Documentation of flow of the project KitchenStory website :

Document and project specification-

Project name- Kitchen Story website

Developer name- Rahul Rana

GitHub Account name- Rahulpersie66

Github repostitry link-

- **Front End of Kitchen story:**
  - https://github.com/Rahulpersie66/project-4-Kitchen_Story
- **Back end of Kitchen Story:**
  - https://github.com/Rahulpersie66/Project-Kitchen-Story-Backend/tree/master/kitchen-story

IDE- Eclipse IDE used for BackEnd and Visual studio for Front end

Front End- Angular is used for front end of Kitchen story

Tool – Maven used for download dependency

Plug-in – Maven compiler plugin and maven war (webapp archive) plugin

Dependency – MySql connector 8(mysql-connector-java), springbootstarter-jpa, spring-boot-devtools, spring-boot-configuration-tools

Programming language- JAVA is used ,typeScript, Html, Css, MySql, Bootstrap

Database name- fun

Table names- Kitchen_User_registered,Kitchen_Product,Kitchen_order_booked

Output of the project – Can be seen on any browser on running angular project

Server- Any server(because spring have inbuild server)

For dependency file- pom.xml for downloading the dependency

MySql connecting done in file- application.properties in Eclipse

Bootstrap- added via CDN copy and paste

Java™

eclipse

GitHub

git

Maven™

MySQL®

spring®

spring boot

Visual Studio

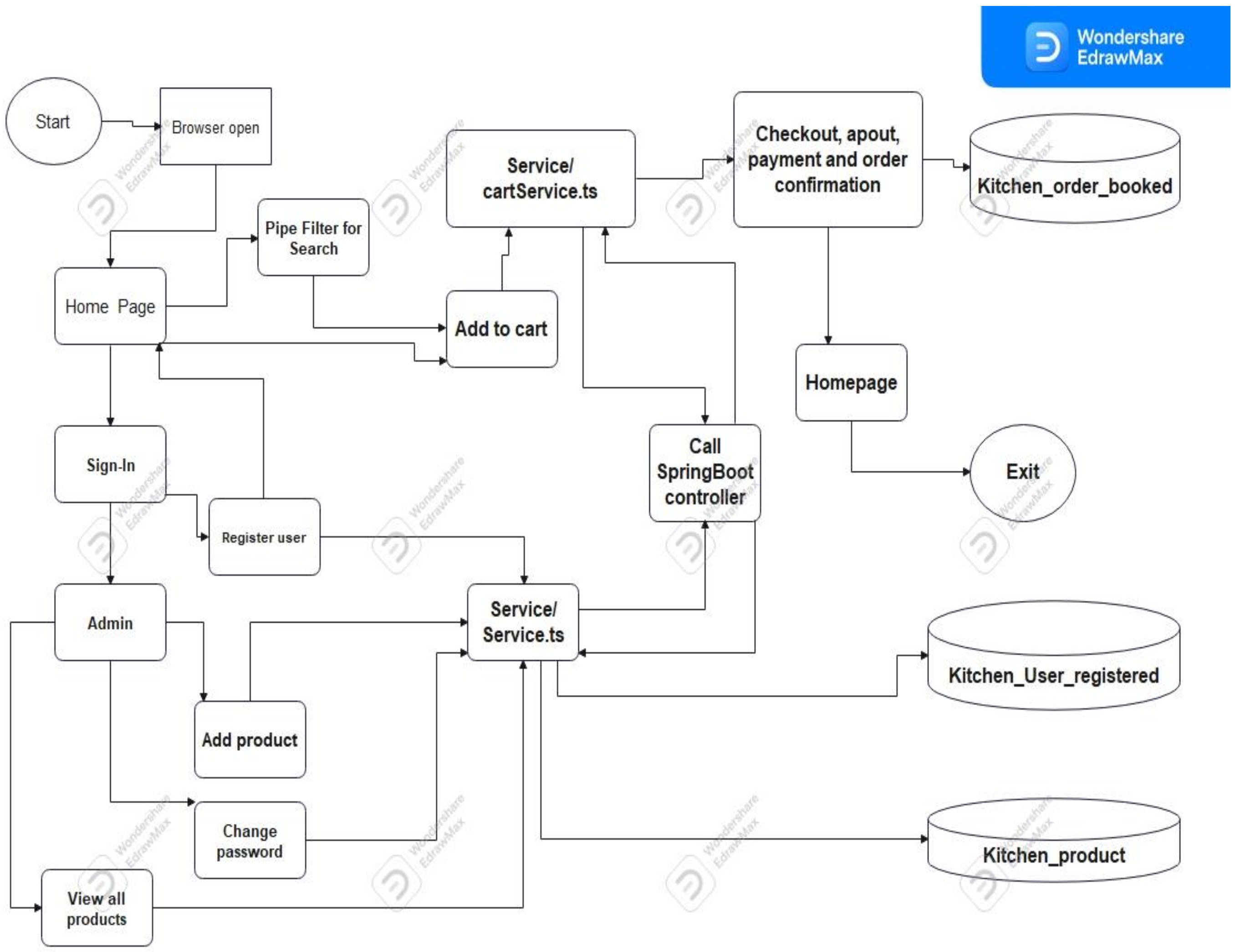
B Bootstrap

HTML

CSS

TS TypeScript

Data flow diagram according to algorithm-



Algorithm -

Step -1 Create a Spring boot project using spring starter project which uses spring.io behind the scene for spring boot project creation.

Step -2 Add required dependency to **pom.xml** file and set **application.properties** file in **src/main/resource** for database configuration.

Step -3 In the main driver class present in the pkg:**src/main/java/com.spring.boot.kitchenstory KitchenStoryApplication.java** add print statement "kitchen story" that will be print on the console.

Step -4 Now create 5 different packages for 5 different operation- table formation(entities), for crud operation(daos), for define services(service interface), for working of services(serviceImpls) and for sending commands(controllers).

Step -5 In entities define all table that are going to be stored in the database using @Entity @Table(name) and define their primary key if needed.

Step -6 Now create Repository for each table by extending repository to JPA Repository for implementing basic operation like add, delete , findall and find one. This class will be marked as @Repository.

Step -7 Now create Services for each table and define services required by both controller admin and user. Remember to make this interface and just write each services definition only.

Step -8 In next package ServiceImpls- implementation each service in details that are required by both the controller and this class will be marked as @Service.

Step -9 In the last package Controller what services require by user or client will be define here alongside with their mapping and will call the respective service from serviceImpl class. This class will be marked as @RestController and mapping of url to services will be done here.

Step -10 Now run the driver class and it will run on the port: **8080** after that construct a Angular project for front end of same application. By command **ng new KitchenStory**.

Step-11 Add bootstrap via cdn and construct various component for the project to perform various services.

Step 12- After that create **service** service of angular to make use of get, post,put,delete ffunctions defined in the springboot project. Andfor search use the **pipe** to filter the result andto produce search navbar output.

Step 13- To make this project run make use of **@CrossOrigin("*")** to make connection between both springboot and angular project.

Step 14- Run the command **ng serve –o** in the console of angular to run the project at port:**4200**.

Step by Step process from sprint planning to product release-

Sprint 1-

Sprint Planning –In this Sprint, all the backend project is done that is defining of which table be to use for storing data and what are the function and service to be define in the controller.

Objective of sprint 1-

-to construct a table in MySql using Annotation way and mark each table @Entity

-Now connect the each table Repository class by extending it to JPA Repository and provide class name of table class alongside with primary key datatype. And mark these class as @Repository.

```
@Repository
public interface KitchenUserDao extends JpaRepository<KitchenUserRegistered, Integer>
{
//all basic CRUD operation
}
```

-after that all services are implemented and these implemented interface are implement by service classes.

```
public interface KitchenUserService {

	public boolean addUserSignUp(KitchenUserRegistered user);

	public List<KitchenUserRegistered> getAllUser();

	public KitchenUserRegistered getUser(int user_id);

	public KitchenUserRegistered updateUserPassword(int user_id,
KitchenUserRegistered user);

}
```

-In service Class all the services is defined clearly.

```

@Service
public class KitchenUserServiceImpl implements KitchenUserService {

    @Autowired
    private KitchenUserDao userDao;

    //addd the USer signUp
    @Override
    public boolean addUserSignUp(KitchenUserRegistered user) {
```

```java
        boolean flag=false;
        userDao.save(user);
        flag=true;
        return flag;
    }

    @Override
    public List<KitchenUserRegistered> getAllUser() {
        return userDao.findAll();
    }

    @Override
    public KitchenUserRegistered getUser(int user_id) {
        return userDao.findById(user_id).get();
    }

    @Override
    public KitchenUserRegistered updateUserPassword(int user_id, KitchenUserRegistered
user) {
        KitchenUserRegistered user1=userDao.findById(user_id).get();
        user1.setPassword(user.getPassword());
        user1.setFullName(user.getFullName());
        user1.setEmail(user.getEmail());
        user1.setAddress(user.getAddress());
        return userDao.save(user1);
    }

}
```

-After that controller is defined to call these services using get, post mapping and Autowired component or variable defined.

```java

@RestController
@CrossOrigin("*")
public class AdminController {

    @Autowired
    private KitchenUserServiceImpl userServiceImpl;
    @Autowired
    private KitchenProductServiceImpl productServiceImpl;
    @Autowired
    private KitchenOrderBookedServiceImpl orderserviceImpl;
    //add the sign up user
    @PostMapping("/admin/addedSignUpUser")
    public String addSignUser(@RequestBody  KitchenUserRegistered user)
    {   if(userServiceImpl.addUserSignUp(user))
            return "User added successfully";
        else
            return "some error occurred";
    }

    //get all the Signed up user
    @GetMapping("/admin/showAllUser")
    public List<KitchenUserRegistered> getAllUser(){
        return userServiceImpl.getAllUser();
    }

    //get user by id
    @GetMapping("/admin/changePassword/{user_id}")
    public KitchenUserRegistered getUser(@PathVariable int user_id) {
        return userServiceImpl.getUser(user_id);

    }

    //update the password
    @PutMapping("/admin/updatePassword/{user_id}")
    public KitchenUserRegistered updateUserPassword(@PathVariable int
user_id,@RequestBody KitchenUserRegistered user) {
        return userServiceImpl.updateUserPassword(user_id, user) ;
    }

    //Add the product details to API
    @PostMapping("/admin/addProductDetails")
    public KitchenProduct addTheProduct(@RequestBody KitchenProduct product)
    {
        if(productServiceImpl.addProduct(product))
            return product;
        else
            return null;
    }

    //get Productby ID
    @GetMapping("/admin/product/{productID}")
    public KitchenProduct getProductById(@PathVariable int productID) {
        return productServiceImpl.getProductById(productID);
    }
    //get all the Products
    @GetMapping("/admin/showProducts")
    public List<KitchenProduct> getAllProducts(){
        return productServiceImpl.getAllProducts();
    }

    //update the product by productId
    @PutMapping("/admin/upadte/{productId}")
    public KitchenProduct updateProduct(@RequestBody KitchenProduct
product,@PathVariable int productId) {
        return productServiceImpl.updateProductById(product, productId);
    }


    //Delete a product
    @DeleteMapping("/admin/deleteProduct/{productId}")
    public String deleteProduct(@PathVariable int productId) {
        if(productServiceImpl.deleteProductById(productId))
        return "product is deleted";
        else
            return "not deleted";
    }

    //Order booked
    @PostMapping("/checkout/orderBooked")
    public KitchenOrderBooked bookOrder(@RequestBody KitchenOrderBooked order) {
        return orderserviceImpl.addOrder(order);
    }

    //get OrderId
    @GetMapping("/checkout/{orderId}")
    public KitchenOrderBooked getOneOrder(@PathVariable int orderId) {
        return orderserviceImpl.getOneOrder(orderId);
    }

    @PutMapping("/checkout/{orderId}")
    public KitchenOrderBooked updateOrderById(@PathVariable int orderId,@RequestBody
KitchenOrderBooked order)
    {
        return orderserviceImpl.updateOrder(orderId, order);
    }

    @GetMapping("/allOrder")
    public List<KitchenOrderBooked> getAllOrder(){
        return orderserviceImpl.getAllOrder();
    }
}
```

Process & development work-

Each table –**Kitchen_user_registered, Kitchen_product,Kitchen_order_booked** will be define alongside with their column name and their datatype and define setter and getter method with constructor for each field.

- It will create table with respective table name in mysql alongside their column.

-DAO interface will use JPA repository to modify or alter or making change in the sql tables.

-Services will be using these JPA commands on the mysql table and define functions which can controller perform.

-Controller will call the various services depending on the user / client need.

Sprint retrospective-

Still we need to implement the frontend of the KitchenStory to make it user interactive and to make it fully working website.

Sprint 2-

Sprint planning- In this phase of sprint 2, I build the Angular KitchenStory project using command **ng new KitchenStory-e-commerce/KitchenStory** , after that it will construct angular files folder.

Sprint Objective-

-In **Index.html** file we will paste Bootstrap cdn to make use of bootstrap further.

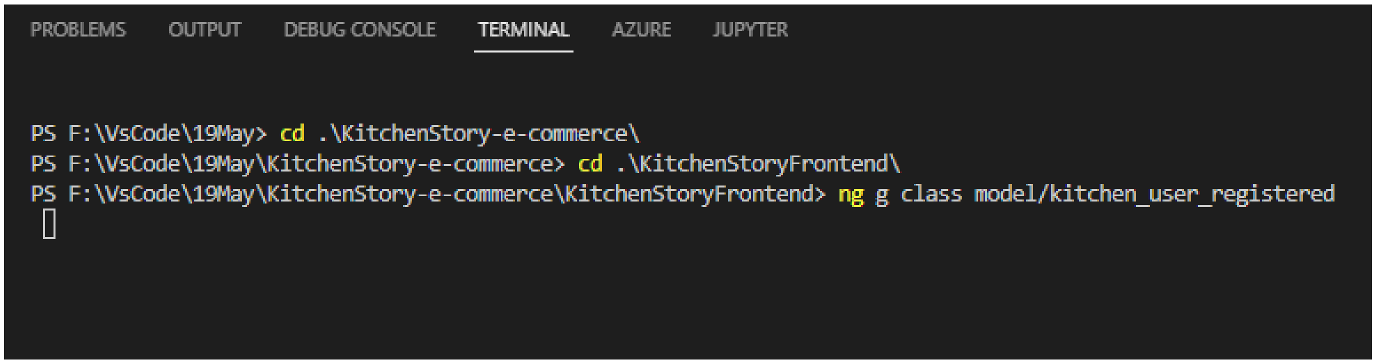
- Then inside app folder construct **model** to save , use data present in the table of mysql. So model can be created using command **ng g class model/kitchen_user_registered.**

-Then inside app folder of Angular construct various component required by the Kitchen Story website. You can create component **ng g c home/home**.

-And then develop a service using **ng g s service/service** to make use of all the services define in the springboot class.

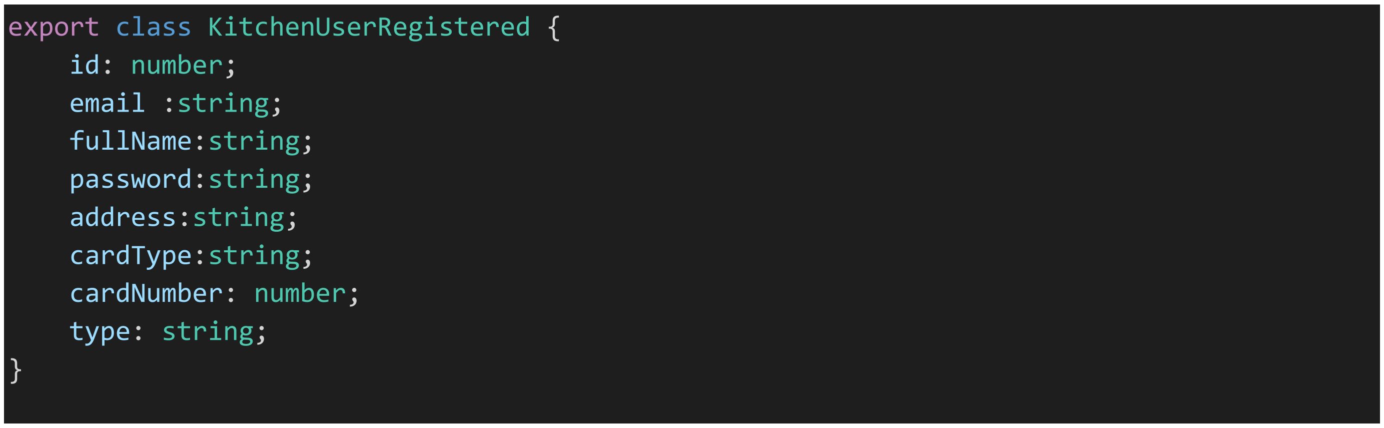
Process and development work-

-After creating model class in angular enter all data member name that are used in Springboot table to store data or manipulate data of table in mysql. The model class will construct typescript file in which we have todefine all column/variable name used for table:**Kitchen_user_registered**, **kitchen_product**, **Kitchen_order_booked.**



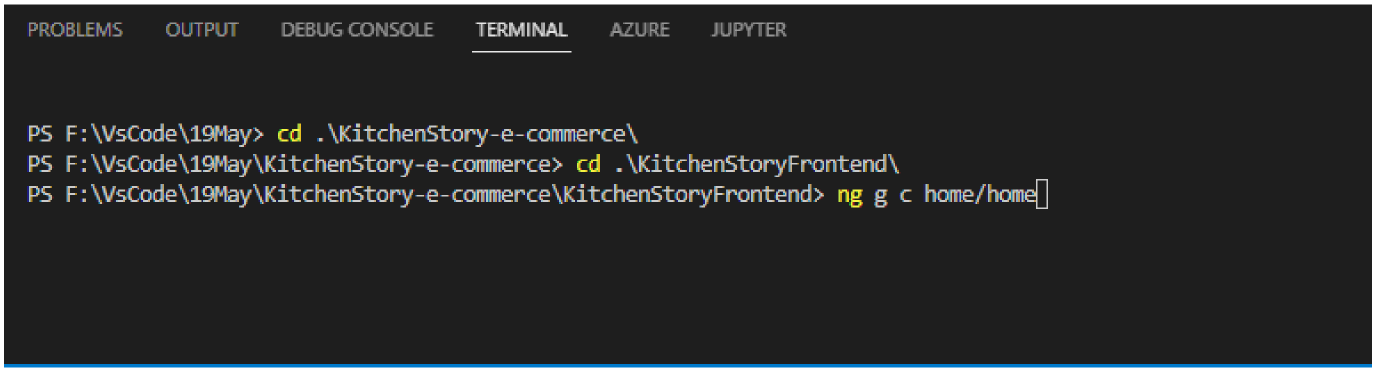
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE JUPYTER

```
PS F:\VsCode\19May> cd .\KitchenStory-e-commerce\
PS F:\VsCode\19May\KitchenStory-e-commerce> cd .\KitchenStoryFrontend\
PS F:\VsCode\19May\KitchenStory-e-commerce\KitchenStoryFrontend> ng g class model/kitchen_user_registered
```



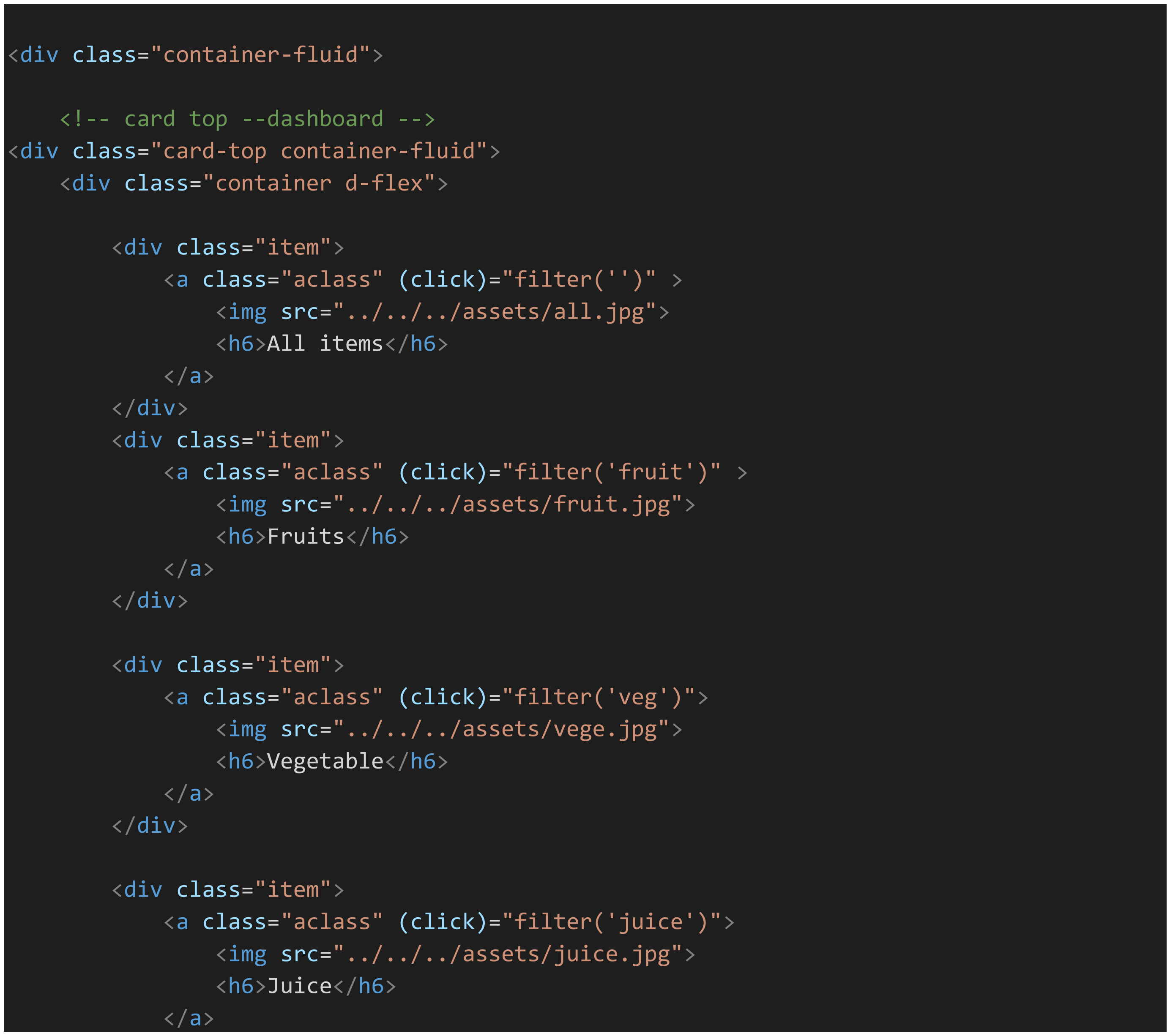
```
export class KitchenUserRegistered {
    id: number;
    email :string;
    fullName:string;
    password:string;
    address:string;
    cardType:string;
    cardNumber: number;
    type: string;
}
```

-For component define similarly we will use **ng g c home/home** which will generate 4 type of files- html, css, typescript and typescript for testing. Out of which we will define component interface in **html** file and styling of that component will be done in **css** file and and calling of service and other technical work like **onSubmit(), isClicked(), ngOnInIt()** will be defined in the typescript class.



```
PS F:\VsCode\19May> cd .\KitchenStory-e-commerce\
PS F:\VsCode\19May\KitchenStory-e-commerce> cd .\KitchenStoryFrontend\
PS F:\VsCode\19May\KitchenStory-e-commerce\KitchenStoryFrontend> ng g c home/home
```

Html file of home-



```
<div class="container-fluid">

    <!-- card top --dashboard -->
<div class="card-top container-fluid">
    <div class="container d-flex">

        <div class="item">
            <a class="aclass" (click)="filter('')" >
                <img src="../../../assets/all.jpg">
                <h6>All items</h6>
            </a>
        </div>
        <div class="item">
            <a class="aclass" (click)="filter('fruit')" >
                <img src="../../../assets/fruit.jpg">
                <h6>Fruits</h6>
            </a>
        </div>

        <div class="item">
            <a class="aclass" (click)="filter('veg')">
                <img src="../../../assets/vege.jpg">
                <h6>Vegetable</h6>
            </a>
        </div>

        <div class="item">
            <a class="aclass" (click)="filter('juice')">
                <img src="../../../assets/juice.jpg">
                <h6>Juice</h6>
            </a>
        </div>
    </div>
</div>

            <div class="container-fluid" >
                <div class="row my-3 ">
                    <div  class="card-down col-lg-3 col-md-4 col-sm-6 border border-
3  bg-dark text-white "
                     *ngFor="let pro of filterCategory | pipe : searchKey : 'prodName'
">

                    <img src="{{pro.imgUrl}}" routerLink="/">
                     <!-- Text inside  -->
                        <h1><b> {{pro.prodName}}</b><br></h1>
                            <p><i>{{pro.vendorName}}</i><br>
                            ₹{{pro.price}}/-<br>
                            {{pro.prodDetail}}</p><br>
                     <!-- Nutton of the card -->
                            <button class="btn btn-warning m-1
"  (click)="addToCart(pro)"><b>Add to cart</b></button>
                             <br>

                    <div>


                </div>

            </div>

    <!--      <div *ngElse class="col-lg-3 col-md-4 col-sm-6  bg-dark text-white ">Column
2</div>
            <div class="col-lg-3 col-md-4 col-sm-6  bg-primary ">Column 3</div>
            <div class="col-lg-3 col-md-4 col-sm-6  bg-dark text-white">Column 4</div>
            <div class="col-lg-3 col-md-4 col-sm-6  bg-primary ">Column 5</div>
            <div class="col-lg-3 col-md-4 col-sm-6  bg-dark text-white ">Column 6</div>
-->

</div>
```

Css file for the same-

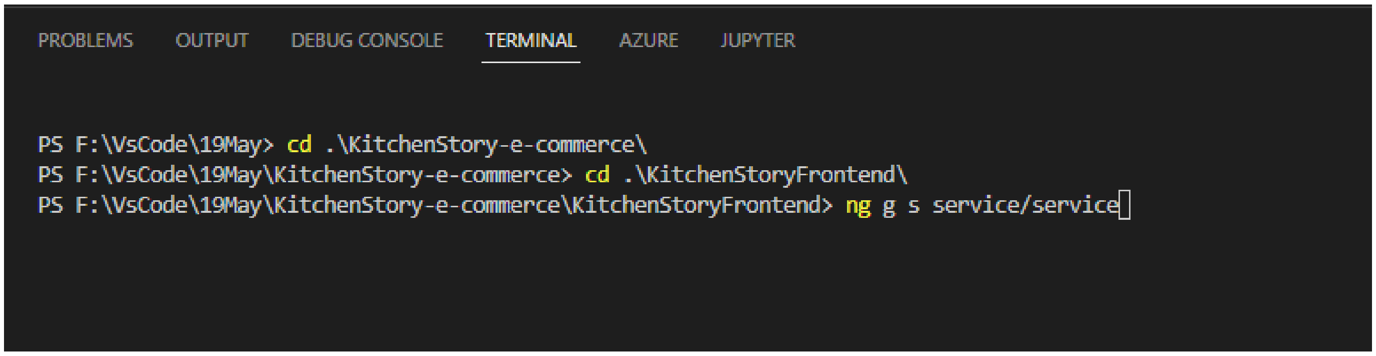
```

.card-top{
    position: relative;
```

```css
    display:flex;
    padding-top: 0%;
    flex-direction: column;
    min-width: 0;
    word-wrap: break-word;
    background: #fff;
    background-clip: border-box;
    border:1px solid rgba(0,0,0,0.2);
    border-radius: .25rem;
}
.item img{
width: 50px;
height: 50px;
border-radius: 50%;
}

.card-down img{
width: 150px;
height: 150px;

}


.item{
    margin: 0px 15px;
    text-align: center;
}
.aclass:hover{
    color: royalblue;
}

.aclass{
    color: black;
}
.card-down{
    text-align: center;

}
.card h1{
    align-content: center;
}
.card-down img{
    margin-top: 2%;
    margin-left: 1%;
    align-content: center;
    margin-bottom: 3%;
    transition: 0.3s ease-in-out;
}

.card-down img:hover{
    transition: 0.3s ease-in;
    transform: scale(1.05);
}
```

Type script file for the home/home-

```
import { Component, OnInit } from '@angular/core';
import { KitchenProduct } from 'src/app/model/kitchen-product';
import { KitchenUserRegistered} from 'src/app/model/kitchen-user-registered';
import { CartService } from 'src/app/service/cart.service';
import { ServiceService } from 'src/app/service/service.service';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {

  public filterCategory :any;
  searchKey :string="";
  allUser: KitchenUserRegistered[];
  allProduct:KitchenProduct[];
  constructor(private service:ServiceService,private cartService :CartService) { }

  ngOnInit(): void {
    this.getAllProduct();

    //here that searc func defined
    this.cartService.search.subscribe(val=>{this.searchKey=val;})
  }

  search(event:any){}
  private getAll(){
    this.service.getAllUser().subscribe(
      data=>{this.allUser=data},
    error=>console.log(error));
  }

  //indide this add to cart QTy and total is added
  private getAllProduct(){
    this.service.getAllProduct()
    .subscribe(data=>{
      this.allProduct=data;
      this.filterCategory=data;
      //set type  of product here if not set
      this.allProduct.forEach((a:any)=>{
       Object.assign(a,{quantity:1,total:a.price});

    });
  })
  }

  //this add to cart items
  addToCart(item:any){
       this.cartService.addToCart(item);
  }


  //to check if type is == "veg"
  filter(type:string){
    this.filterCategory=this.allProduct.filter((a:any)=>{
      if(a.type===type || type=='')
        return a;

    });
  }

}
```

-So for defining service class we can use **ng g s service/service** and it will generate 2 file one typescript file for doing service task and other for testing.



```
PS F:\VsCode\19May> cd .\KitchenStory-e-commerce\
PS F:\VsCode\19May\KitchenStory-e-commerce> cd .\KitchenStoryFrontend\
PS F:\VsCode\19May\KitchenStory-e-commerce\KitchenStoryFrontend> ng g s service/service
```

TypeScript file for service where all the http request are defined and url of the springboot get, post request is given.

```
import { HttpClient, HttpClientModule } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { KitchenOrderBooked } from '../model/kitchen-order-booked';
import { KitchenProduct } from '../model/kitchen-product';
import { KitchenUserRegistered } from '../model/kitchen-user-registered';

@Injectable({
  providedIn: 'root'
})
export class ServiceService {

  url= 'http://localhost:8080';

  constructor(private httpClient:HttpClient) { }

  getAllUser():Observable<KitchenUserRegistered[]>{
    return
this.httpClient.get<KitchenUserRegistered[]>(`${this.url}/admin/showAllUser`);
  }

  getAllProduct():Observable<KitchenProduct[]>{
    return this.httpClient.get<KitchenProduct[]>(`${this.url}/admin/showProducts`);
  }

  addUser(user:KitchenUserRegistered) :Observable<Object>{
    return
this.httpClient.post(`${this.url}/admin/addedSignUpUser`,user,{responseType:'text'});
  }

  addProduct(product:KitchenProduct):Observable<Object>{
    return this.httpClient.post(`${this.url}/admin/addProductDetails`,product);
  }

  getProductByProductId(productId:number):Observable<KitchenProduct>{
    return
this.httpClient.get<KitchenProduct>(`${this.url}/admin/product/${productId}`);
  }

  getUserById(userId:number):Observable<KitchenUserRegistered>{
    return
this.httpClient.get<KitchenUserRegistered>(`${this.url}/admin/changePassword/${userId}`)
;
  }

  getOrderById(orderId:number):Observable<KitchenOrderBooked>{
    return this.httpClient.get<KitchenOrderBooked>(`${this.url}/checkout/${orderId}`);
  }

  updateProduct(productId:number,product:KitchenProduct):Observable<KitchenProduct>{
    return
this.httpClient.put<KitchenProduct>(`${this.url}/admin/upadte/${productId}`,product);
  }

  updateUser(userId:number,user:KitchenUserRegistered):Observable<KitchenUserRegistered>
{
    return
this.httpClient.put<KitchenUserRegistered>(`${this.url}/admin/updatePassword/${userId}`,
user);
  }

  updateOrderById(orderId:number,order:KitchenOrderBooked):Observable<KitchenOrderBooked
>{
    return
this.httpClient.put<KitchenOrderBooked>(`${this.url}/checkout/${orderId}`,order);
  }

  deleteProduct(productId:number):Observable<Object>{
    return
this.httpClient.delete(`${this.url}/admin/deleteProduct/${productId}`,{responseType:'tex
t'});
  }

  addOrderBooked(order : KitchenOrderBooked):Observable<Object>{
    return this.httpClient.post(`${this.url}/checkout/orderBooked`,order);
  }
  getAllOrder():Observable<KitchenOrderBooked[]>{
    return this.httpClient.get<KitchenOrderBooked[]>(`${this.url}/allOrder`);
  }

}
```

Sprint Retrospective-

In this sprint 2, the project is still missing the search option in navbar functioning and running of the project on port:**4200** and url : http://localhost:4200/ .

Sprint 3

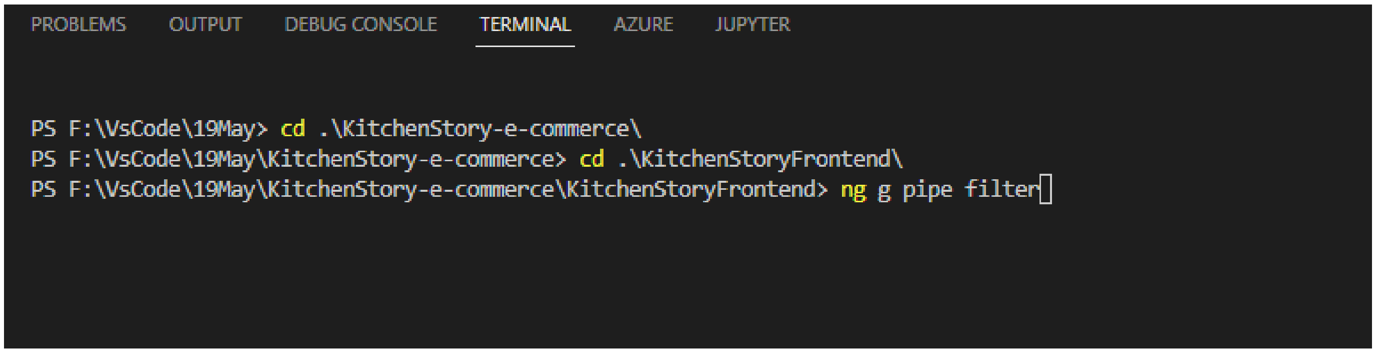
Sprint Objective-

-Finally we will construct the **pipe and filter** for search textfield in the navbar, whose function is to give the product on entering the product name iin the search navbar.

-And in last run the Website on port:4200 using command **ng serve –o**.

Process and development work-

-for creation of pipe run the command **ng g pipe filter** which will create filter class and produce 2 typeScript files, one for testing and for implementing filter for search bar.



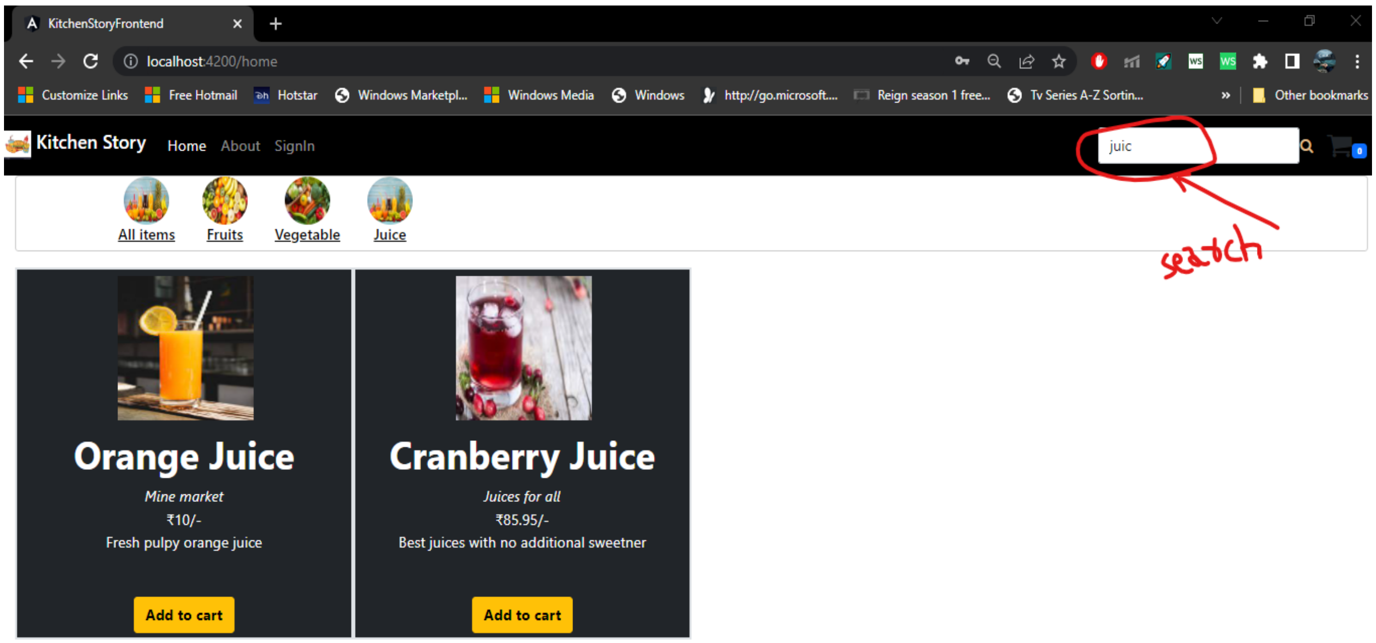
```
PS F:\VsCode\19May> cd .\KitchenStory-e-commerce\
PS F:\VsCode\19May\KitchenStory-e-commerce> cd .\KitchenStoryFrontend\
PS F:\VsCode\19May\KitchenStory-e-commerce\KitchenStoryFrontend> ng g pipe filter
```

Type script file for search-

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'pipe'
})
export class PipePipe implements PipeTransform {
```

```
transform(value:any[],filterString:string,propName:string): any[] {
   const result:any=[];
   if(!value||filterString===''|| propName==='')
       {
         return value;
       }

   value.forEach((a:any)=>{
           if(a[propName].trim().toLowerCase().includes(filterString.toLowerCase())){
             result.push(a);
           }
         });
         return result;
   }


}
```

And now working of search in the navabar is shown in below picture.

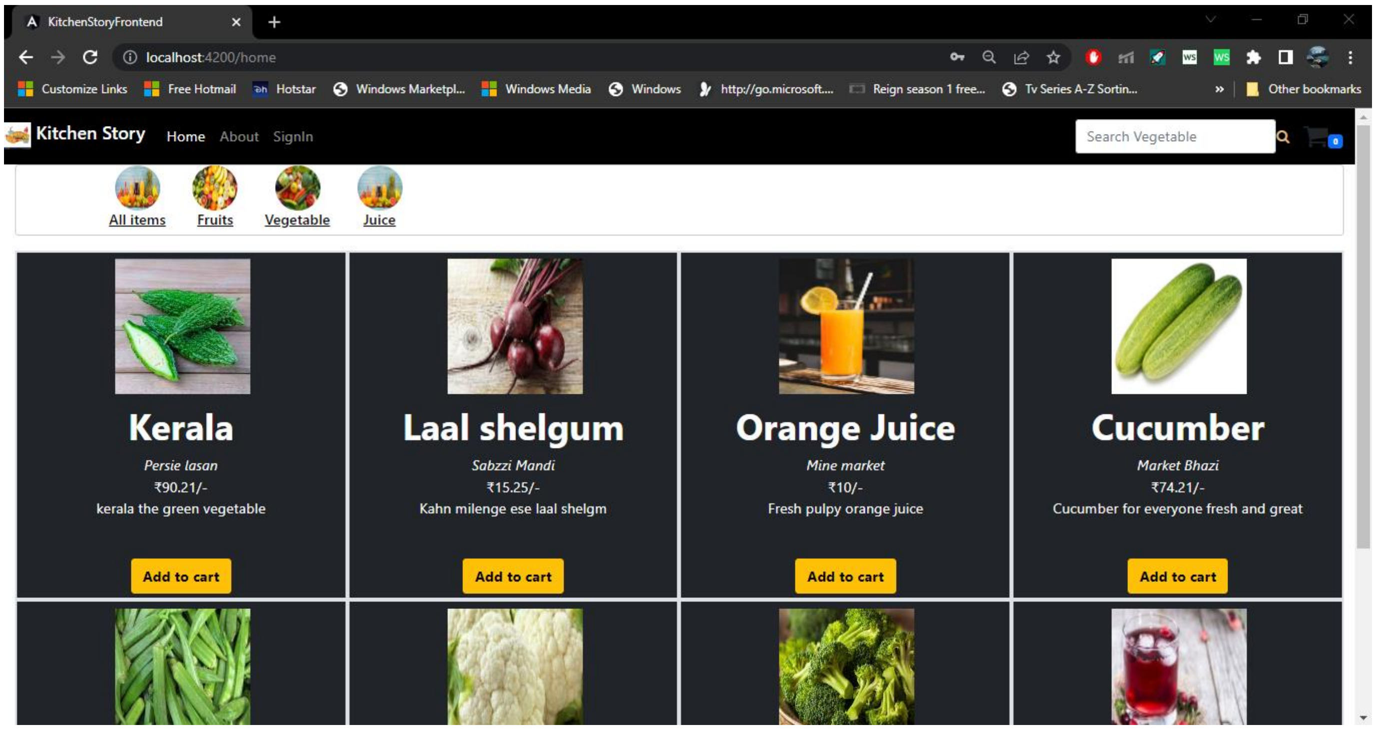

KitchenStoryFrontend

localhost:4200/home

Orange Juice
Mine market
₹10/-
Fresh pulpy orange juice
Add to cart

Cranberry Juice
Juices for all
₹85.95/-
Best juices with no additional sweetner
Add to cart

-now to run the project on web browser just enter the command – **ng serve –o** and the project will run on port:**4200**.



Sprint Retrospective-

In this sprint 3, still deploying of project using AWS on IP address is missing.

Conclusion-

This SpringBoot + Angular project had constructed a fully functional website from where we can buy product and pay for the product and all the data will be stored in the tables of MySQL.

Can enhance project by-

→Still deployment of the project to IP address is missing.

→ No Carousell(Sliding picture) in the website.

→No real payment gateway, the implemented one is dummy one.

→Product delivery data send to delivery partner is missing.

Unique selling points-

→You can purchase item directly after sign-In.

→For admin it is easy to use as it is user Interactive and can add product, change password, see list of product all on the same webpage.

→ Easy to operate as have similar functioning as like amazon, flipkart and other shopping website.