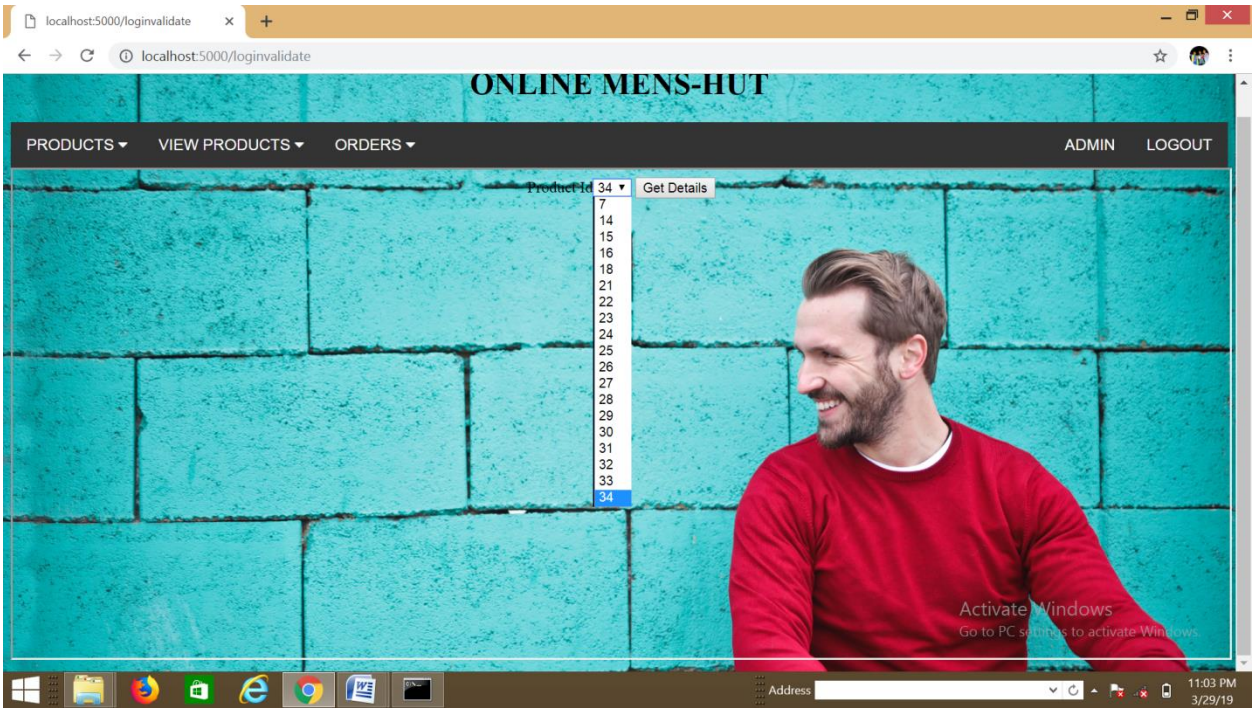


Figure25: 6.15: Add Item Page

16. ITEM UPDATE PAGE:



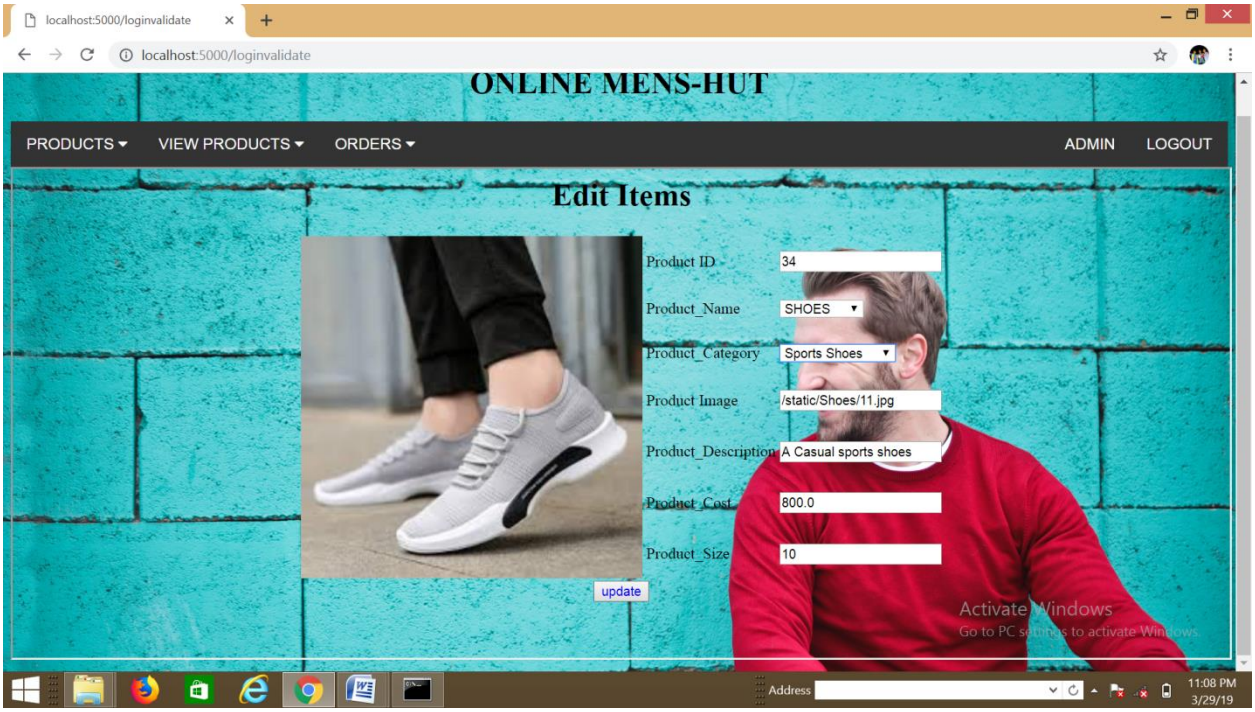
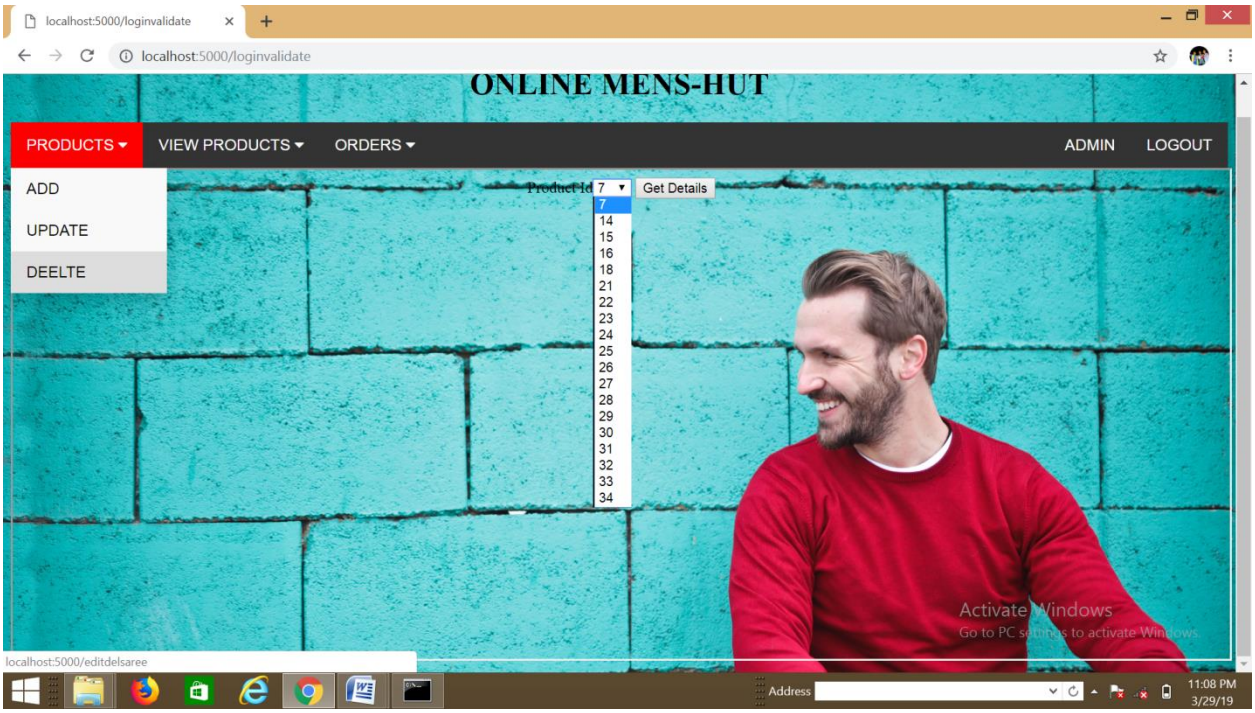


Figure26: 6.16: Item Update Page

17. ITEM DELETE PAGE:



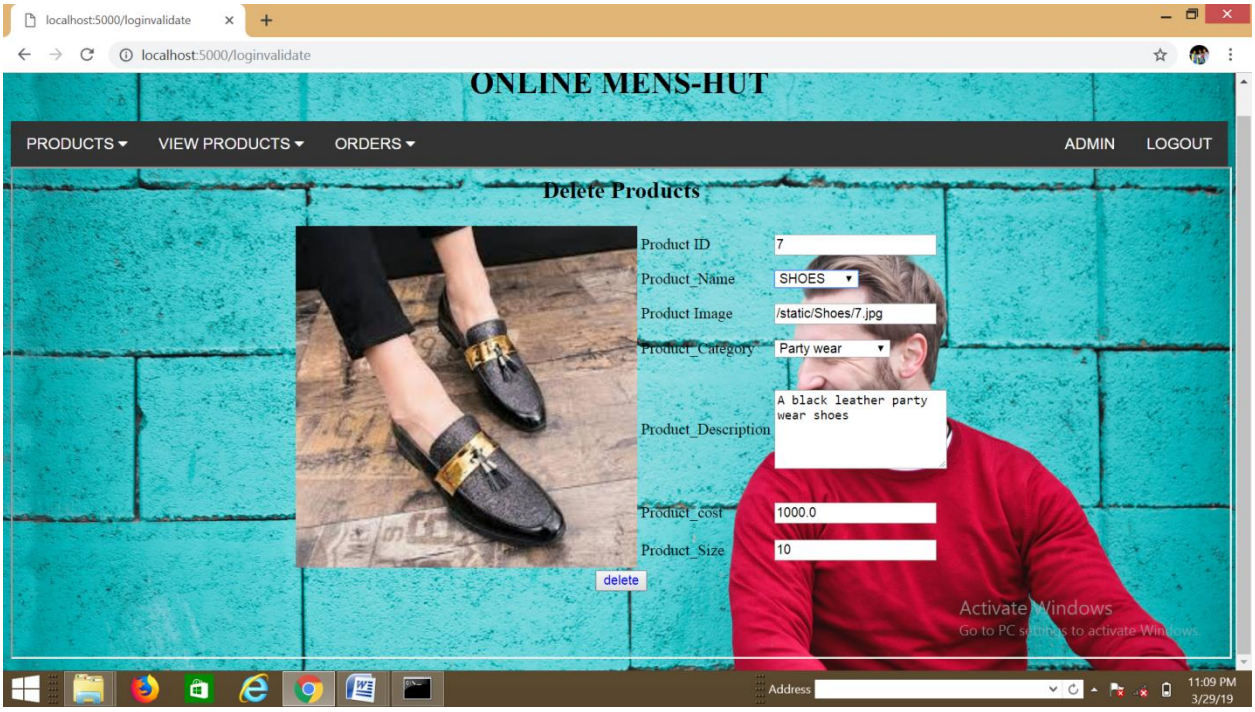
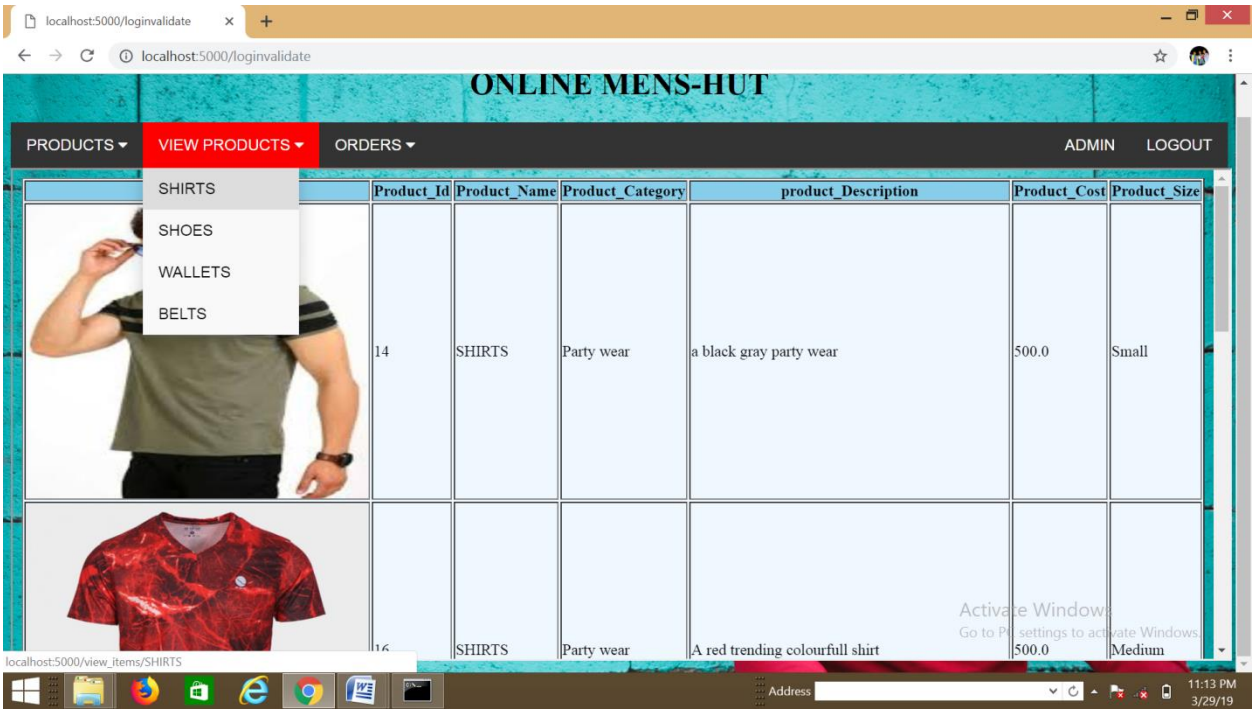


Figure27: 6.17: Item Delete Page

18.ADMIN_VIEW PRODUCTS PAGE:

18.1. SHIRTS PAGE:



18.2. SHOES PAGE:

18.3. WALLETS PAGE:

18.4. BELTS PAGE:

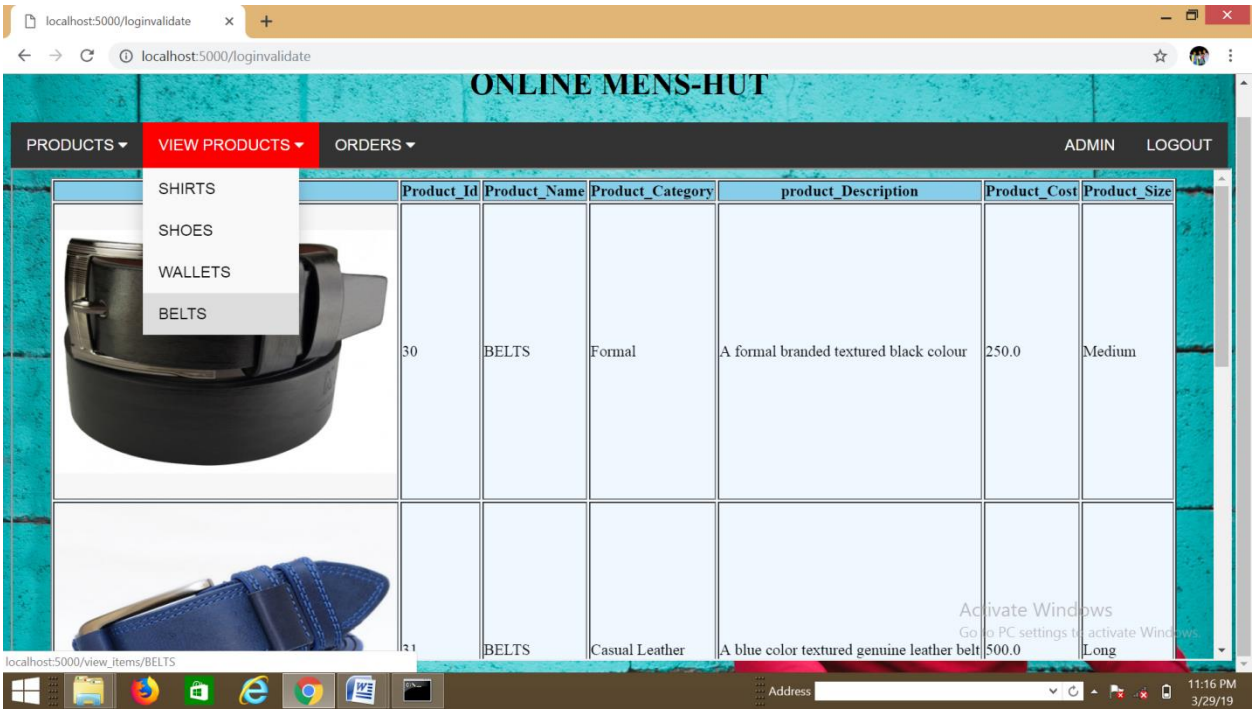


Figure28: 6.18: Admin_View Products Page

19. ADMIN PLACED_ORDERS PAGE:

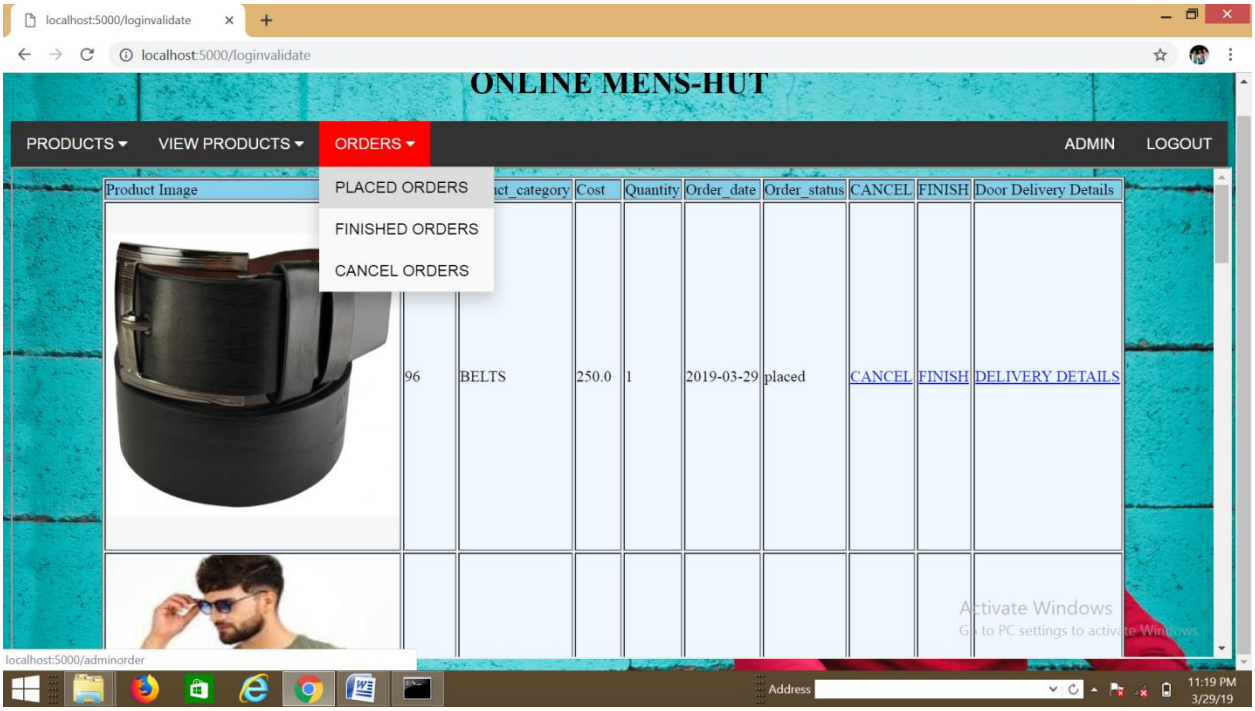


Figure29: 6.19: Admin Placed_Orders Page

20.ADMIN FINISHED_ORDERS PAGE:

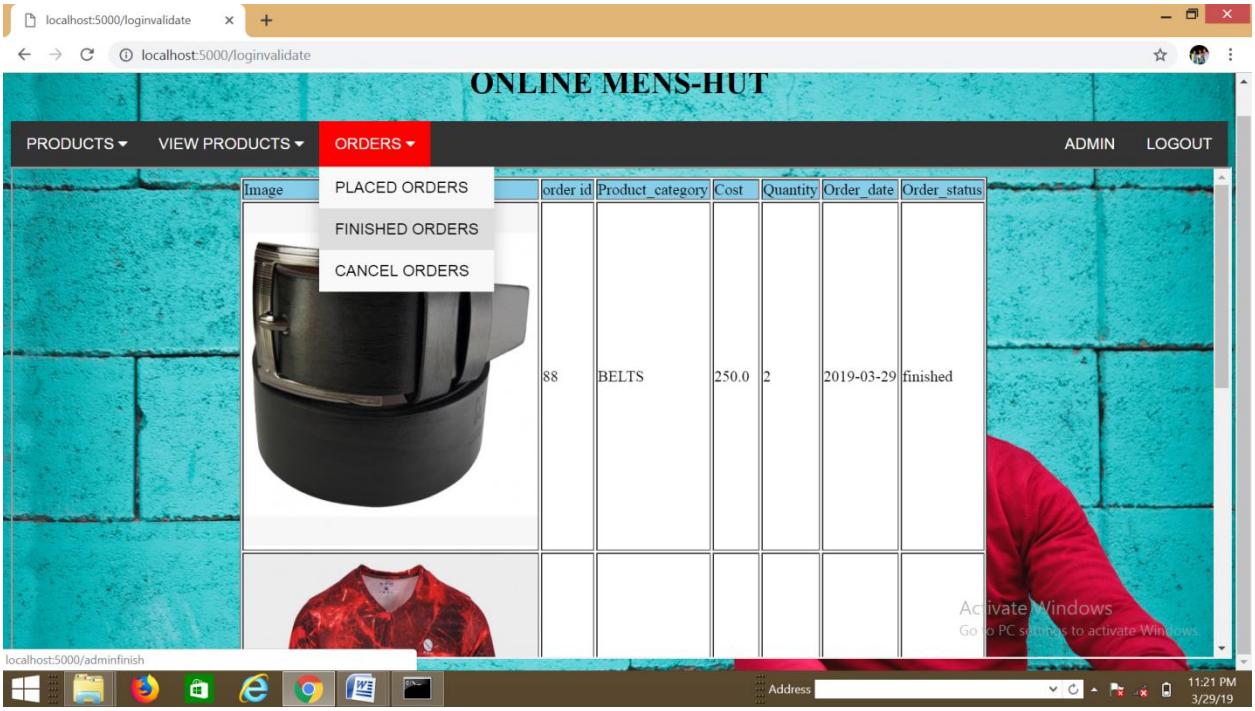


Figure30: 6.20: Admin Finished_Orders Page

21. ADMIN_CANCEL_ORDERS PAGE:

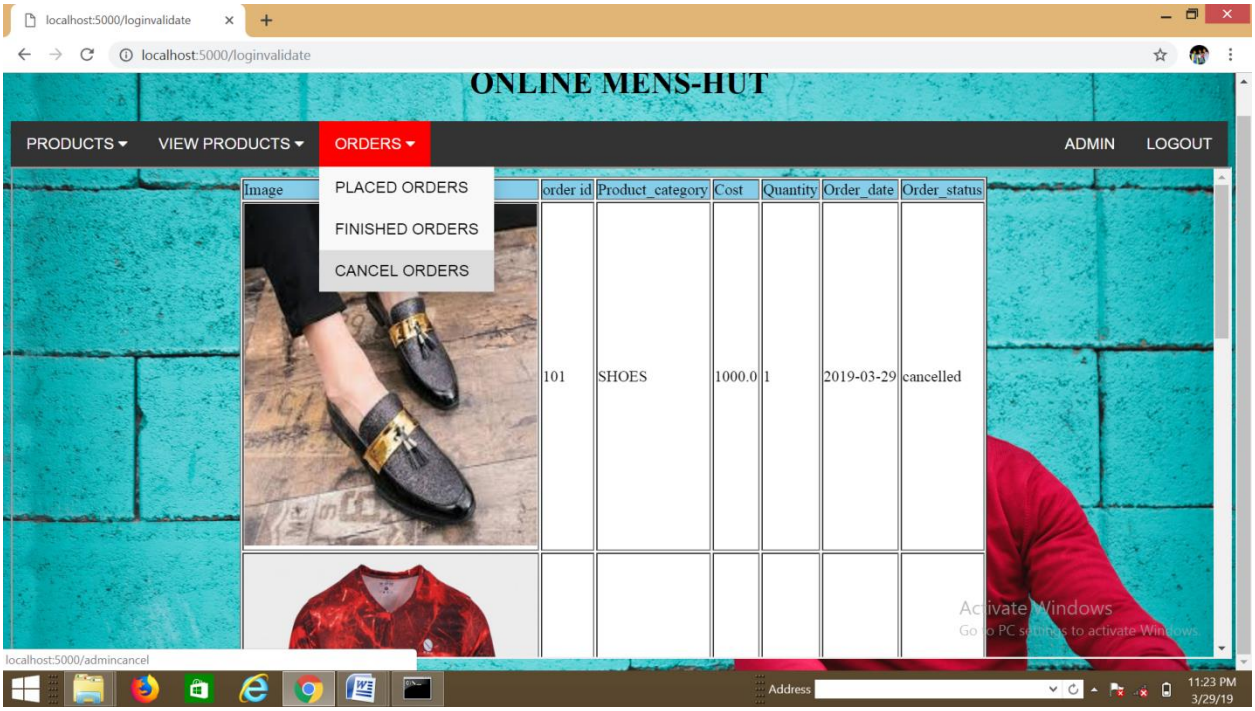


Figure31: 6.21: Admin Cancel_ Orders Page

22.DOOR DELIVERY DETAILS PAGE:

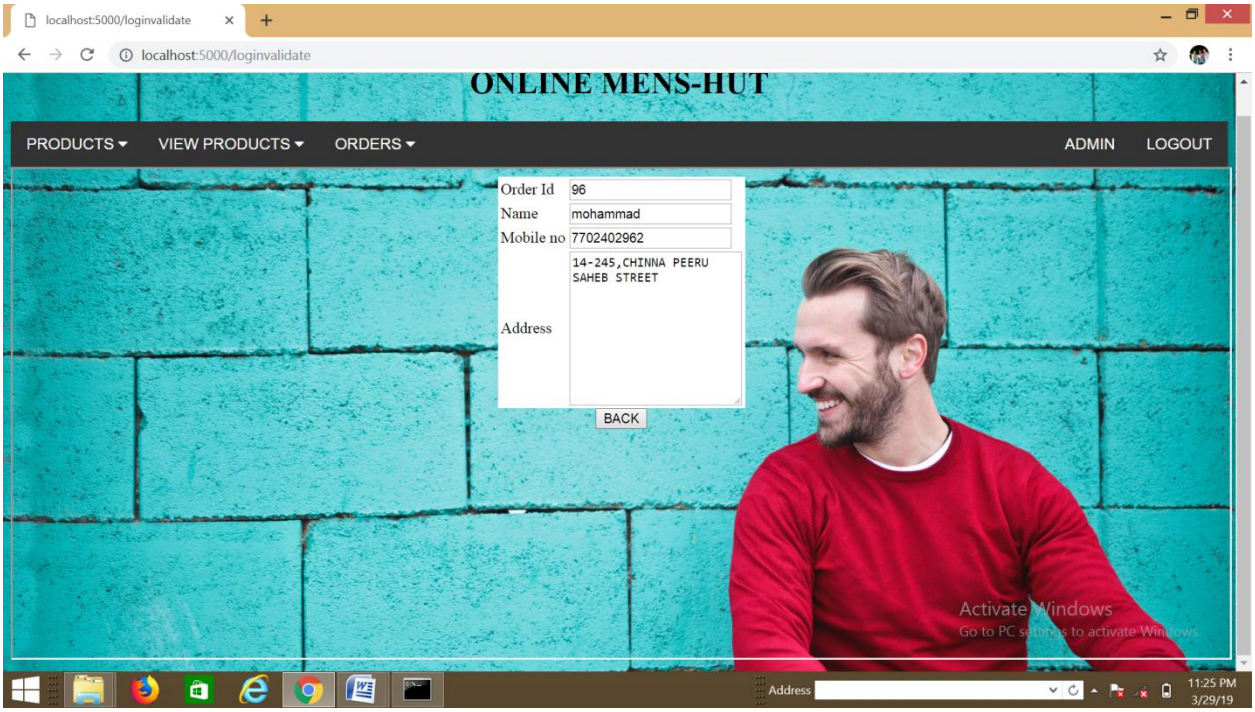


Figure32: 6.22: Door Delivery Details Page

7.CONCLUSION & FUTURE ENHANCEMENT

CONCLUSION:

- **Online Mens-Hut** will allow customers to place order without visiting the shop.
- Being able to buy any time, any place, anywhere.
- By using this system, user can view his previous orders also.
- Admin can easily maintain details regarding items.
- It saves the precious time of customer.

FUTURE ENHANCEMENT:

- Currently in this system, the admin manages only one seller and in future more sellers can be added.
- Whatever the technology implemented in the project is not existed forever and as the world is emerging day to day, the system should also suits for the new era.
- This system is only for men and in future it can be extended irrespective of age.

8.BIBLIOGRAPHY

ONLINE REFERENCES:

- http://www.google.com/
- https://www.tutorialspoint.com/python/
- https://www.w3schools.com/bootstrap4/
- https://realpython.com/tutorials/flask/

REFERENCES:

- Flask Web Development: Developing Web Applications with Python
  By Miguel Grinberg
- Flask By Example
  By Gareth Dwyer

DATABASE BOOKS:

- Using SQLite3
  By Jay A. Kreibicb