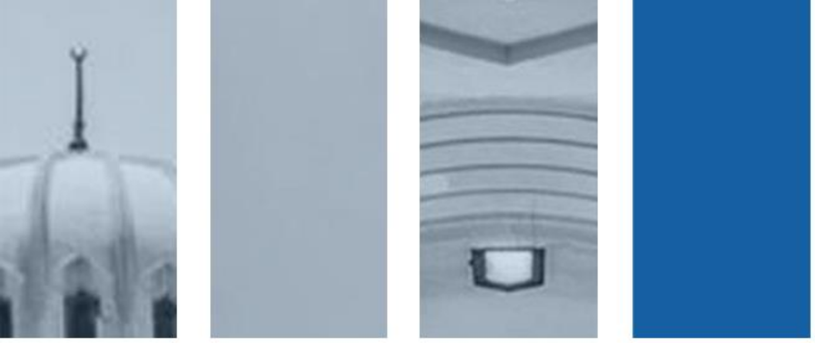


Distributed Project – phase 1

team members name	***IDs***
Loay Abdalla Youssef Abdelhady Elshall	*1701043*
Omar Mohamed Omar Abd El Samea	*1700903*
Mohamed Khaled Sayed Ibrahim Mohran	*1601166*
Amr Ahmed Mohamed Fathy	*1700918*
Omar Mohamed Mohamed Mostafa	*1700907*

Contents

1. Introduction:

In our E-commerce project, we will create a website that allows clients to use many services with powerful software design; we aimed to achieve some characteristics in our design like heterogeneity, transparency, openness, concurrency, security, and scalability

On our website, we provide many features to clients that user can create a new account and Create Login to your account Adds/Edits/removes Items to your account to be sold specifying the needed price, Deposits Cash into your account to purchase items, Search for items for sale by other users. Purchases item from another user transfers money from your account to his account and transfers the item from your account to his account. Views your account info such as current cash balance, a list of purchased items, a list of sold items, and items that have not been sold yet. Manages inventory of the items. Allows other stores to sell your products. Allows you to sell the products of other stores. Generates different kinds of reports such as reports about the transactions performed on the systems.

We divided the implementation into three parts, back end, front end and database

2. Target Beneficiaries of the Project:

A different set of beneficiaries. The fulfillment warehouse and their employees benefit,. The small package carriers and their employees. Factories get to sell more to distributors. Distributors get to sell more to online sellers.

Online sellers get to present their products to a much larger audience than if they only had a brick and mortar store.

Online customers get much more convenience, access to a larger selection of products, and more competitive prices and Lower prices. Convenient and safe shipping, customers can choose from Wide product variety, consumers can easily compare products, brands, and websites with even side-by-side comparison possible.

3. Adopted Programming Language:

Front end: Vue (HTML, CSS, Java script), Bulma (SCSS, CSS)

Back end: Django (python)

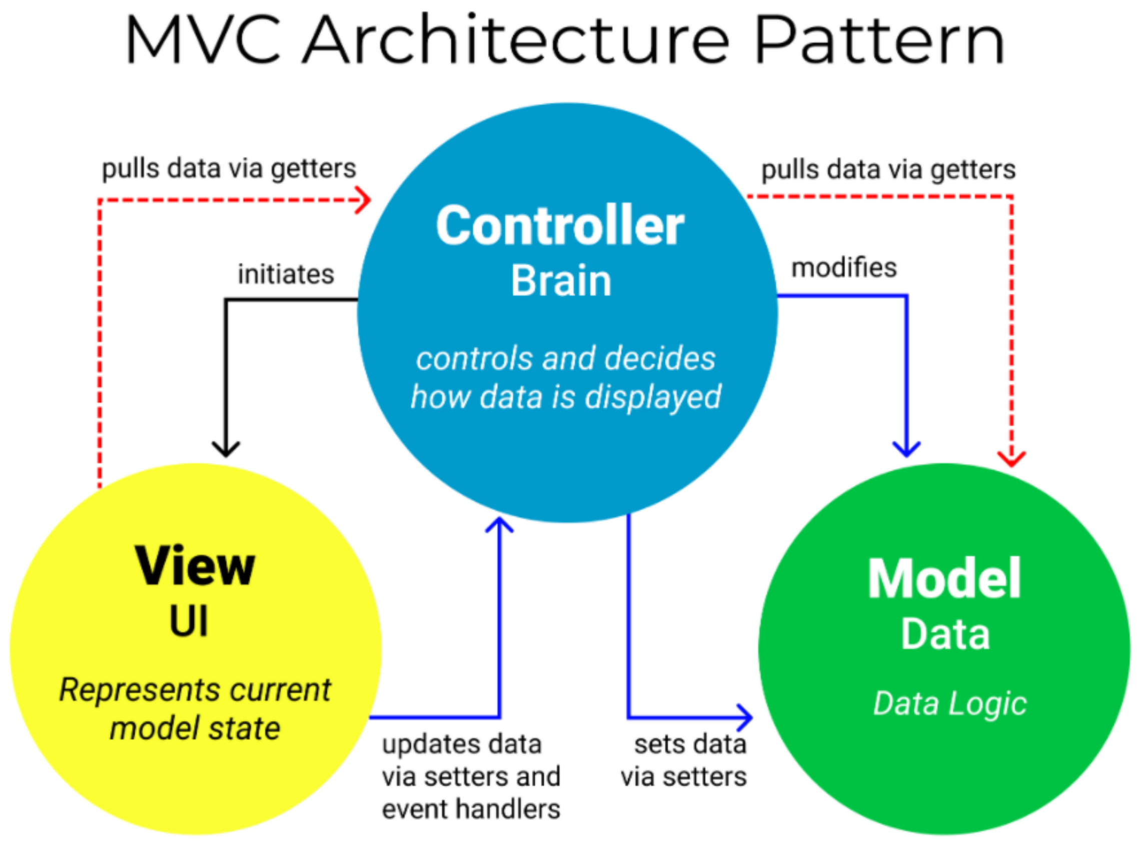
Database: MongoDB, NoSQL

4. System Architecture:

We used The MVC architecture pattern in our project, The MVC architecture pattern turns complex application development into a much more manageable process. It allows several developers to simultaneously work on the application.

The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

MVC is more of an architectural pattern, but not for complete application. MVC mostly relates to the UI / interaction layer of an application. You are still going to need business logic layer, maybe some service layer and data access layer.



MVC stands for model-view-controller. Here is what each of those components mean:

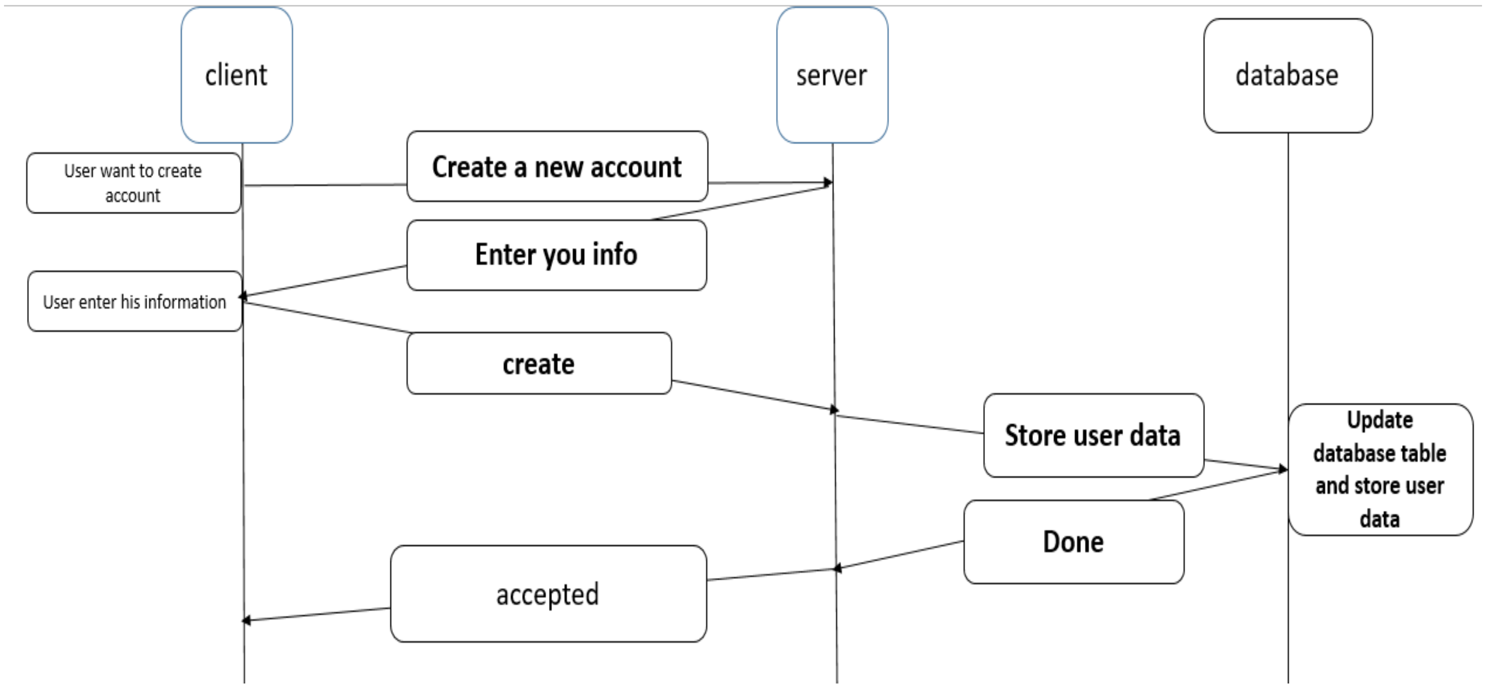
The Model contains only the pure application data; it contains no logic describing how to present the data to a user.

The View presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

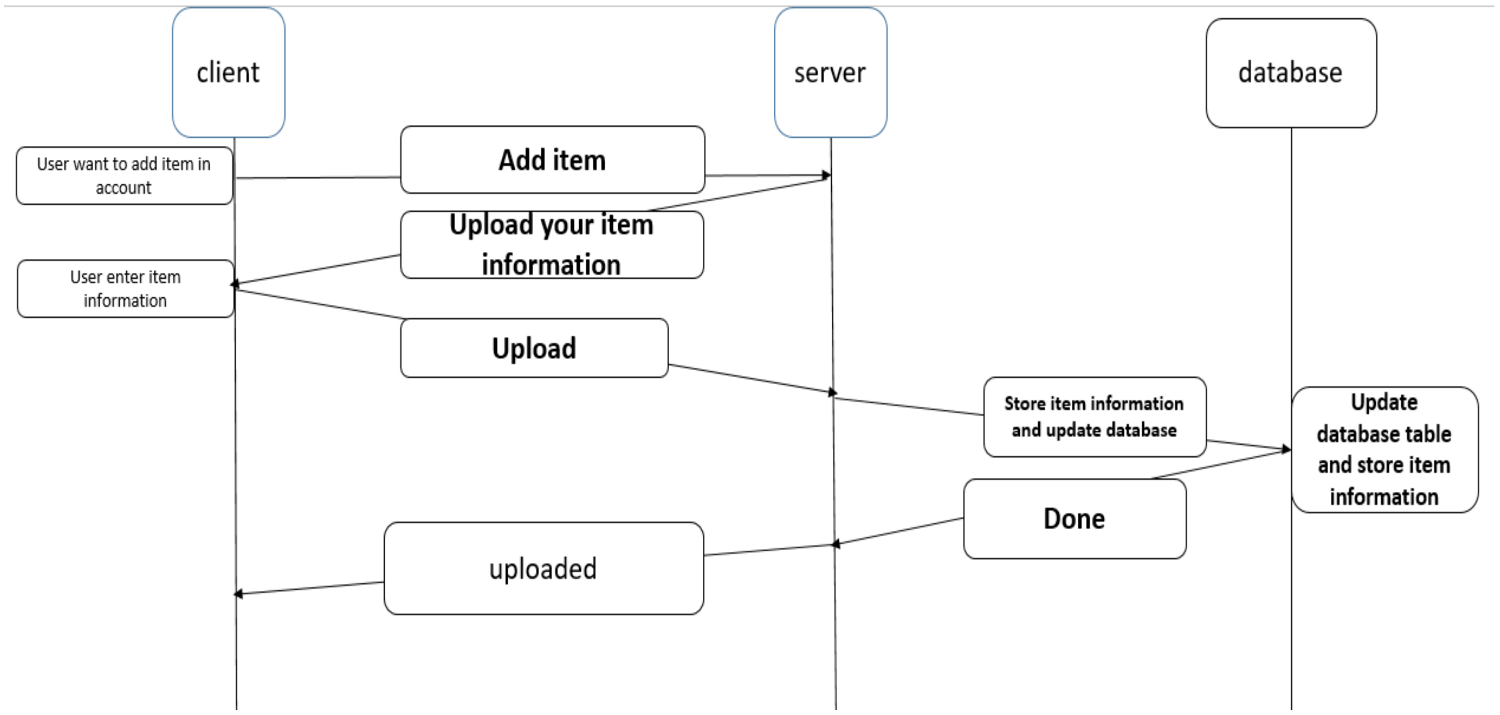
The Controller exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view**.**

5. Application Level Protocol:

First scenarios: the user create a new account:


client
server
database
User want to create account
Create a new account
Enter you info
User enter his information
create
Store user data
Update database table and store user data
Done
accepted

Second scenarios: the user add an item to his account:


client
server
database
User want to add item in account
Add item
Upload your item information
User enter item information
Upload
Store item information and update database
Update database table and store item information
Done
uploaded

6. Distributed Database Design:

Sharding:

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

There are two methods for addressing system growth: vertical and horizontal scaling.

Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space. Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload. Additionally, Cloud-based providers have hard ceilings based on available hardware configurations. As a result, there is a practical maximum for vertical scaling.

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade off is increased complexity in infrastructure and maintenance for the deployment.

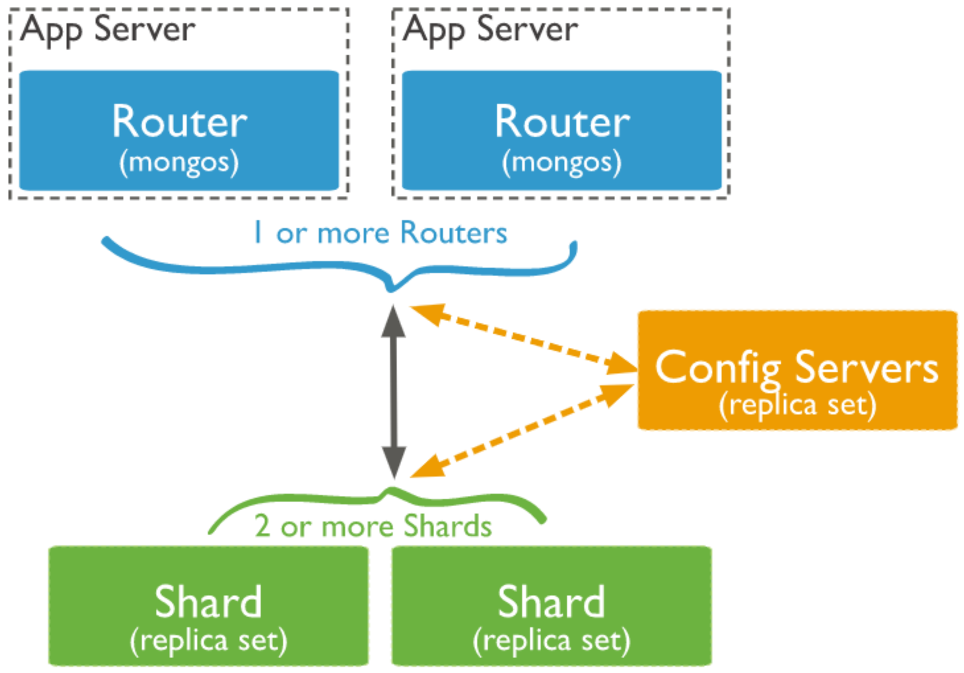
MongoDB supports horizontal scaling through sharding.

Sharded Cluster:

A MongoDB sharded cluster consists of the following components:

- **shard:** Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- **mongos:** The mongos acts as a query router, providing an interface between client applications and the sharded cluster. Starting in MongoDB 4.4, mongos can support hedged reads to minimize latencies.
- **Config servers:** Config servers store metadata and configuration settings for the cluster.

The following graphic describes the interaction of components within a sharded cluster:



MongoDB shards data at the collection level, distributing the collection data across the shards in the cluster.

Shard Keys:

MongoDB uses the shard key to distribute the collection's documents across shards. The shard key consists of a field or multiple fields in the documents.

- Starting in version 4.4, documents in sharded collections can be missing the shard key fields. Missing shard key fields are treated as having null values when distributing the documents across shards but not when routing queries. For more information, see Missing Shard Key Fields.
- In version 4.2 and earlier, shard key fields must exist in every document for a sharded collection.

You select the shard key when sharding a collection.

- Starting in MongoDB 5.0, you can reshard a collection by changing a collection's shard key.
- Starting in MongoDB 4.4, you can refine a shard key by adding a suffix field or fields to the existing shard key.
- In MongoDB 4.2 and earlier, the choice of shard key cannot be changed after sharding.

A document's shard key value determines its distribution across the shards.

- Starting in MongoDB 4.2, you can update a document's shard key value unless your shard key field is the immutable _id field. See Change a Document's Shard Key Value for more information.
- In MongoDB 4.0 and earlier, a document's shard key field value is immutable.

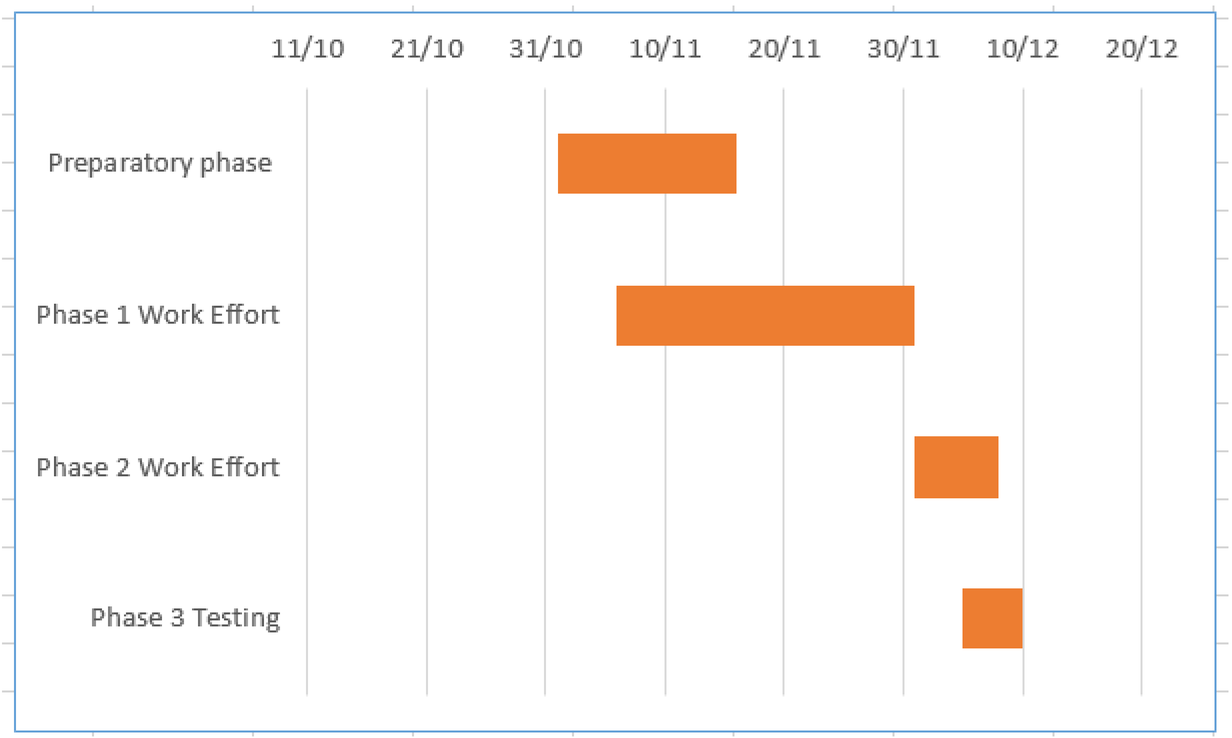
Shard Key Index:

To shard a populated collection, the collection must have an index that starts with the shard key. When sharding an empty collection, MongoDB creates the supporting index if the collection does not already have an appropriate index for the specified shard key. See Shard Key Indexes.

Shard Key Strategy:

The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster. A cluster with the best possible hardware and infrastructure can be bottlenecked by the choice of shard key. The choice of shard key and its backing index can also affect the sharding strategy that your cluster can use.

7. Time Plan:



Preparatory phase:

Date: 1/11/2021 to 10/11/2021

Description: discusses the implementations methods and decided to use Front-end: Vue (HTML, CSS, Java script), Bulma (SCSS, CSS), Back-end: Django (python) and Database: MongoDB, NoSQL.

Then studied this methods by watching online tutorials before starting implementation phase

Phase 1 Work Effort:

Date: 6/11/2021 to 30/11/2021

Description: start implementing database and back-end

Phase 2 Work Effort:

Date: 1/12/2021 to 7/12/2021

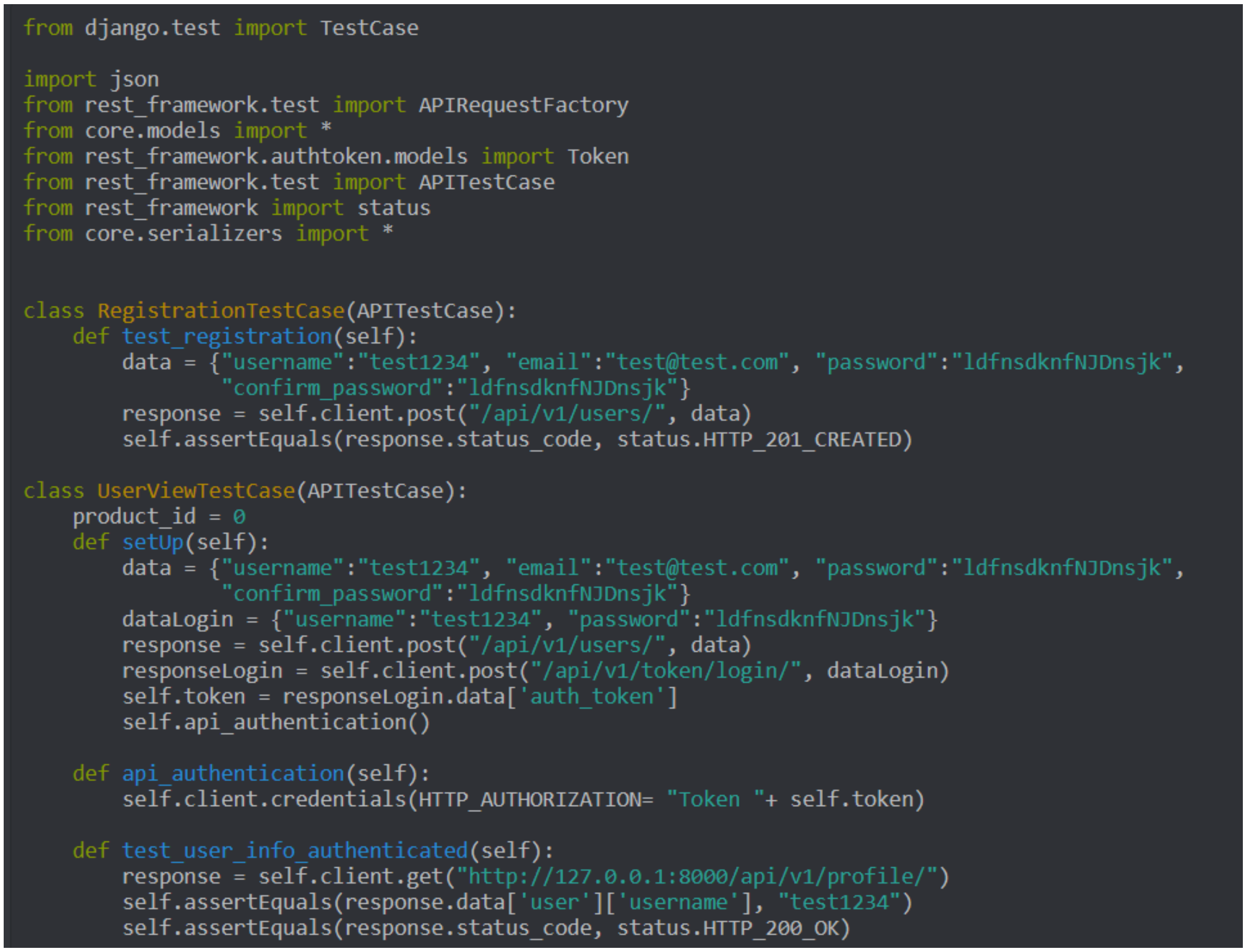
Description: start implementing front-end

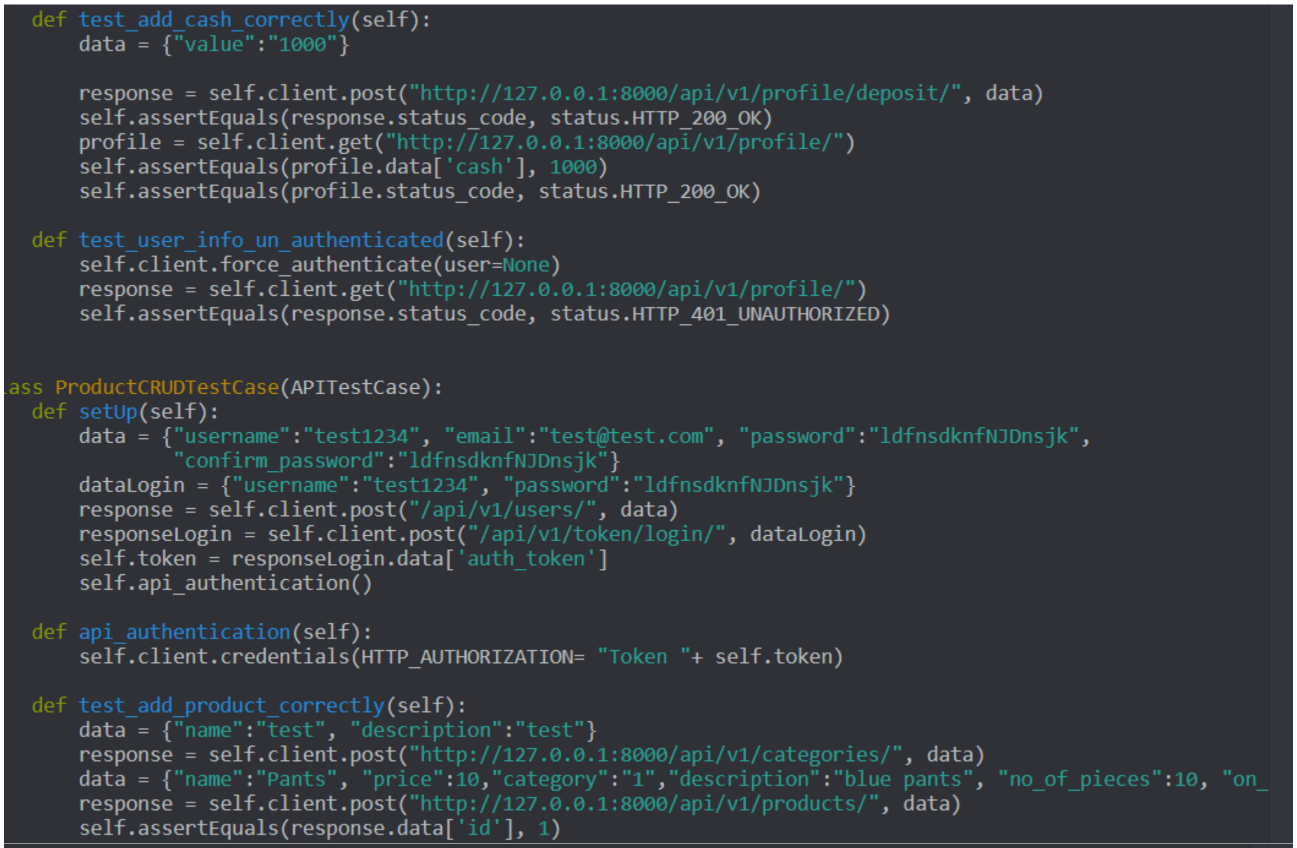
Phase 3 testing and reporting:

Date: 7/12/2021 to 10/12/2021

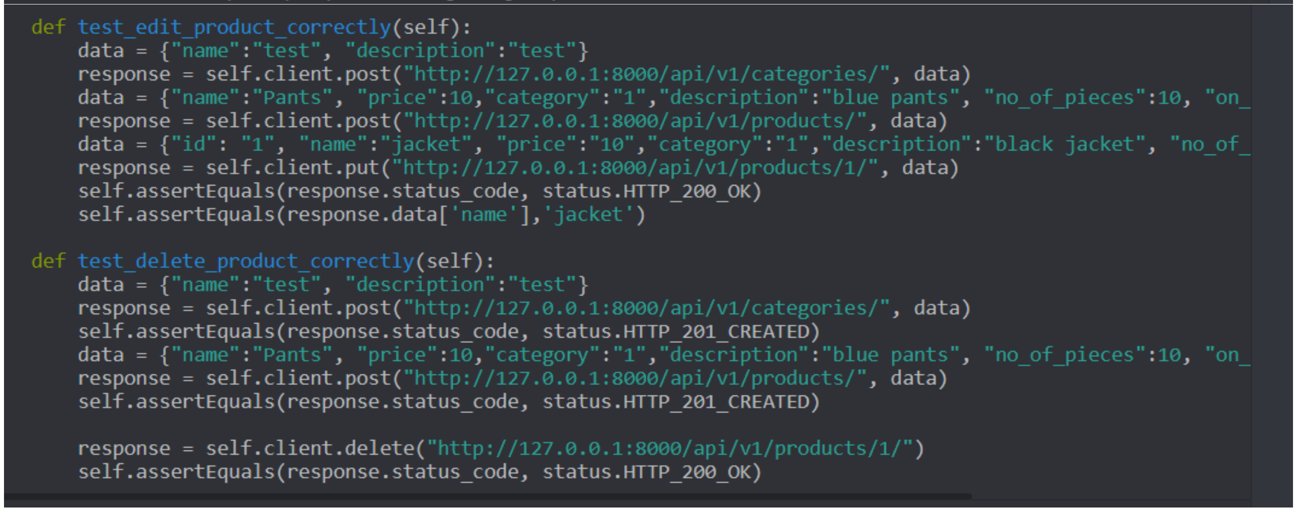
Description: start integrating, testing and preparing the report

8. Testing:


```
from django.test import TestCase

import json
from rest_framework.test import APIRequestFactory
from core.models import *
from rest_framework.authtoken.models import Token
from rest_framework.test import APITestCase
from rest_framework import status
from core.serializers import *


class RegistrationTestCase(APITestCase):
    def test_registration(self):
        data = {"username":"test1234", "email":"test@test.com", "password":"ldfnsdknfNJDnsjk",
                "confirm_password":"ldfnsdknfNJDnsjk"}
        response = self.client.post("/api/v1/users/", data)
        self.assertEquals(response.status_code, status.HTTP_201_CREATED)

class UserViewTestCase(APITestCase):
    product_id = 0
    def setUp(self):
        data = {"username":"test1234", "email":"test@test.com", "password":"ldfnsdknfNJDnsjk",
                "confirm_password":"ldfnsdknfNJDnsjk"}
        dataLogin = {"username":"test1234", "password":"ldfnsdknfNJDnsjk"}
        response = self.client.post("/api/v1/users/", data)
        responseLogin = self.client.post("/api/v1/token/login/", dataLogin)
        self.token = responseLogin.data['auth_token']
        self.api_authentication()

    def api_authentication(self):
        self.client.credentials(HTTP_AUTHORIZATION= "Token "+ self.token)

    def test_user_info_authenticated(self):
        response = self.client.get("http://127.0.0.1:8000/api/v1/profile/")
        self.assertEquals(response.data['user']['username'], "test1234")
        self.assertEquals(response.status_code, status.HTTP_200_OK)
```



```
    def test_add_cash_correctly(self):
        data = {"value":"1000"}

        response = self.client.post("http://127.0.0.1:8000/api/v1/profile/deposit/", data)
        self.assertEquals(response.status_code, status.HTTP_200_OK)
        profile = self.client.get("http://127.0.0.1:8000/api/v1/profile/")
        self.assertEquals(profile.data['cash'], 1000)
        self.assertEquals(profile.status_code, status.HTTP_200_OK)

    def test_user_info_un_authenticated(self):
        self.client.force_authenticate(user=None)
        response = self.client.get("http://127.0.0.1:8000/api/v1/profile/")
        self.assertEquals(response.status_code, status.HTTP_401_UNAUTHORIZED)


ass ProductCRUDTestCase(APITestCase):
    def setUp(self):
        data = {"username":"test1234", "email":"test@test.com", "password":"ldfnsdknfNJDnsjk",
                "confirm_password":"ldfnsdknfNJDnsjk"}
        dataLogin = {"username":"test1234", "password":"ldfnsdknfNJDnsjk"}
        response = self.client.post("/api/v1/users/", data)
        responseLogin = self.client.post("/api/v1/token/login/", dataLogin)
        self.token = responseLogin.data['auth_token']
        self.api_authentication()

    def api_authentication(self):
        self.client.credentials(HTTP_AUTHORIZATION= "Token "+ self.token)

    def test_add_product_correctly(self):
        data = {"name":"test", "description":"test"}
        response = self.client.post("http://127.0.0.1:8000/api/v1/categories/", data)
        data = {"name":"Pants", "price":10,"category":"1","description":"blue pants", "no_of_pieces":10, "on_
        response = self.client.post("http://127.0.0.1:8000/api/v1/products/", data)
        self.assertEquals(response.data['id'], 1)
```



```
    def test_edit_product_correctly(self):
        data = {"name":"test", "description":"test"}
        response = self.client.post("http://127.0.0.1:8000/api/v1/categories/", data)
        data = {"name":"Pants", "price":10,"category":"1","description":"blue pants", "no_of_pieces":10, "on_
        response = self.client.post("http://127.0.0.1:8000/api/v1/products/", data)
        data = {"id": "1", "name":"jacket", "price":"10","category":"1","description":"black jacket", "no_of_
        response = self.client.put("http://127.0.0.1:8000/api/v1/products/1/", data)
        self.assertEquals(response.status_code, status.HTTP_200_OK)
        self.assertEquals(response.data['name'],'jacket')

    def test_delete_product_correctly(self):
        data = {"name":"test", "description":"test"}
        response = self.client.post("http://127.0.0.1:8000/api/v1/categories/", data)
        self.assertEquals(response.status_code, status.HTTP_201_CREATED)
        data = {"name":"Pants", "price":10,"category":"1","description":"blue pants", "no_of_pieces":10, "on_
        response = self.client.post("http://127.0.0.1:8000/api/v1/products/", data)
        self.assertEquals(response.status_code, status.HTTP_201_CREATED)

        response = self.client.delete("http://127.0.0.1:8000/api/v1/products/1/")
        self.assertEquals(response.status_code, status.HTTP_200_OK)
```

9. End-User Guide:

• Search item process:

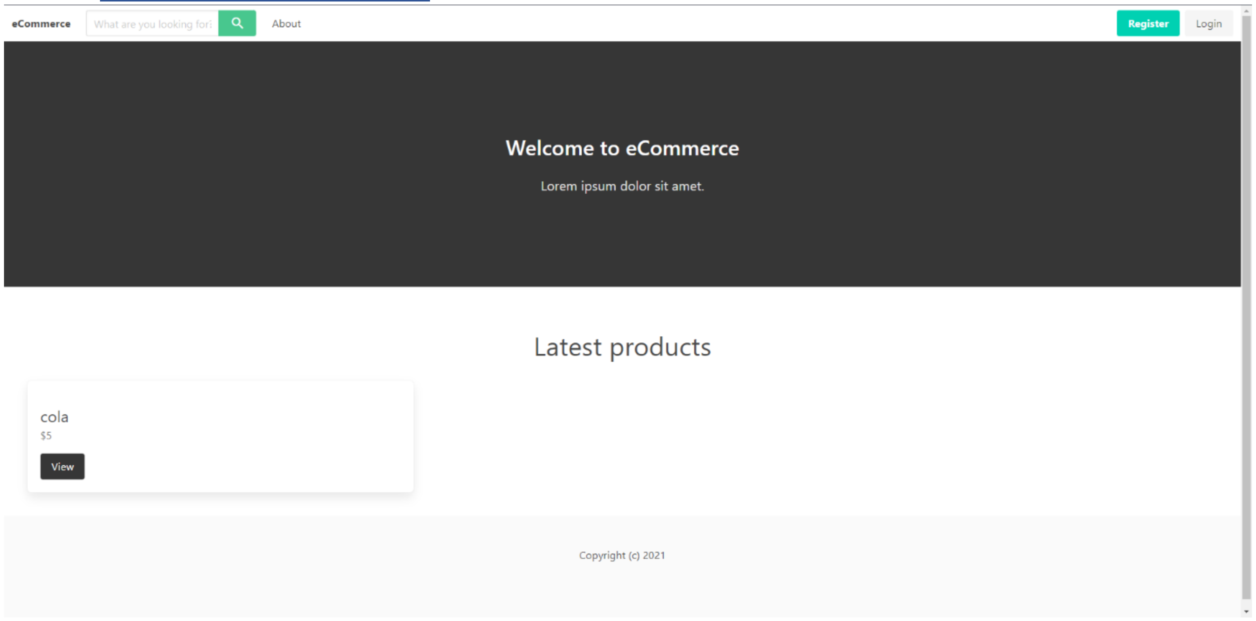


Figure 1

From figure 1 user can search for products by using search bar, login in his account by click on login button and make registration process by click on register button

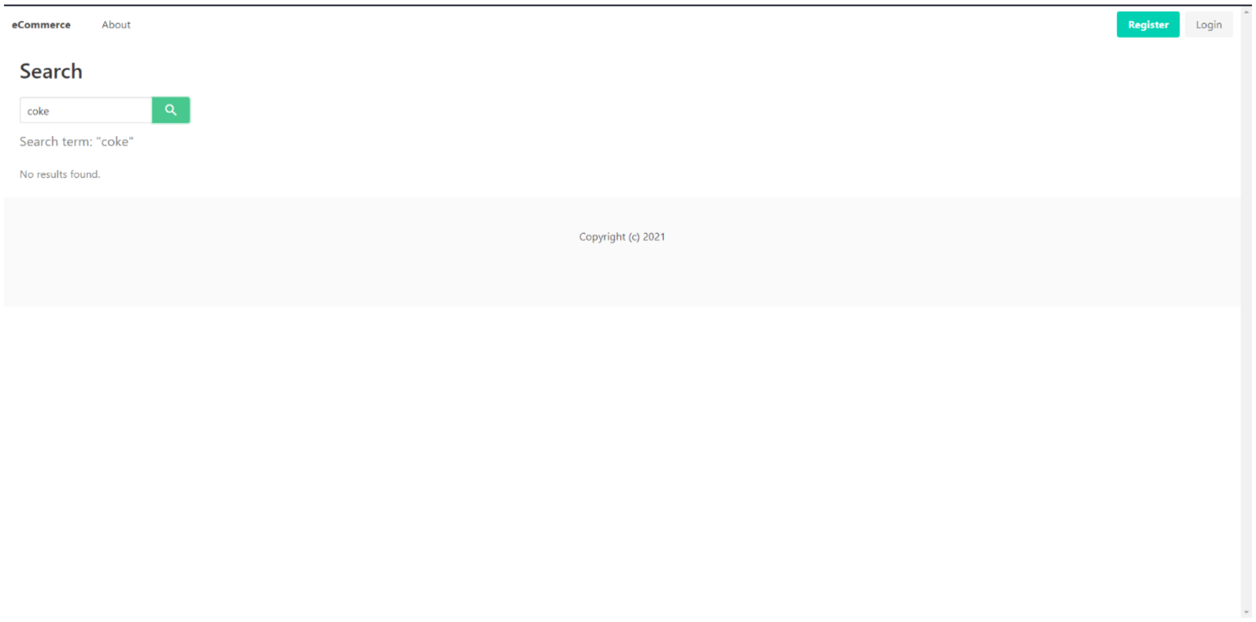


Figure 2

Figure 2 after searching for item but the item not found

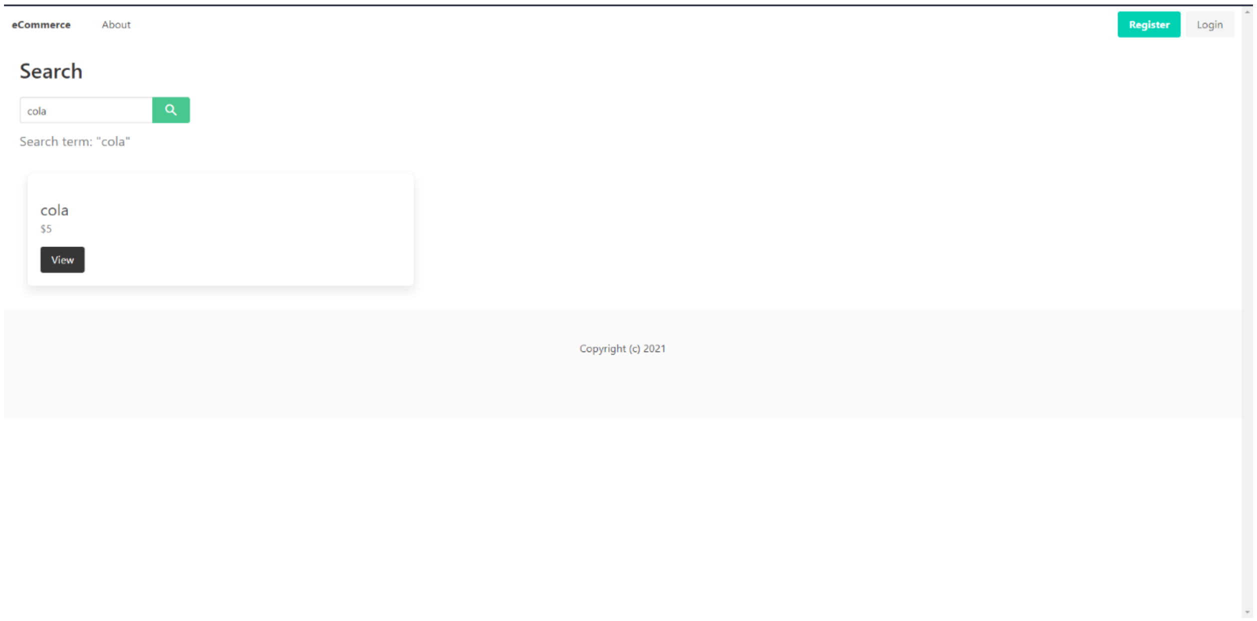


Figure 3 – search

Figure 3 after searching for item and the item is found

- **Login and registration process:**

From figure 1 user can login in his account by click on login button and make registration process by click on register button

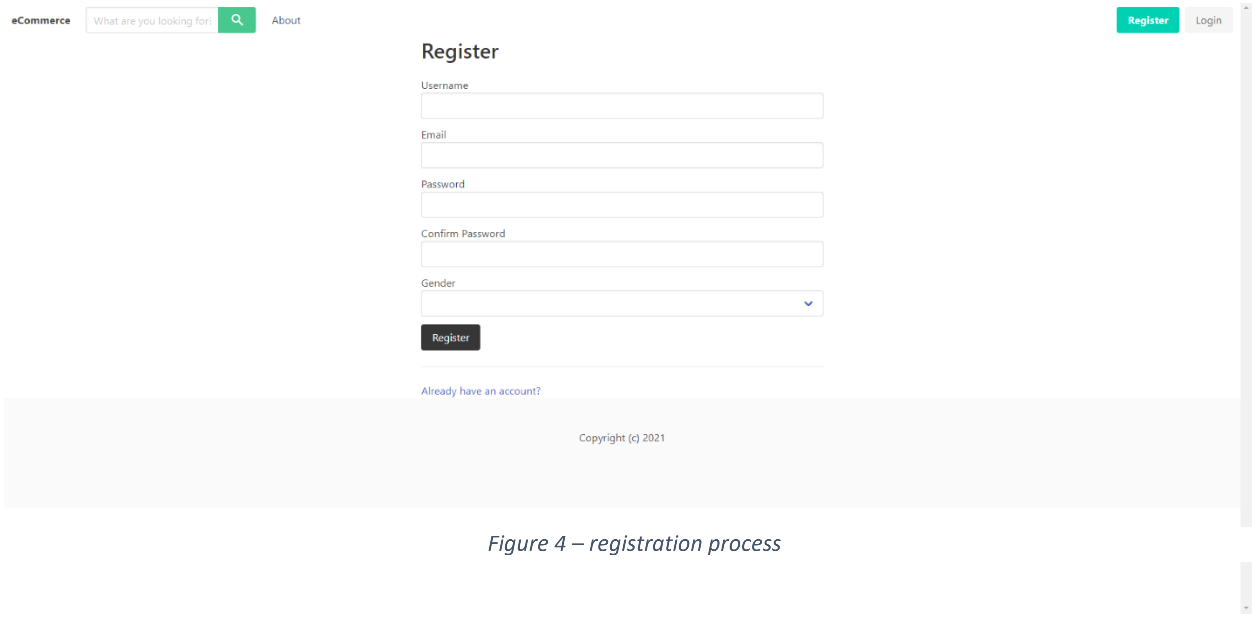
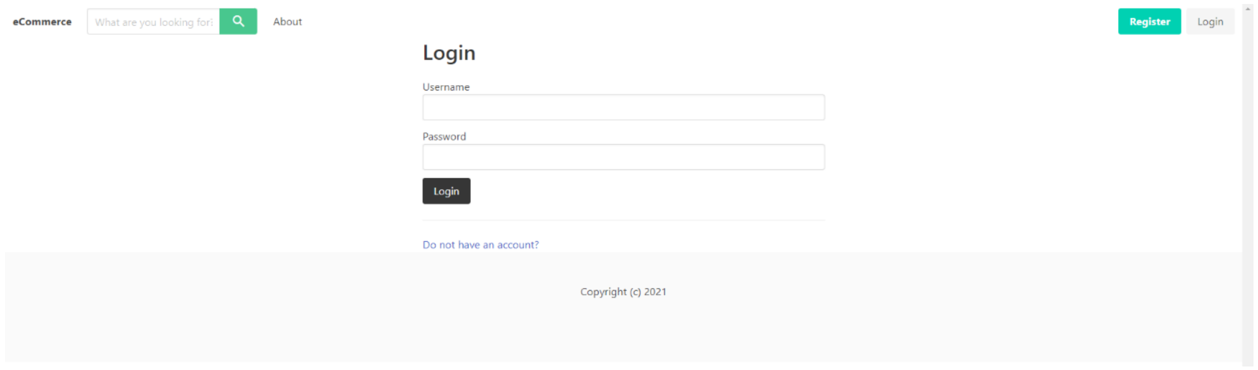


Figure 4 – registration process

Figure 4 after user press register button , user shell complete registration process by enter his data



Login

Username

Password

Login

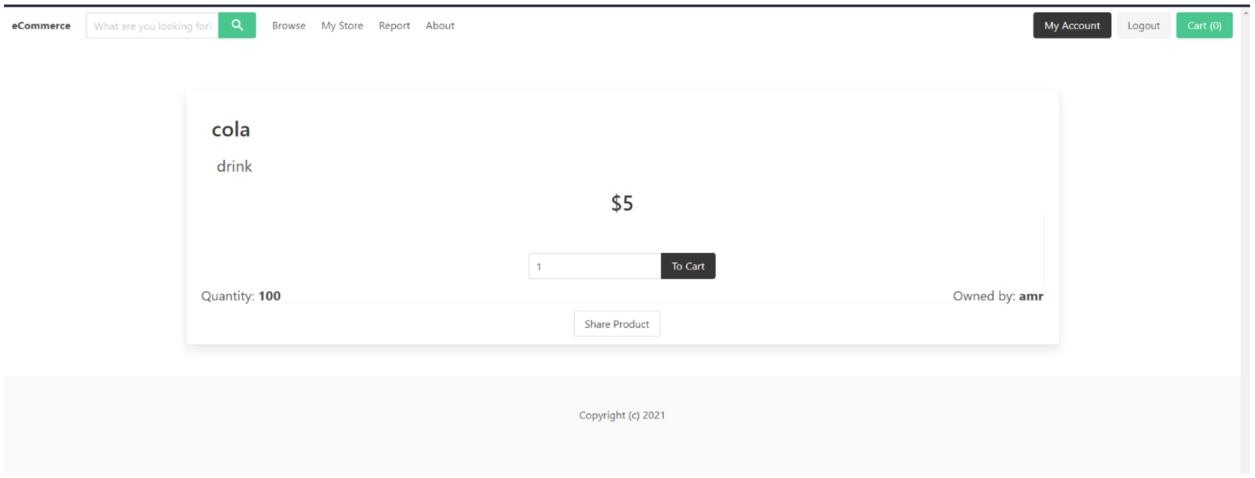
Do not have an account?

Copyright (c) 2021

Figure 5 – login process

figure 5 after user complete registraion process , user can login to his account by entering his username and password and press login button

- **Purchases item from another user:**



cola

drink

$5

1 To Cart

Quantity: **100**

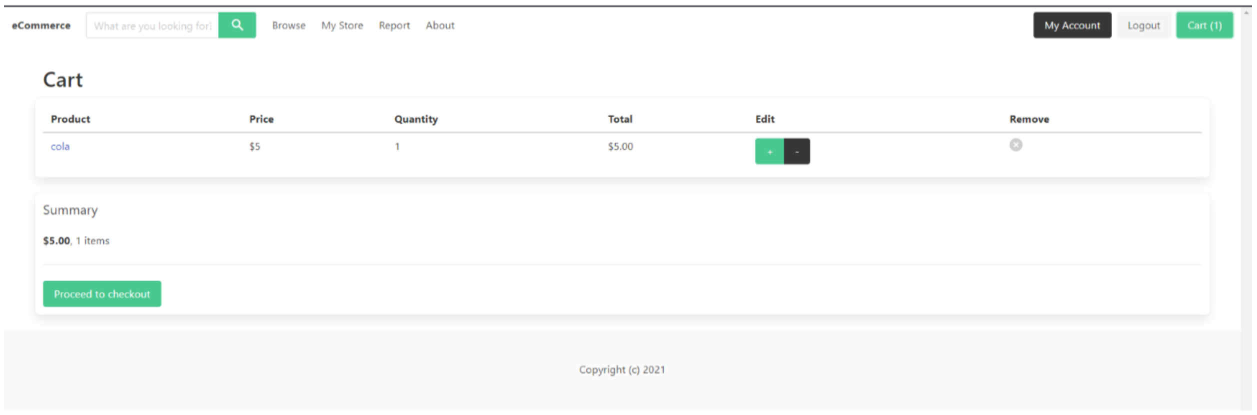
Owned by: **amr**

Share Product

Copyright (c) 2021

Figure 6 – item cart

Figure 6 after choosing item by user, user shell choose the amount of item and press to cart



Cart

Product	Price	Quantity	Total	Edit	Remove
cola	$5	1	$5.00	+ -	

Summary

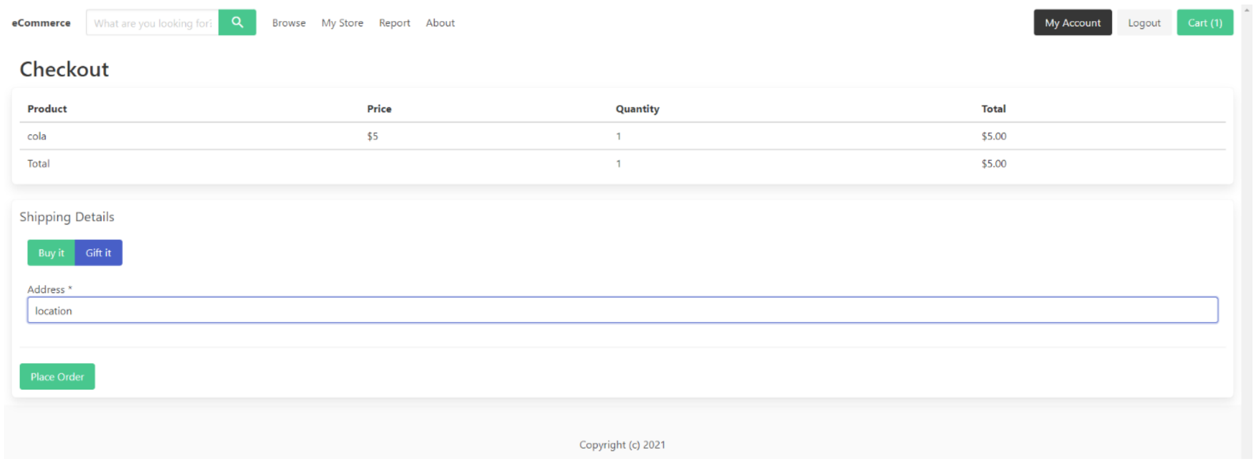
$5.00, 1 items

Proceed to checkout

Copyright (c) 2021

Figure 7 – user cart

Figure 7 the user cart after adding items, user can show his items from cart and user can press to checkout by press proceed to checkout button



Checkout

Product	Price	Quantity	Total
cola	$5	1	$5.00
Total		1	$5.00

Shipping Details

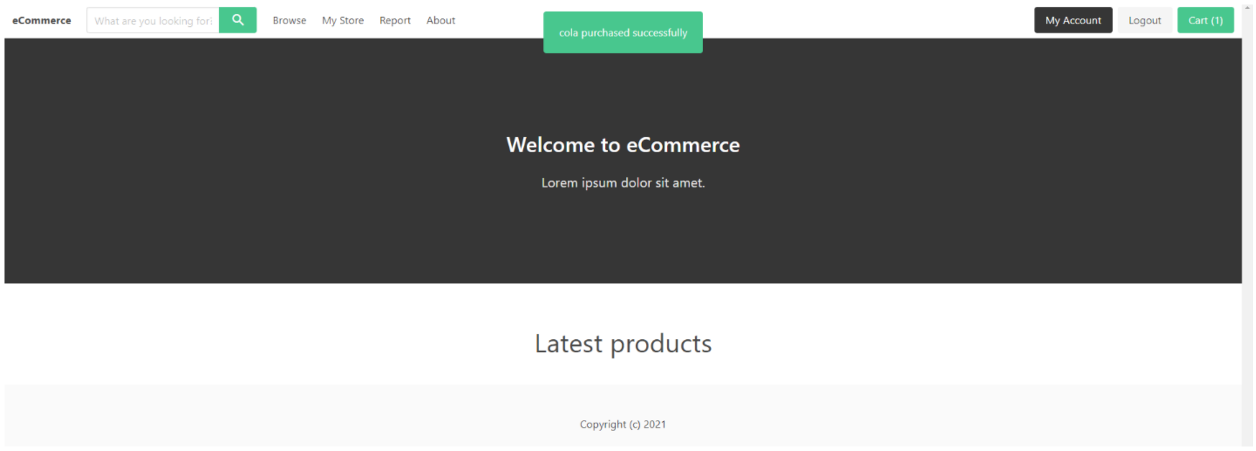
Address *

location

Copyright (c) 2021

Figure 8 - checkout

Figure 8 checkout process, user can complete Purchases process by press buy it button and add his address in address bar and press place order



cola purchased successfully

Welcome to eCommerce

Lorem ipsum dolor sit amet.

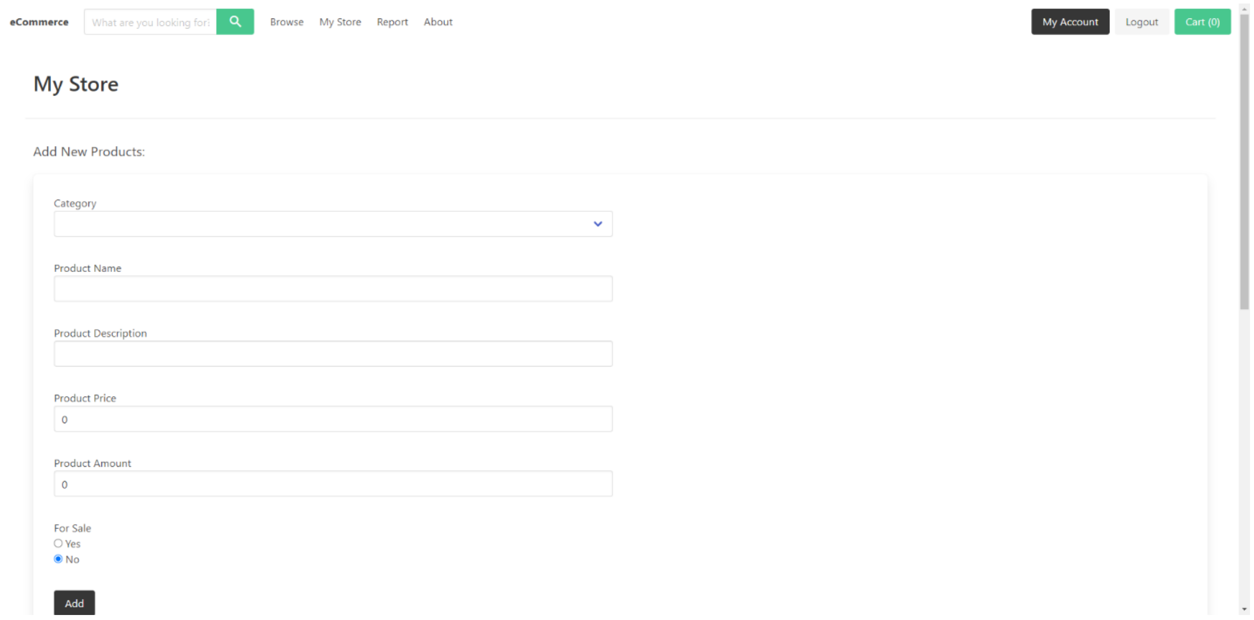
Latest products

Copyright (c) 2021

Figure 9 – purchased successfully

Figure 9 after Purchases process complete

- **Add item to my store:**



My Store

Add New Products:

Category:

Product Name:

Product Description:

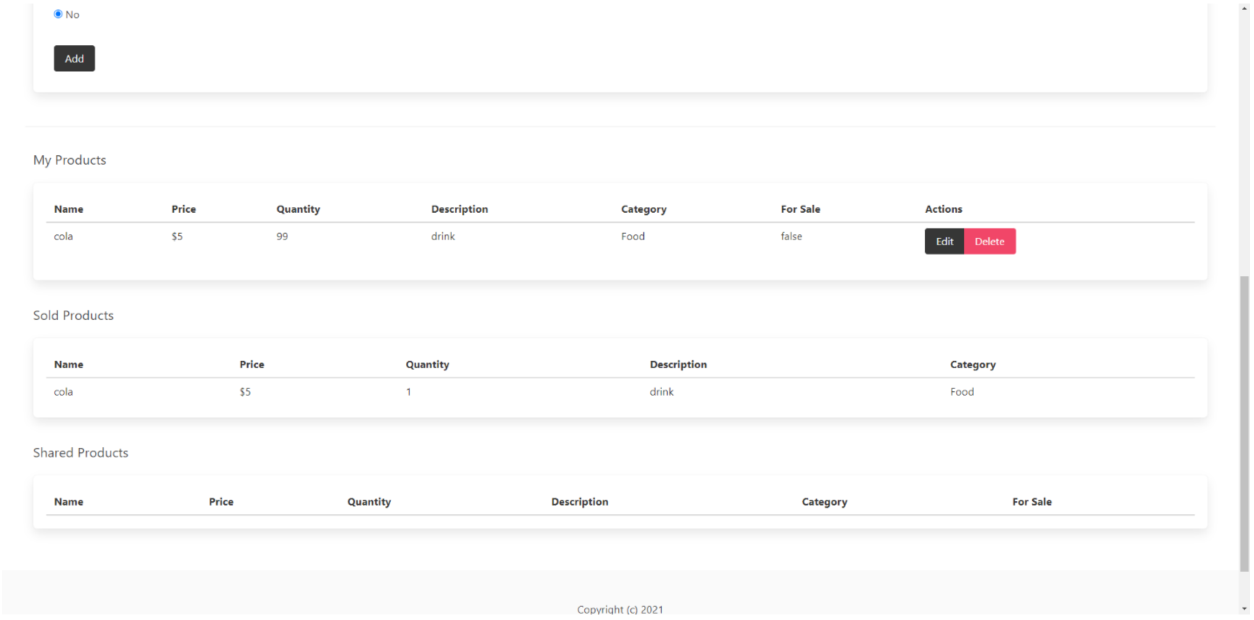
Product Price: 0

Product Amount: 0

For Sale: Yes [✓ No]

Figure 10 - add item

Figuer 10, user can add item in his store by fill its required data and press add button



My Products

Name	Price	Quantity	Description	Category	For Sale	Actions
cola	$5	99	drink	Food	false	Edit Delete

Sold Products

Name	Price	Quantity	Description	Category
cola	$5	1	drink	Food

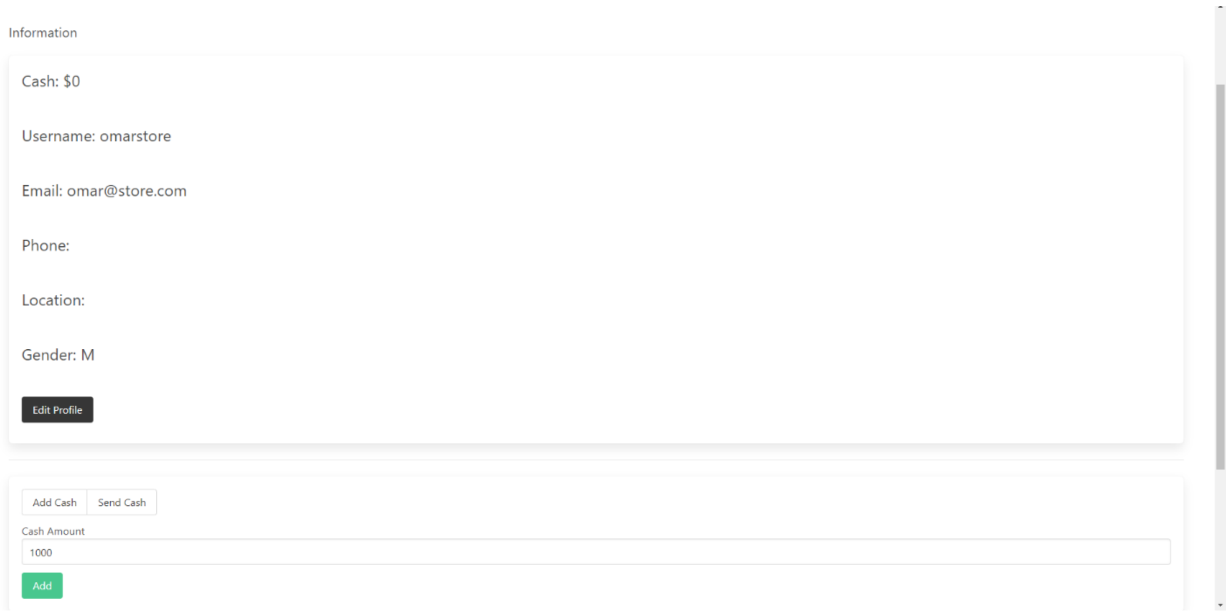
Shared Products

Name	Price	Quantity	Description	Category	For Sale

Figure 11 - user products

Figuer 11, user can find his products list

- **Update user information:**



Information

Cash: $0

Username: omarstore

Email: omar@store.com

Phone:

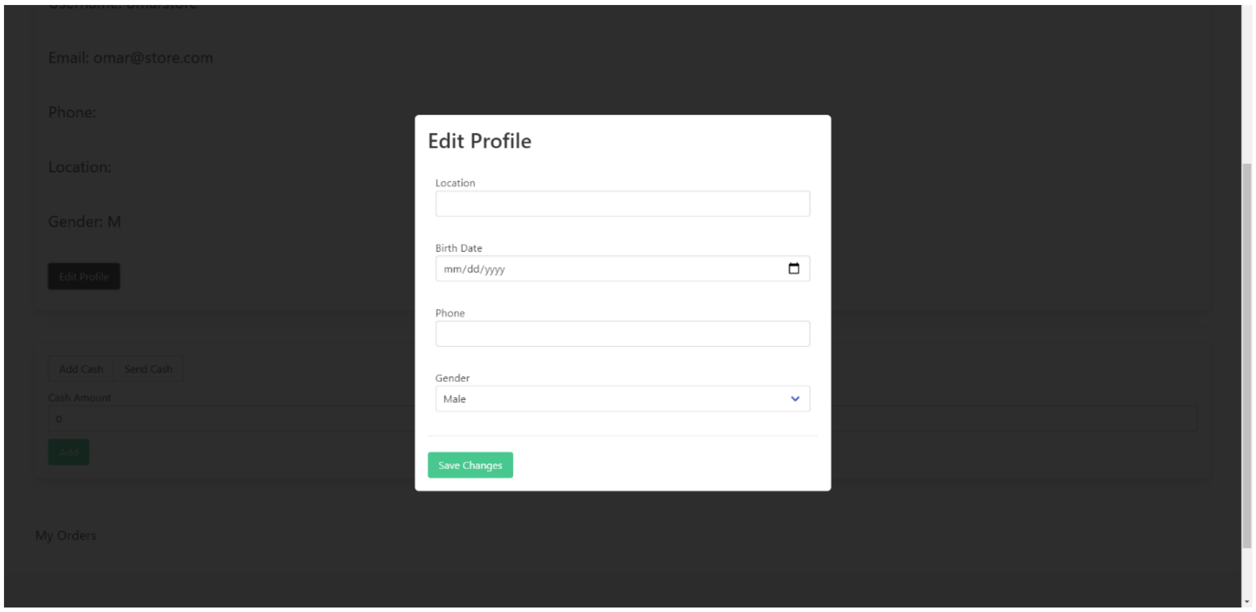
Location:

Gender: M

Cash Amount: 1000

Figure 12 - user information

Figure 12, user can see his information and can edit it by press edit profile button or add cash by enter cash value and press add button or send cash



Edit Profile

Location:

Birth Date: mm/dd/yyyy

Phone:

Gender: Male

Figure 13 - edit profile

Figure 13 user can edit his profile information and save it by press save changes button

10. Resources Needed:

1. https://v3.vuejs.org/guide/introduction.html

2. https://bulma.io/documentation/

3. https://docs.djangoproject.com/en/4.0/

4. https://www.django-rest-framework.org/api-guide/requests/

5.https://docs.mongodb.com/manual/sharding/#:~:text=Sharding%20is%20a%20method%20for,sets%20and%20high%20throughput%20operations.&text=Horizontal%20Scaling%20involves%20dividing%20the,to%20increase%20capacity%20as%20required

11. Role of Each Member:

team members name	***role***
Loay Abdalla Youssef Abdelhady Elshall	*Front-end + report*
Omar Mohamed Omar Abd El Samea	*Back-end + Database*
Mohamed Khaled Sayed Ibrahim Mohran	*Back-end + report*
Amr Ahmed Mohamed Fathy	*Front-end + report*
Omar Mohamed Mohamed Mostafa	*Back-end + Database*

12. Appendices:

All required information is described in each point

13. References:

1. https://v3.vuejs.org/guide/introduction.html

2. https://bulma.io/documentation/

3. https://docs.djangoproject.com/en/4.0/

4. https://www.django-rest-framework.org/api-guide/requests/

5.https://docs.mongodb.com/manual/sharding/#:~:text=Sharding%20is%20a%20method%20for,sets%20and%20high%20throughput%20operations.&text=Horizontal%20Scaling%20involves%20dividing%20the,to%20increase%20capacity%20as%20required