Online Cloth Buying Store

using Microservice Architecture

Major Project Report

Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Technology

in

Information Technology

By

Rutvik Borad

17BIT008

Department of Computer Science and Engineering

Institute of Technology

NIRMA UNIVERSITY

Ahmedabad 382481

CERTIFICATE

This is to certify that the Major Project Report entitled "**Online Cloth Buying Store using microservice architecture**" submitted by Mr. **Rutvik Vipulbhai Borad (17BIT008)** towards the partial fulfilment of the requirements for the award of degree in Bachelor of Technology in the field of Computer Engineering of Nirma University is the record of work carried out by him under our supervision and guidance. The work submitted has in our opinion reached a level required for being accepted for examination. The results embodied in this major project work to the best of our knowledge have not been submitted to any other University or Institution for award of any degree or diploma.

Date: _____

Dr Gaurang Raval
Associate Professor,
Department of Computer Science and Engineering,
Institute of Technology,
Nirma University
Ahmedabad

Dr Madhuri Bhavsar
Head of Department
Department of Computer Science and Engineering,
Institute of Technology,
Nirma University
Ahmedabad

Dr Rajesh Patel
Director,
Department of Computer Science and Engineering,
Institute of Technology,
Nirma University
Ahmedabad

Acknowledgement

I would like to thank my major project guide Dr Gaurang Raval for his consistent direction and help all throughout the project. I also thank my family and friends for the help and support.

I also would like to thank Nirma University & Dr Madhuri Bhavsar for giving me this amazing opportunity to build this project.

ABSTRACT

The Online Cloth buying store is a web based microservice application for sellers who want to broaden their business online. It follows the B2C - Business to Consumer Model. The focus area of the application is to make a scalable, fault tolerant web application to connect sellers and customers. It contains most of the functionality that a modern e-commerce portal has such as searching, viewing products , tracking orders, online payment options etc. It has a convenient search functionality for a user to search as per their need. For better performance it uses inverted indexing for retrieving the product documents. After the searching a user can view any product in detail from the retrieved list of products. In addition to product details a user can also see the rating and reviews of the product. One can write his/her own review along with the rating on any product. A user can become a seller and sell his/her own products on the portal and can also manage the orders of his/her products. Also based on the purchase history of a user, he/she will see some recommended products on the dashboard. The Main goal is to provide a platform for buyers and sellers to meet.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

Chapter 1: Introduction

1.1 Goal

Project definition is to build a scalable and fault tolerant online cloth store web app using microservice architecture. For years the e-commerce domain has been dominated by horizontal marketplaces like Amazon and Flipkart. But recently there has been an emergence of niche marketplaces on the e-commerce domain such as Myntra (clothes), PharmEasy (Pharmacy) and BigBasket (groceries) etc. These vertical marketplaces target a well- defined segment of population by analyzing the specific needs through targeted market research. As part of the Major project my goal of this web application is:

- To develop a scalable, low-latency and feature-rich clothing store web application which will allow customers to purchase clothes online with ease.
- A specialized web application for selling clothes will allow for targeted research and analysis generating greater revenue for sellers.
- To develop a platform where a user can search,view, explore and buy products at the same time a seller can post, edit, and sell the products.
- Search functionality that searches products more efficiently than standard regex search..
- Recommendation functionality that recommends products to the user based on the user's purchase history.
- Users can view product details, read reviews of the products and also can write reviews after purchasing the products.

1.2 Why an E-commerce Cloth Store ?

There are many e-commerce online Shopping websites out there offering a large number of functionalities and various shopping experiences. Many sellers build their own website as many software provide GUI for building the websites. But our scope doesn't cover those sellers as they don't have that much traffic on their sites. Other than that there are many Online Marketplaces out there where sellers can sell their products online without worrying about maintaining or building a website for that. Some of the well known Online marketplaces in India are *Amazon, Flipkart, Snapdeal* etc.

Problems:

- People nowadays lean towards online shopping instead of buying in stores.
- Not all sellers can afford multiple shops in various areas. so they need some platform to extend their business.
- Most of the sellers don't know how to build or maintain the websites. They have to use the available online platforms for the business to grow.
- Most of the online marketplaces are built on Monolithic Architecture which are not suitable for horizontal scaling. As these Marketplaces must have a flexibility to scale when needed like on holidays or on Black friday etc. Also they need to be fault tolerant to give the users a smooth and amazing shopping experience.
- Search engines in most of the traditional shopping websites use regex search /exact match in which matched products are directly shown to users without ranking them by relevance score.

Solutions:

- The main intention of this project is to allow the developer/owner of the Marketplace to Scale the individual functionality independently as per need.
- The user will get searched products based on the relevance score. Also get recommendations based on the purchase history and similar products that are similar to the product they are viewing , for a better shopping experience.

- An Interactive Interface by which a user (buyer/seller) can explore various functionalities of the application easily.
- Use Microservice Architecture to make the application scalable. One or more functionality served by any one of the microservice that is independent from other microservices so one can scale out any microservice without affecting other ones.
- Updates or any fault in one microservice won't affect other microservices so the whole application won't have to shut down completely.
- Horizontal scaling is possible in microservice architecture.

1.3 Scope

Based on the knowledge I had and the time period I have been given for the major project, I have limited the project scope to basic e-commerce functionalities. So In this project,

- There is a user side dashboard as well as seller side dashboard. so there are only 2 entities: one is the user and second one is the seller.
- Users can create account save products in wishlist, search product, view product details and reviews, Purchase Product , track the product and finally can write review or give rating to the products.
- Valid User can become a Seller and then a Seller can sell the product, edit his/her product details, manage the orders of his/her products.
- A I have integrated the third party Payment gateway instead of developing one by myself..
- Other than that each microservice runs independently with their own separate database.
- For running event driven architecture I have implemented Apache Kafka as a messaging service.

Chapter 2: System Analysis

2.1 Functional Requirements

Table 2.1 - Functional Requirements

FR-ID	Functional Requirement (Details)
A-001	**Registration/Login**
	● Users can register and create a new account. After that an email verification is mandatory. ● Verified users can login to the account using the email address and the password. ● To access the authorized data, user must be logged In. ● A session must be maintained after successful login. ● Forgot password functionality must be there in case the user forgot the password and wants to reset the password. ● Users can use the existing account to logged In as a seller.
B-001	**Product Search**
	● No need to be authenticated for a user to search for a product. ● Users can search for the product using the product title/name. ● Searching must not be an exact match and also it should show autocomplete suggestions for keywords of length more than 3. ● Search results must be sorted based on the relevance score. ● Searching keywords can be a part of product title, description, features, category, seller company name. ● As a result of the search query a user will see the list of products each containing a minimized image of the

	product, product name, its average rating and its price.
B-002	**Product Details**

	• Any User can see any product's detailed view by clicking on the product.
	• Detailed view must contains following product informations:
	  ○ Name
	  ○ Full Image
	  ○ Price
	  ○ Offer
	  ○ Availability
	  ○ Description
	  ○ Seller Company Name
	  ○ Top 5 Similar products
	  ○ Rating & Reviews

C-001	**Recommendation of Products**
	• If a user has placed more than 1 order then each time a user lends on the Home page, he/she can see the recommended products based on his/her past orders.
D-001	**Shopping Cart**
	• Shopping cart works as a real life shopping cart. • A user can add products and remove products in the shopping cart before purchasing them. • A user can change the quantity of the products in the cart. • Users can add product in the cart by clicking on " **Add to Cart"** button in product detailed view. • Cart Automatically calculates the Total Bill amount and the user can see the total bill amount in the Cart before placing the order. • After placing the order shopping cart will be emptied automatically
E-001	**Place Order**
	• Users must be logged In to place an order. • After confirming the products in cart, users can place orders by pressing the button ***Place Order***. • After that User has to add shipping and billing information and at last the payment to confirm the order. • An order Invoice is sent to the user's email after order confirmation.
	Track the Order

F-001	● After Placing the order, an authenticated user can track the order in the profile section. ● Users can only see the order status and other order information like orderId, products contained in Order , payment Information etc. ● Order status is changed by the seller as he/she process the order.
G-001	**Payment** ● After placing the order and entering the shipping and billing information , User will be redirected to the third party payment gateway. ● Users can pay using Debit/Credit card,wallet,Net banking or UPI. ● After successful payment order is confirmed. ● Refund can be set back to the User after a successful payment in the following two cases : ○ Order canceled by seller / technical issue. ○ User canceled the order before the order Shipped. ● After the order is delivered successfully the order payment is sent to the seller account.
H-001	**Profile Settings** ● User must be Authenticated to access the profile settings. ● User can change the following informations: ○ Full Name ○ Phone Number ○ Email Id ○ Password ● For changing password user first has to write current password EmailId re-verification is needed. ● On changing a password a email notification is sent on the the registered email Address.
I-001	**Recommendation of Products** ● Authenticated users can see the recommended products on one of the sections in the home page. ● This recommendation is based on the purchase history of the user as well as the bookmarked products. ● If a user is new or unauthenticated then this section won't appear in the home page.
J-001	**Similar Products** ● On the Product detail page a viewer can see the similar products in one of the sections of the page.

	● Similarity is based on how the product has similar content in title, description, features and category. ● No need for authentication to view the similar products.
K-001	**Rating and Reviews**
	● Any Unauthenticated user can see the product reviews and ratings of any product in product details page. ● Only an Authenticated user can rate the product and write a review after buying that product.
L-001	**Order Related Services**
	● After the order is placed an authorized user can see the order details in the profile section. ● After the order is confirmed an invoice is sent to the registered email id. ● Invoice will consist of the digital bill with the payment info and the payment Status. ● If any failure occurs and order canceled than notification set via email to the user with refund details. ● As the order is confirmed the order related product quantity automatically decreased from the stock. ● If stock is updated by the order and then order canceled then stock updates are undo automatically.
M-001	**Bookmark the product**
	● The authenticated user can bookmark the product by clicking on the Add ***to wishlist*** button in product detail view. ● The user can view or remove the bookmark products..

2.2 Non-Functional Requirements

Table 2.2 - Non- Functional Requirements

NFR-ID	Non Functional Requirement (Details)
NA-001	**Security**
	• Jwt based secure authentication mechanism • Authorization check before providing sensitive resources. • Sensitive Data like passwords must be stored in hash form that un- interruptible
NB-001	**Scalability**
	• The loosely coupled nature of services in microservice architecture allows to run multiple instances of the services which are called the most. • In this application the most called upon services will be authentication and browsing the products. Hence, we can scale these services individually whenever needed.
NC-001	**Availability & Performance**
	• Availability and Performance of an application have a substantial impact on the user experience. • Our application will have an all-service encapsulating API-Gateway which will provide efficient **load balancing** for the network traffic. • Avoiding the services to be overwhelmed with traffic we can ensure that the requests are fulfilled in low time and the services remain available round the clock.

2.3 Use Case Diagram

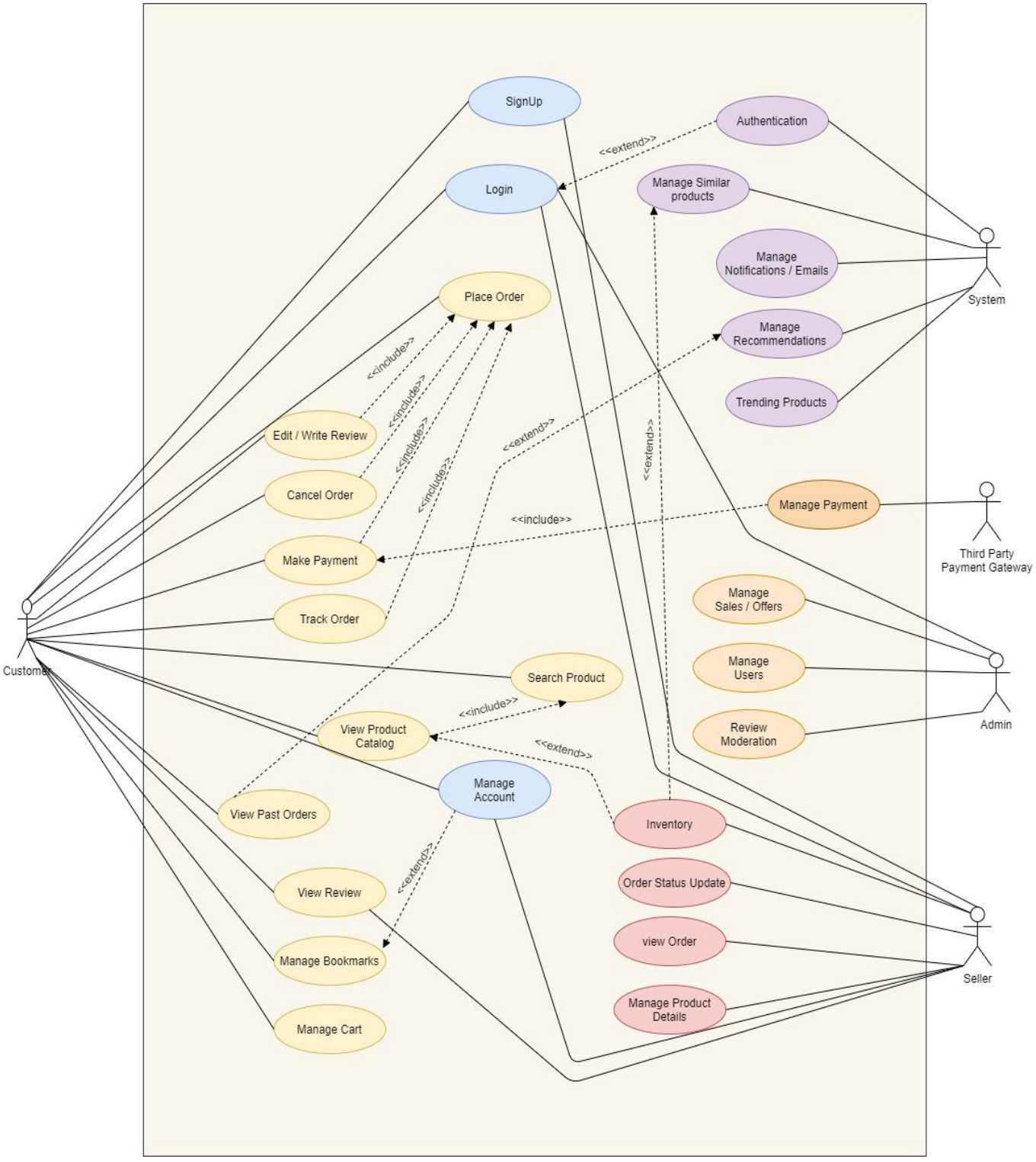

Fig 1 Use Case Diagram of Online Cloth Store

Chapter 3: Architectural Design

3.1 Service Architecture

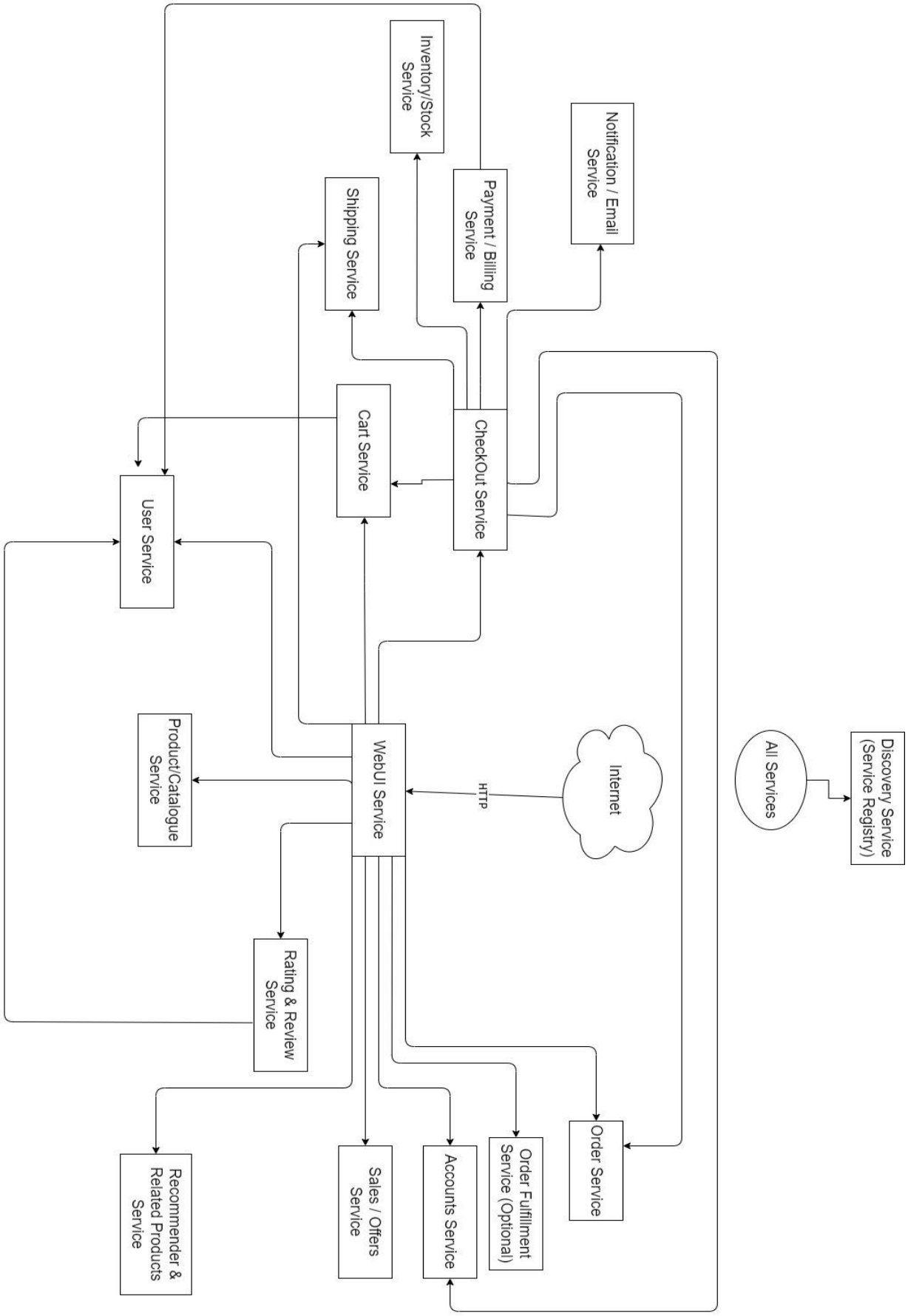

Fig 2 - Service Architecture Diagram of Online Cloth Store

3.2 Description of Architecture

As one can see in Fig 2 all the services are accessed by the end user through Web UI service over the Internet using HTTPS protocol. So the modularity of the application is hidden behind the WebUI service and User gets the experience of the whole application.

Each service has its own database as well as bounded context. Here Arrow represents **internal api calls** between services. One service can access another service through internal api calls or by publishing events in the message queue. For api calls one service needs the hostname and port number of the instance of the service it wants to access. As one service can have multiple instances running on different port numbers , we can't hard code the service urls. As a solution to this there is a **Discovery Service** that i**s** developed, to keep track of the services and its instance currently running. It provides a functionality to access any service just by using its application Name .

So In short it works as a service registry where each service can register its current running instances inside of its application name. Other services can access this service by using the application name and this discovery service will transfer the request to any one of the running instances. More about this service is discussed in further sections.

3.3 Project Implementation (Microservice Approach)

Following are the services made by me for this Online cloth store project.

3.3.1 Product Catalogue Service

This Service is used most of the time and triggered when the user first visits the site home page. As any user can see the categories and product lists on the home page that are provided by this service. This service also handles the search functionality so whenever a user types anything in the search bar, this service will provide autocomplete suggestions to the user. After writing the search query when the user hits the search button this service will fetch the relevant products and return them to the user ranked by the

relevance score. Other than that when the user opens any product detail page the product information is provided by this service. It also provides end points to create, edit and delete the products. It also keeps the track of the stock information of the products. It listens to some events that automatically update the stocks based on the order placed and canceled.

Note :- Each response is returned in JSON form and authentication is done at gateway level wherever it's needed. So individual service has to check only authorization for the request wherever needed.

Table 3.1 Product Catalog Service End Points

Request Information	Response Information
`GET` `/products`	It will return all the products without any filter. Any user can access this endpoint without logged in.
	Response contains product information like, ● id ● title ● price ● average rating
`GET` `/product?id={product_id}`	Returns specific products. When the user open product detail page this request is generated where *product_id* is provided as a path variable to fetch the product details.
	Response contains all product info, which contains, ● id ● title ● Image uri ● price ● average rating ● Seller id ● description ● category ● features like color, material, size etc

GET /category	It will return all the root categories(categories without parent category). Any user can access this endpoint without logged in. When a user lands on the home page this request is generated.
	Response contains category information like, ● id ● name
GET /category/{category_id}	It will return all children of the category with *category_id.* we can find the children of the category(with id = category_id) simply by searching categories that have parentId equals category_id.
	Response contains list of category information like, ● id ● name
GET /products?category={name}	It will return all the products that have category equals to *name* .
	Response contains product information like, ● id ● title ● price ● average rating
POST /products/create	This request is used for creating new products. Authentication is needed for a user to create a product and also a user must be a seller otherwise this request returns an error message in response. Input validation is done by the server side. If a product is created successfully then a success message is sent as a response. Seller id and product id is set automatically on creation of the topic.
	Request contains product information like, ins,

	• title • Image uri • price • description • category • features like color, material, size etc
POST /category/create	This end point is used for creating new categories. Any Authenticated Seller can create a new category and can set its parent category from the existing categories available. Only an Admin can create a root category(category without parent category).
POST /update/product/	When an authenticated seller change the product detail this request is generated. If the product is updated then a success message is sent back in response. Seller can change every information except the productID. Requests must contain the productId.
POST /update/category/	Only an admin can change the category.
DELETE /product/{productId}	This request will delete the product with the id equals *productId.* Only authenticated seller who created the product can delete the product.
DELETE /category/{categoryId}	Only an Admin can delete the category using the categoryId provided in uri.
GET /search/{keyword}	This end point is used for giving autocomplete suggestions when the user typing in the search bar. Response contains top 10 product titles that match with the search *keyword*.
GET /fulltext/search/{keyword}	This end point is used for giving the final search result when the user hit the search button.

	Response contains product list relevant to the search query.

Implementation Details

In product search functionality I have used a concept named inverted indexing for retrieving the documents efficiently. For real time autocomplete suggestions during product search I have used jQuery for the retrieving product titles, that are relevant to keywords typed in the search bar.

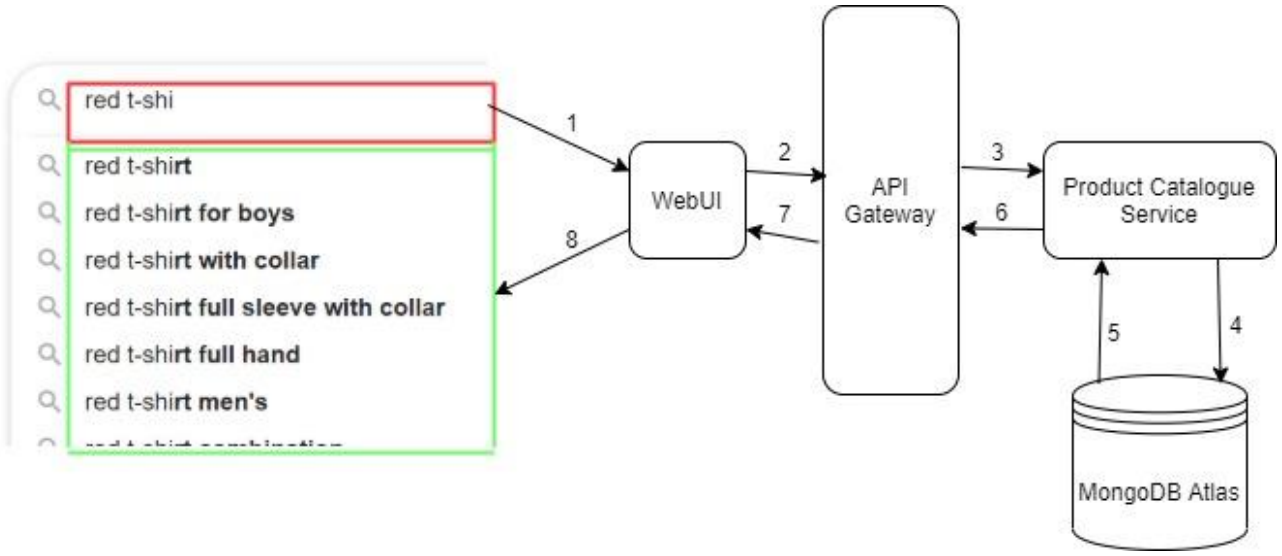

Fig 3.1 - Keyword Suggestions using jQuery

In Above figure ,

1. User type keywords (of length more than 3) in a search bar.
2. WebUI uses jQuery to send a request containing the keywords typed in searchbar, to apigateway.
3. Api Gateway routes the request to one of the running product catalogue service instances.
4. Service builds a query for the database using the keywords and prebuilt indexes in the request. and send it to Atlas.
5. MonogoDB runs the query and returns the retrieved data to the product catalogue service.
6. Service bind the retrieved data in JSON and send it to apigateway.
7. Api gateway sends the data to its appropriate origin.
8. WebUI shows the response data as suggestions to the user.

After typing the keywords in the search bar when a user clicks on the search button, a request is sent to this service, then the service does a search known as ***Full text search*** . A full text search is more efficient than traditional regex search which provides a user better

search experience. Full text search uses the **Inverted Indexing** concept for document retrieval.

"Efficiency of the full text search depends on how a document is Indexed."

Today most of the search engines use elastic search for full text search. Elastic search uses ***Lucene Library*** for indexing the documents. After some research I found that MongoDB also uses the Lucene library for indexing the document. So for the ease of implementation I have decided to use the Cloud database provided by the mongoDb named ***MongoDB Atlas*** .

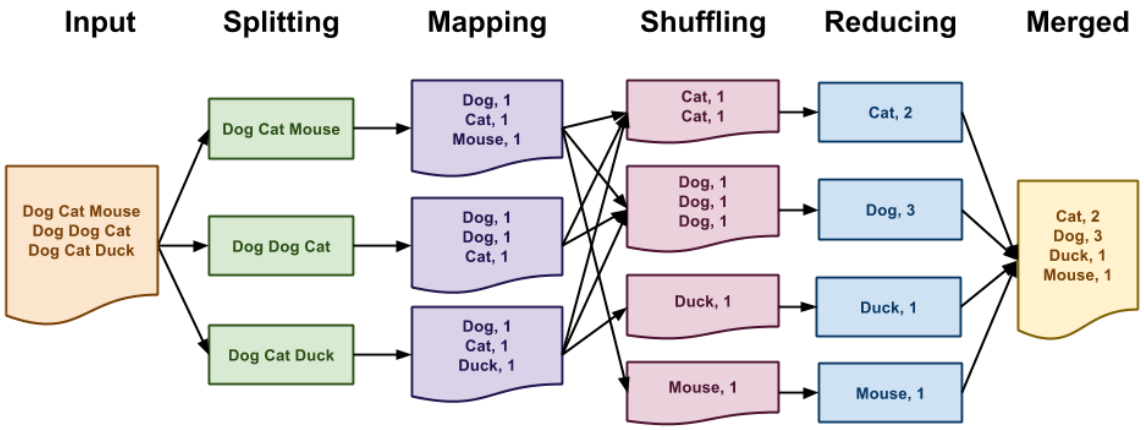

Fig 3.2 - Example of Inverted Indexing

In simple terms, an inverted index stores the documents in mapping with words where words are the key and the value is the list of documents containing that word. More complex indexes also store the frequency and position of a word in a document. The simple one where the only reference to the document stored is known as ***Record-Level-Inverted Index*** and the complex one where the word frequency, position and the reference to the document stored is known as ***Word-Level_Inverted Index.*** Record level inverted Indexing is fast and consumes less space where the Word - level inverted indexing is little bit slow and consumes more space compared to Record level but the word-level inverted index is more accurate or effective than the record level Inverted Indexing.

As **search suggestions** need low latency over accuracy I have used **Record Level Inverted Indexing** and for final search which is a **full text search** where accuracy is

crucial , I have used the **Word Level Inverted Indexing.** Also for saving the space and achieving low latency I only indexed the important informations out of all informations in product, which are mentioned below,

- Autofill Search Suggestions - only Title of the products are indexed.
- Full text Search - Title, description, seller Company name, category, color and material of the products are indexed.
- Also for Full text search I haven't stored the position of the Word to decrease the index size.

In full text search the retrieved documents are also sort based on the relevance score that generated by the MongoDb Atlas based on how similar is the document to the search query. Every new document added in the database is Automatically indexed at the time of creation. Below are the **Index informations** of both indexes.,

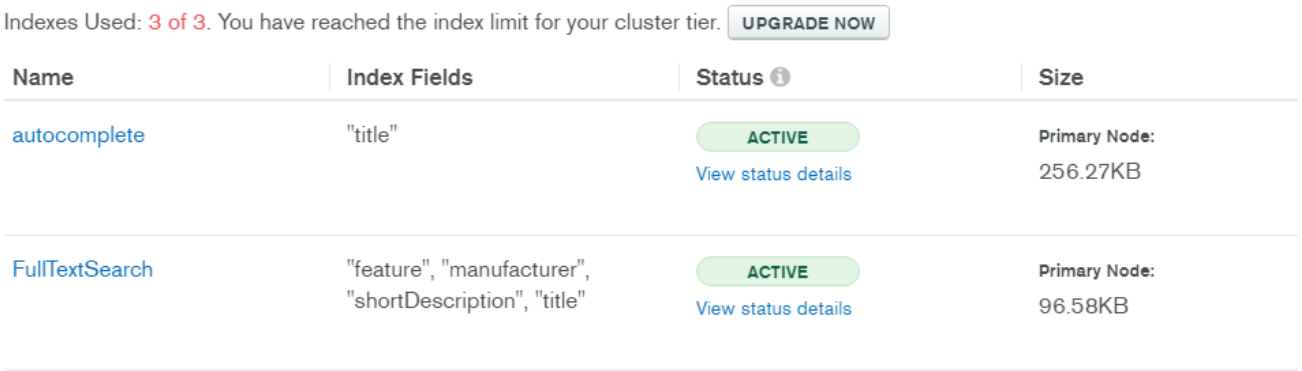

Fig 3.3 - Index informations on MongoDB Atlas Dashboard

Tools & Technology used

- **NodeJs** to develop service
- **MongoDb** as a DataBase
- **MongoDb Atlas** for indexing and data management
- **eureka-js-client** for register service to Eureka service(Discovery Service)
- **kafka-node** for the use of async messaging service.

3.3.2 Email Service

Email service is used for sending emails to customers. Emails are used for informing the user about the result of async events like order confirmed, order canceled, payment failed etc. Emails are also used for notifying the users like order shipped, order delhivery, password change, offers available, product arrived etc. This Service is mainly used by the service through **Asynchronous communication** which is through Kafka Messaging queue. To use the service we need to provide the following details,

- Sender
- Recipient
- Subject
- Email Text
- Attachment (Optional)

For the Testing purpose This service is currently using Gmail SMTP services to send the email, Which has a daily limit of sending emails of 2000 Emails. Other than that there are end points set for the admin to use the service directly instead of publishing in the queue. No one other than the admin and other services can access this service through gateway. So this Service provides 3 Types of email that we can send. These three types are as follows,

- Simple Email (Single Recipient, No attachment)
- Email with Attachment (Single Recipient, with Attachment)
- Multicast Email (Multiple Recipient)

Things to notice that in multicast Email the same email is sent to multiple recipients individually instead of sending them in group by adding them into CC section. Also if email Text contains HTML text then it'll automatically render when the user opens the email. Here are some examples of email sent by this service.

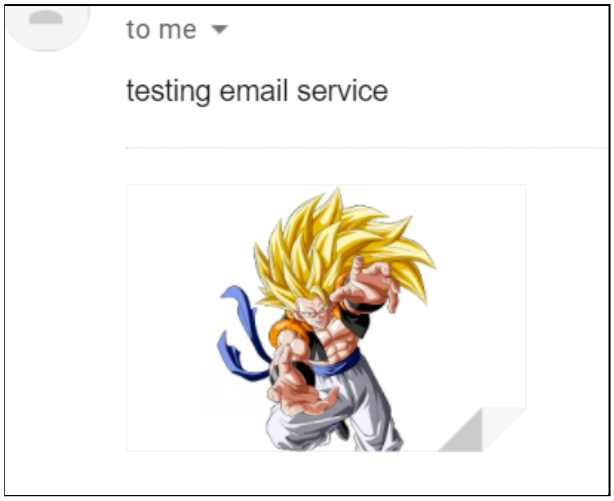

Fig 3.4.1 Email with Attachment

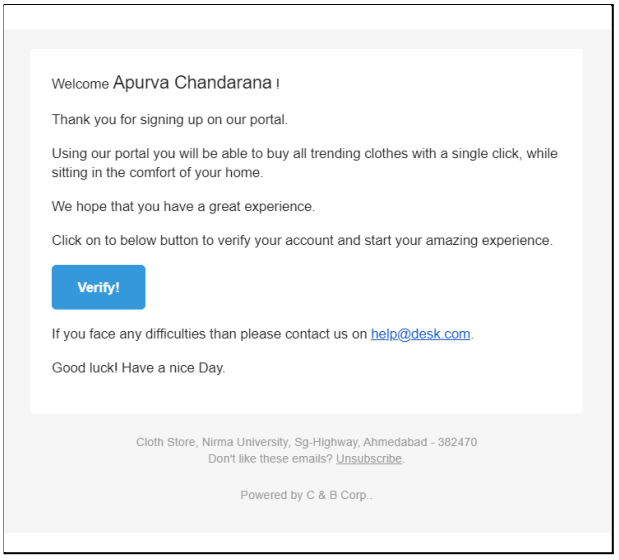

Fig 3.4.2 verification Email with html text rendered automatically

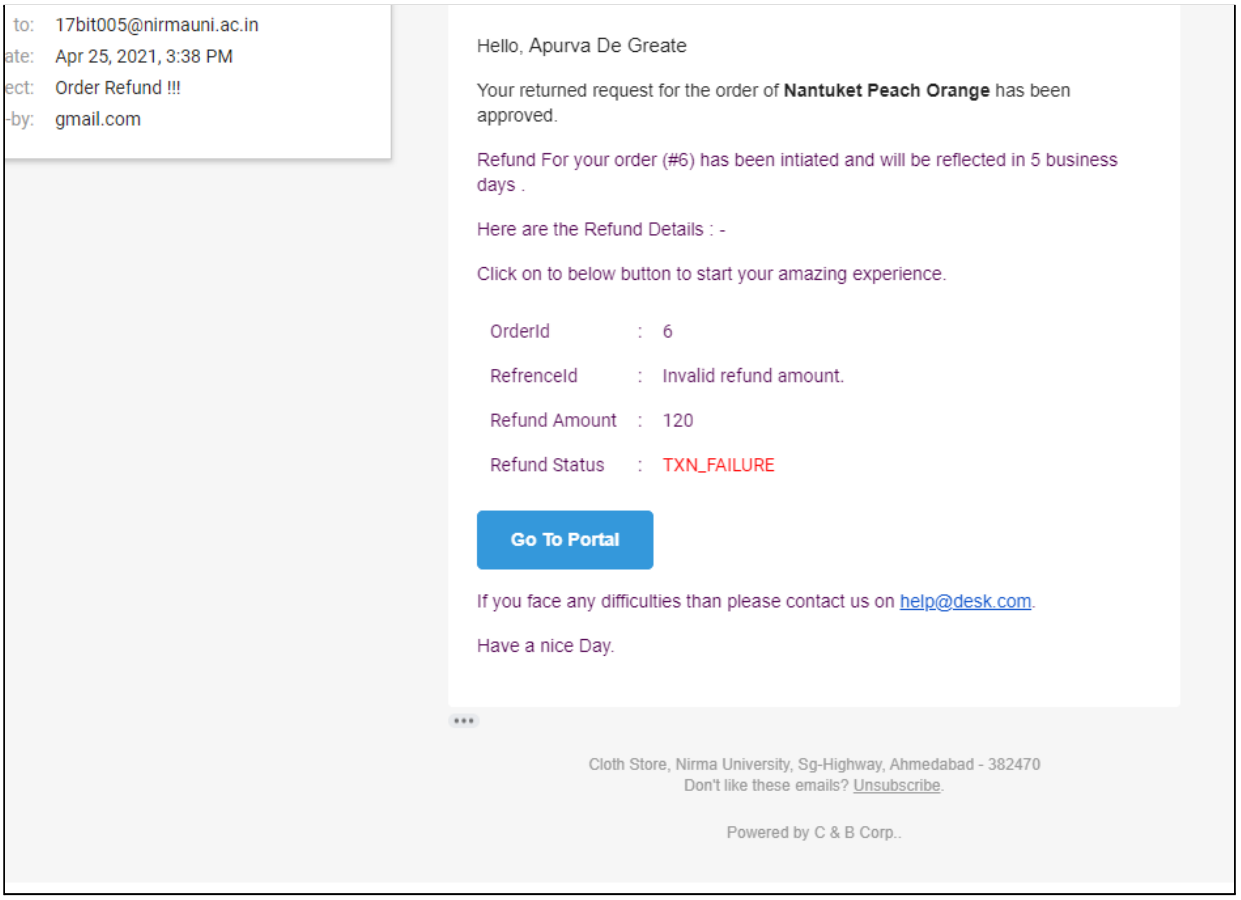

Fig 3.4.3 Email to notify the failure of refund event

Tools and Technology used

- **Spring Boot -** Service build on the spring boot

- **Spring-Kafka** - to use an async messaging queue to listen to events.
- **Gmail SMTP -** to send emails and keep track of emails .(Only for Testing)

3.3.3 Order Fulfillment Service

Order Fulfillment service is related to post-order activities. These activities includes the followings,

- Check payment Status of the order and update status according to it.
- Check the stocks and update order status according to it.
- Generate Order Invoice if order is confirmed.
- Generate cancelation Notification if fail to confirm the order.
- Initiate the refund for the Failed order if payment has been succeeded for that order.
- Generate Refund details for the user on the cancelation of order.

This Service does most of the work by using asynchronous messaging and listening to the events published by the other services like payment service for payment related events, order service for order related events etc.

Here in fig 3.5 there is the flow of events how order fulfillment service manages the order and also how it manages the post order services like cancel order or returned order. As we can see, the first Action a user can make is to place an order and make a payment. As payment may take time to proceed I have set this process asynchronous, which means the user won't have to wait for the end of all this process before continuing exploring the site. So the after user makes payment order service creates order in pending state and payment service proceeds the payment and returns the status of the payment to the callback function.

This callback function publishes an event with payment status to order fulfilment service where it checks the payment status and if its failure then as per diagram it publishes

2 events, one for order service to cancel the order and second one for email service to notify the user about cancellation of order.

Now if Payment is successful then order fulfilment service check for the stock by calling Inventory service. If Stocks are available then reserve the stocks and send invoice through email service and if stocks aren't available then publish an event to cancel order for order service and refund event for payment service to initiate refund. At the end send cancelation notification with refund details through email service.

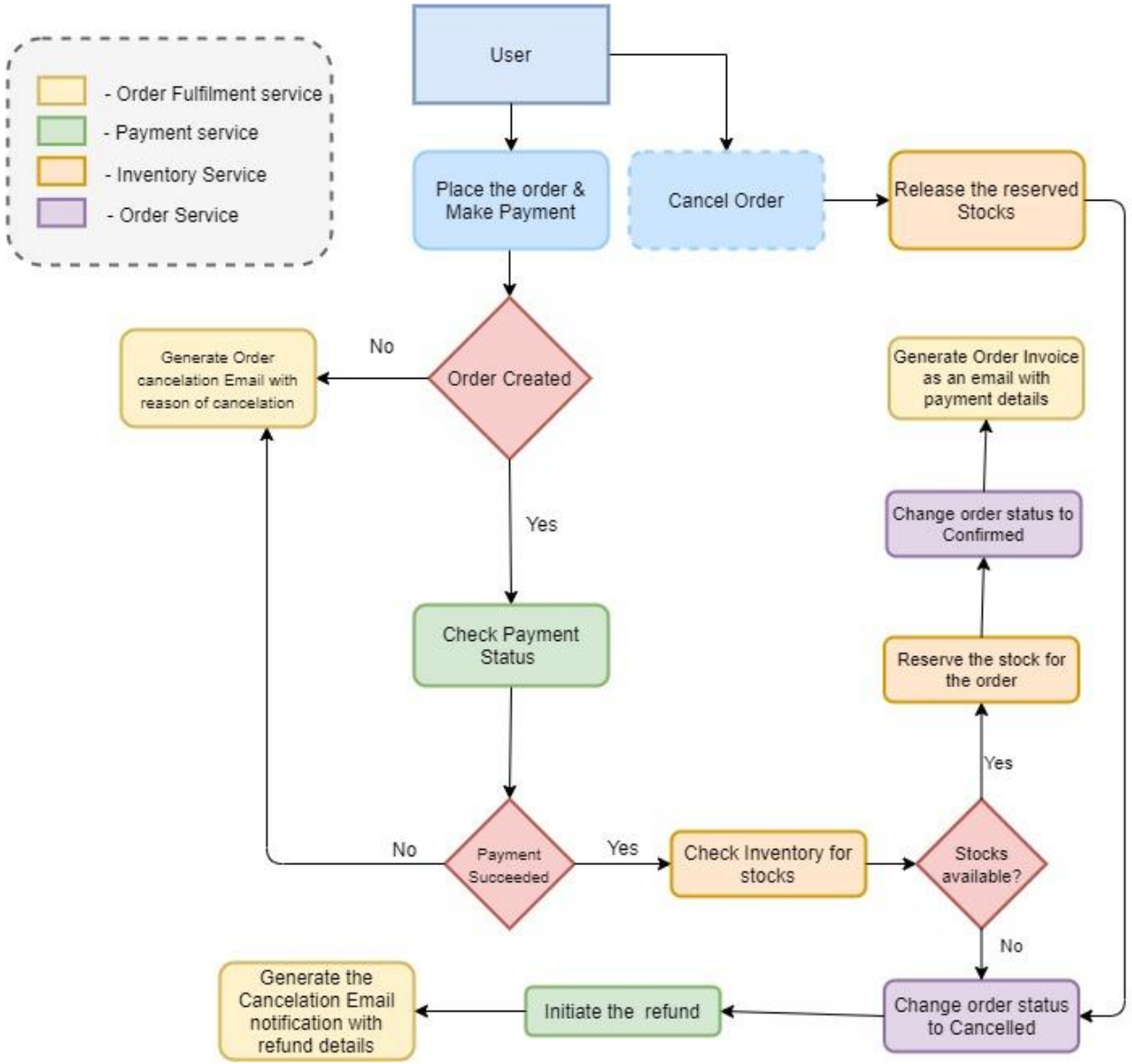

Fig 3.5 - Flow diagram of order fulfilment service managing order

Second Action is one can cancel an order by oneself in that case order fulfillment service publish release stock event for inventory service cancel order event for order service, refund event for payment service and finally send notification with refund details via email service to user.

Tools & Technology used

- **NodeJs -** for building service.
- **kafka-node -** for creating a kafka publisher and listener.
- **eureka-client-js -** for registering with eureka server.
- **node-fetch -** for making api calls.

3.3.4 Rating & Review Service

This service is associated with the rating and reviews of the products. As nowadays ratings and reviews are more promising than product details, photos and company name. Each product can have multiple reviews and ratings. But to give a review a user must be logged in and also must have purchased that product in the past. This is required otherwise anyone can manipulate the reviews of any product. other than that it's not mandatory to give review after the purchase of the product. Users can give only ratings instead of writing a review. But to write a review User must rate the product first. Also for the ease of data storing, users can only give review in text form. Users can give ratings between 1 to 5 where 1 is worst and 5 is best.

After submitting the review user can edit it afterwards and can delete the review whenever needed. Users can review his/her reviews under the past orders section. Any user can see the reviews of the product on the bottom of the product detail page. Each review contains a reviewer name, rating to the product given by that user and review text. A seller can’t edit reviews on his/her product page.

Other than that avg rating is maintained for each product that is calculated by taking the sum of all the ratings available in the database for that product divided by total

number of review records available for that product. Database stores both records (only ratings and resting+review) but when a product detail page makes a request to the service for reviews it only retrieves those records that contain rating+ review. Also it sends an average rating among the list of reviews.

here is an example of review and rating in database,

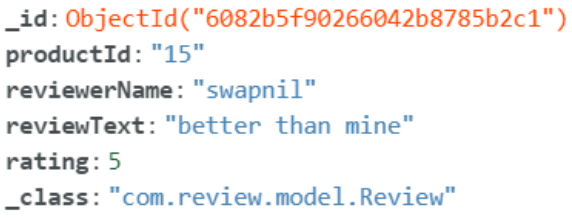

Fig 3.6.1 - Rating + review record

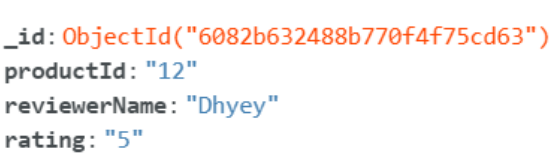

Fig 3.6.2 - Only Rating record

Tools & Technology used

- **Spring Boot-** for building service.
- **cloud.spring.kafka-** for creating a kafka publisher and listener.
- **cloud.spring.EurekaClient -** for registering with eureka server.
- **MongoDB-** for Storing the rating and review data..

3.3.5 Similar Product

Similar product service is used to retrieve some products that have highest similarity with the given products in terms of title, description, category etc. It is similar to the full text search we have done in search functionality. But in search functionality we consider the search keyword and find the products that have similar information in the product details. In finding similar products we have to search the whole product document to find the matching products that have similar information in product details.

To reduce the processing time we have considered 3 things to generate a search query of the given product, which are title, description and category. As these 3 things are the

main when you want to find products having common information. Again I have used **Lucene Library** for indexing and retrieving the products. For retrieving the document it uses the **Inverted Indexing** of the document and after that it uses **Cosine Similarity** to generate the relevance score. Based on the score, retrieved documents are sorted from higher score to lower. Higher the score more similar to the product. After that top 5 results are sent back to WebUI to showcase in the Similar product section of the product detail Page.

A User can see the similar products only in the product detail page. Also on a product detail page users can't see the other products' similar products. Users may get different results each time they visit the page.

Implementation Details

Here are the index details that are used for retrieving the similar products,

```
Similarity_Index :-  {
                       "analyzer": "lucene.english",
                       "searchAnalyzer": "lucene.english",
                       "mappings": {
                         "dynamic": false,
                         "fields": {
                           "title": {
                             "analyzer": "lucene.english",
                             "ignoreAbove": 255,
                             "indexOptions": "freqs",
                             "norms": "omit",
                             "searchAnalyzer": "lucene.english",
                             "type": "string"
                           },
                           "Description": {
                             "analyzer": "lucene.english",
                             "ignoreAbove": 255,
                             "indexOptions": "freqs",
                             "norms": "omit",
                             "searchAnalyzer": "lucene.english",
                             "type": "string"
                           },
                           "Category": {
                             "analyzer": "lucene.keyword",
```

```
            "indexOptions": "docs",
            "searchAnalyzer": "lucene.keyword",
            "store": false,
            "type": "string"
        }
      }
    }
}
```

Here we can see that there are different analyzers mentioned in the index definition. As mentioned in the search functionality, the accuracy of inverted indexing depends on how to index the document, that means it's very crucial to choose how we tokenize the document. Here these analyzers are the one that decides how to fragment the document in words for indexing. Lucen library provides many analyzers like keyboard, simple, standard, whitespace, language specific etc as shown in below image. One can learn more about it in Mongodb atlas documentations.

Analyzer	Description
Standard	Uses the default analyzer for all Atlas Search indexes and queries.
Simple	Divides text into searchable terms wherever it finds a non-letter character.
Whitespace	Divides text into searchable terms wherever it finds a whitespace character.
Language	Provides a set of language-specific text analyzers.
Keyword	Indexes text fields as single terms.

Fig 3.7 - Different in built analyzers

Tools & Technology Used

- **NodeJs -** for building a service
- **MongoDb Atlas -** for database and indexing.

3.3.6 Payment Service

Payment service manages the payment related information and the actions. Considering my knowledge and the time given for the project I have decided to integrate Payment gateway instead of building one by myself. There are many payment gateways available for integrating with your websites. These api reach gateways provide many functionalities so that we can design our own payment gateway without knowing the complexity of building and managing one. I have integrated ***Paytm Gateway*** because of my familiarity with paytm interface.

Paytm gateway provides you many api endpoints from getting payment options to make refunds. Other than that it also provides you a live dashboard where you can see the payment logs and also can manage refunds and all. This service mainly uses the payment gateway for the following purpose,

- Initiate Payment
- Check Payment Status
- Get offers available
- Initiate refund
- Check refund status

For these operations this service makes post api calls with relevant data in the request body to the paytm server. Users only use this service after placing the order and press on the button named **proceed to pay.** After that this service takes the *Total Amount* and *Receipt Id* that are generated by the order service and makes an api call to the paytm server. The whole process is shown below Fig 3.8.

A callback url must be passed in the request body that is called after the payment is processed. We have set a callback function that generates an event of Payment Completion and publishes it on the messaging queue and also stores the transaction id in the database for future reference. Other than that this service also listen to other service's events to perform action according to it like, make refund on order cancellation.

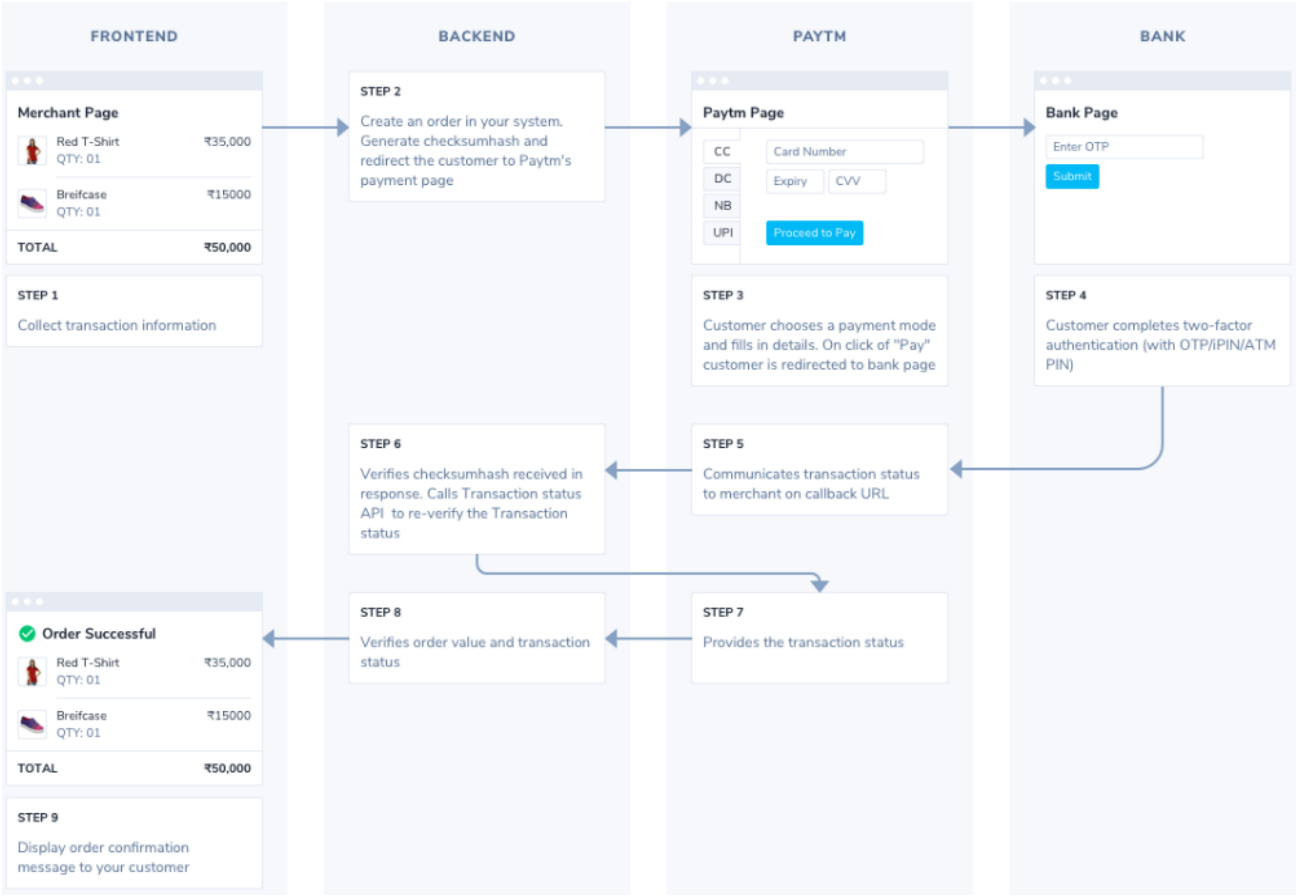

Fig 3.8 - Payment gateway flow diagram

Tools & Technology used

- **NodeJs -** for building service
- **Paytm Payment Gateway -** as a third party payment gateway
- **MongoDb -** to store transaction id for further use
- **kafka-node -** to connect to kafka queue
- **eureka-js-client -** to connect to eureka server

3.3.7 Shipping Service

Shipping service is used for managing shipments. Whenever the seller packs the order and sends it to the shipping company it makes the request to this service to create a shipment with autogenerated shipment Id. After generating this Id this service sends a notification to the user through email service with shipment information. At the same time it also publishes events to change the order status to Shipped. And when the item is delivered successfully this service again changes the order status to delivered. In this project I haven't connected any shipping service and haven't built a shipping interface. So this service is just for demonstration purposes only to help to understand the flow of data and events in Microservice architecture.

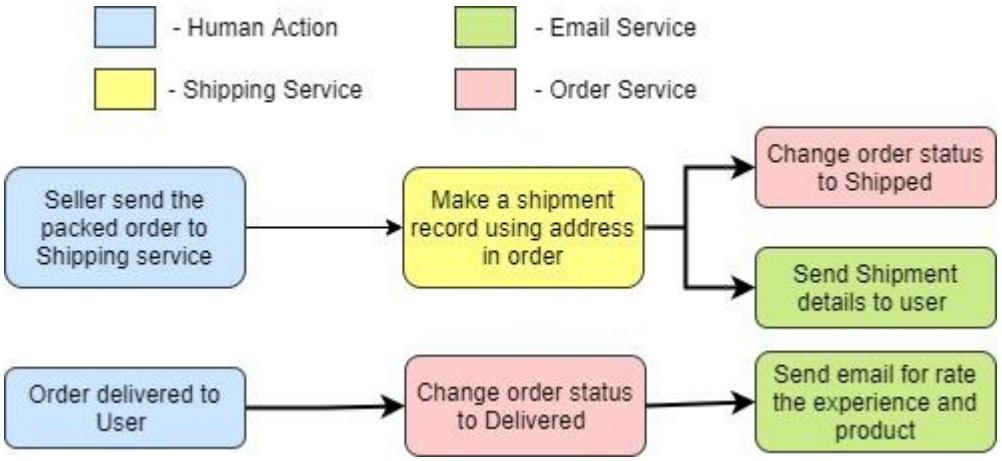

Fig 3.9 - Flow diagram of shipping service

Tools & Technology Used

- **NodeJs -** building service
- **kafka-node -** for kafka pub-sub operations
- **eureka-client-js -** for registering the service to eureka server

3.4 Kafka Implementation (Event Driven Architecture)

Traditionally the standard way of communication between microservices within an application has been through REST APIs. However, as the application scope and number of services increase communication becomes complex. Although communication through APIs has very low latency it requires services to be highly available. To solve this problem we have used a kafka-centric approach in our application. This allows for low latency message exchange between services while providing load balancing and a centralized management of messages. Kafka was first introduced as a message queue and open sourced by LinkedIn in 2011.

Kafka centric approach has the following attributes:

1. Publish and Subscribe to streams/topics like Message queue.
2. Storage system which allows for the messages to be consumed asynchronously. We can even allow new subscribers to consume older messages.
3. Kafka supports Stream Processing which allows complex joining of different streams.

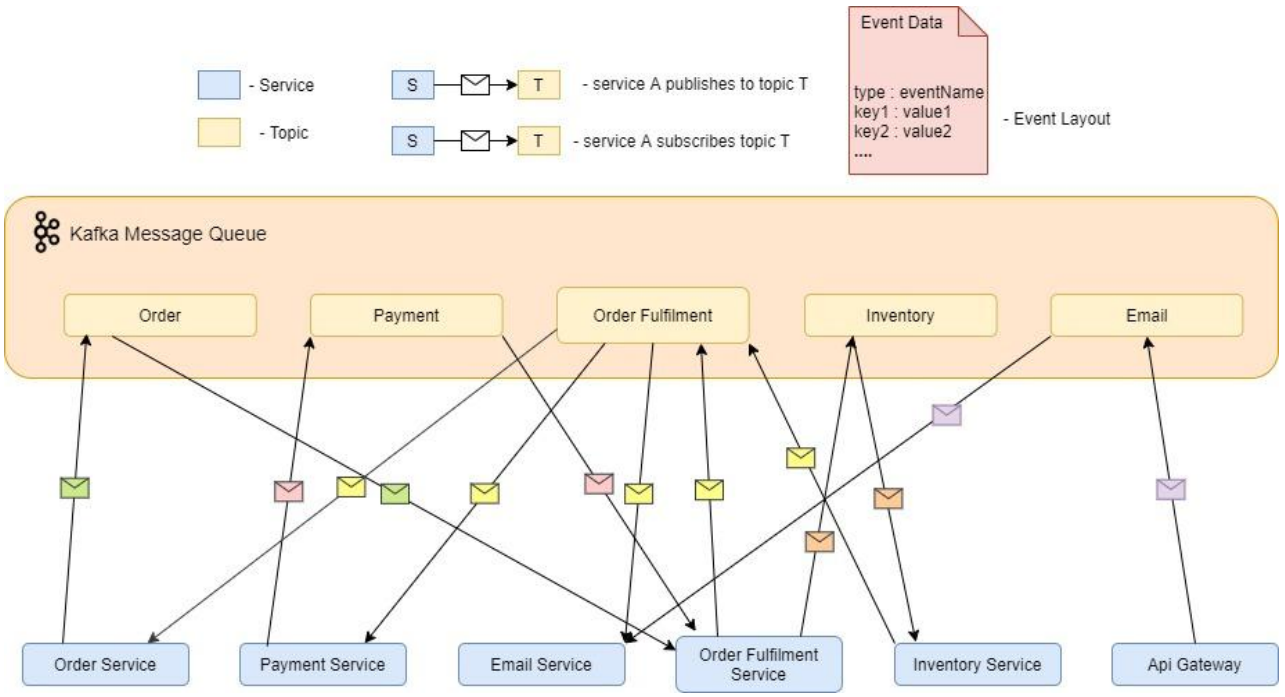

Fig 3.10 - Pub-Sub Model of Online cloth store

Table 3.2 List of Events published in Topics

Topic	Event Name & Description
Order	orderConfirmed
	• This event is generated when the order is confirmed.Order only confirmed after the payment is done. • So this event contains Payment info and order Id .
	orderCanceled
	• This event is generated when the order is canceled. Order may be canceled by failure of payment or the user canceled the order or order canceled due to out of stock of the product. • So this event contains only the order Id of the canceled order..
	orderShipped
	• This event is generated when the order is packed and dispatched to the shipping company. • So this event contains shipping info and order Id .
	orderDelivered
	• This event is generated when the order is delivered to the customer . • So this event contains orderId only.
Order Fulfilment	orderInvoice
	• This event is generated when the order is confirmed and the order invoice is generated. • So this event contains orderInvoice.
	canceledOrderRefundInvoice
	• This event is generated when the order is canceled, and refund is initiated. • So this event contains a refund receipt with order cancelation information .
	returnOrderRefundNotification
	• This event is generated when the order is returned by the user and refund is initiated. • So this event contains a refund receipt and the order return information.
	paymentFailure
	• This event is generated when the payment completed is failed and order canceled due to payment failure.

	• So this event contains order cancelation information and the failed payment information.
Payment	paymentCompleted
	• This event is generated when the payment is done by the payment gateway side. • So this event contains Payment info with the payment status .
	refundInitiated
	• This event is generated when the order is canceled or returned and refund is initiated for that order. • So this event contains refund info with refund status and order id .
Inventory	OutOfStock
	• This event is generated when the stock is less than the required quantity to fulfill the order.. • So this event contains orderId for which the stocks are not available.
Email	simpleEmail
	• This event is generated when any service wants to send an email through email service. • So this event contains email recipient, subject and email text.

Chapter 4: Web User Interface & Final Result

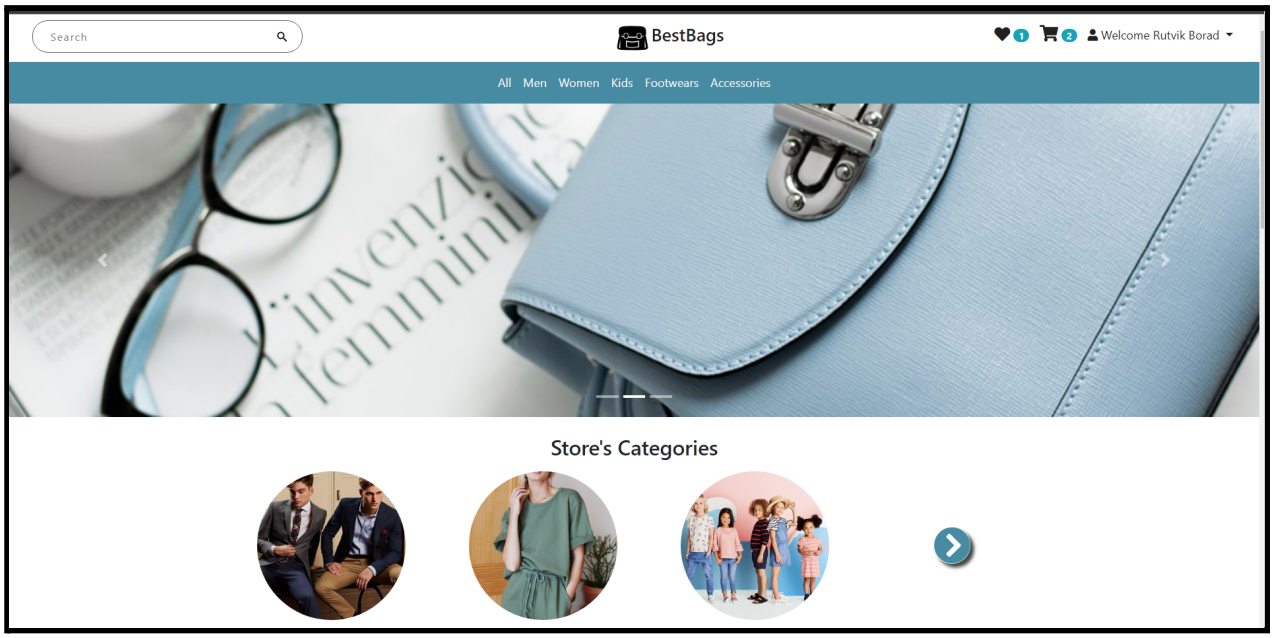

Fig: 4.1 Dashboard

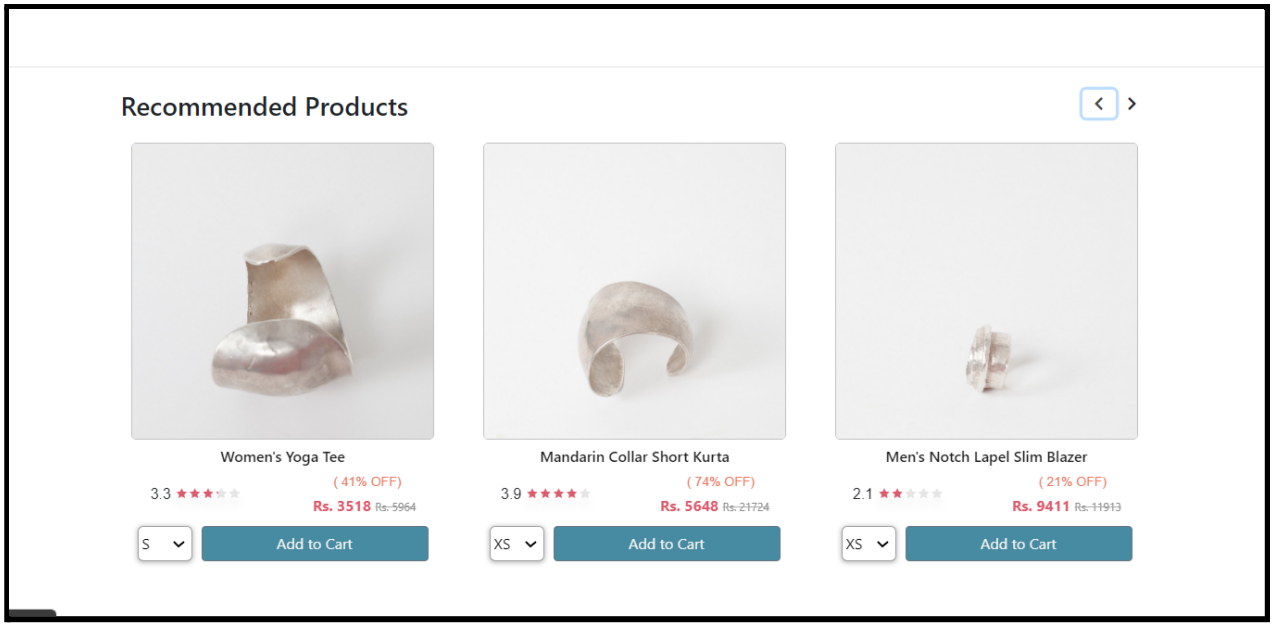

Fig 4.2 - Recommended Products on Dashboard

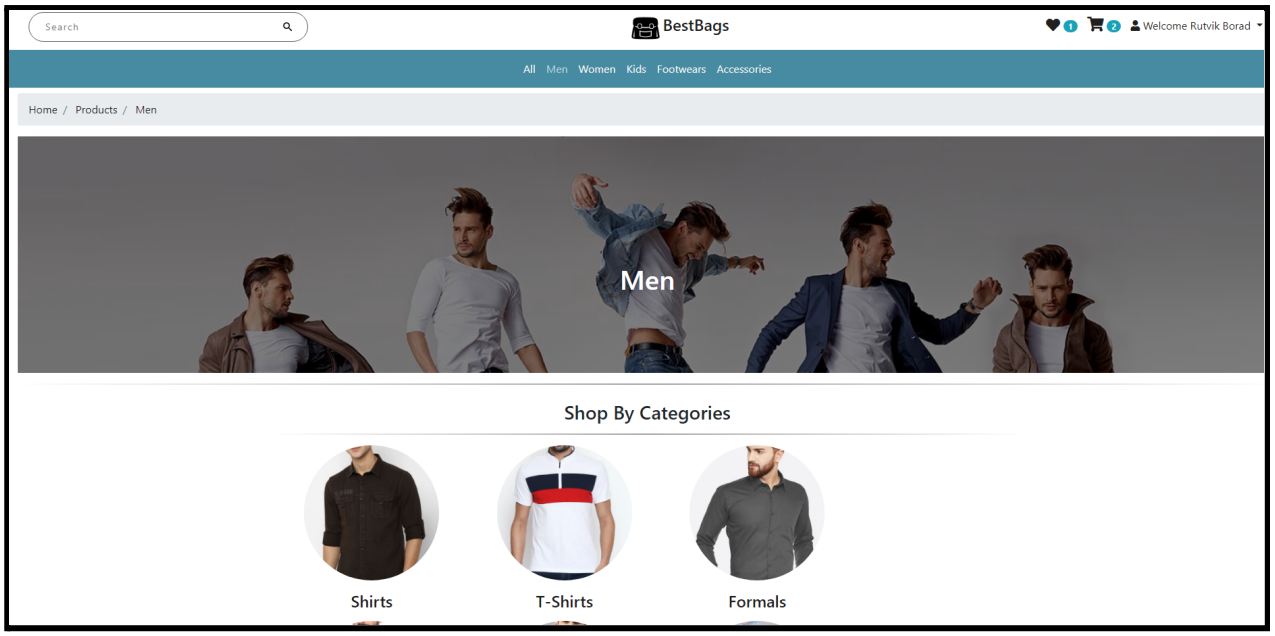

Fig 4.3 Root Category Page

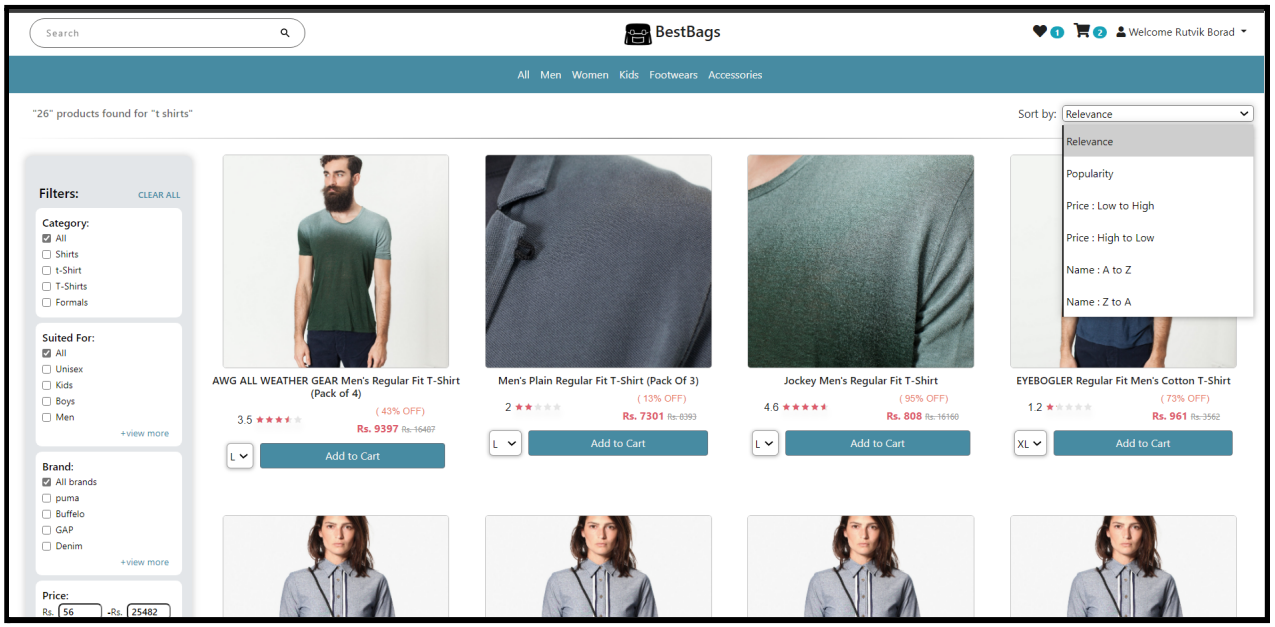

Fig 4.4 Product List Page with Filter

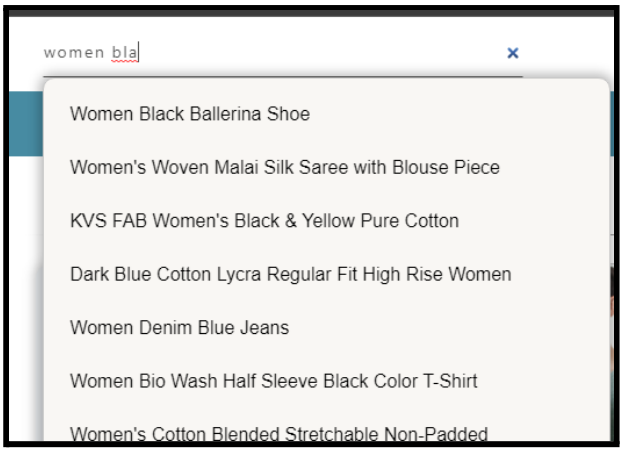

Fig 4.5 Search Suggestions

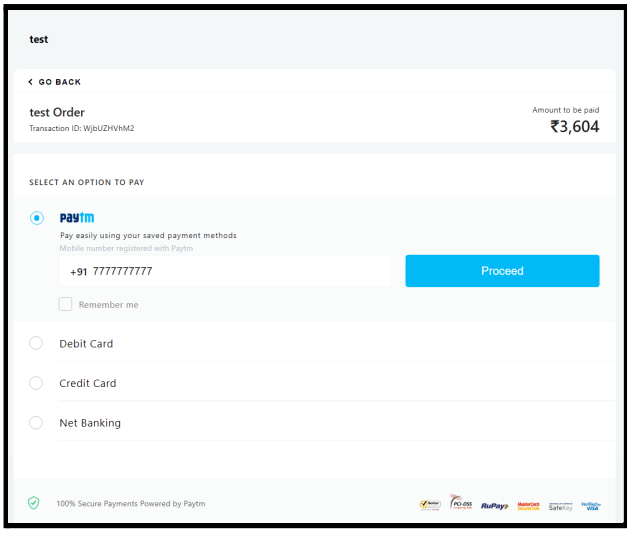

Fig 4.6 Payment Gateway

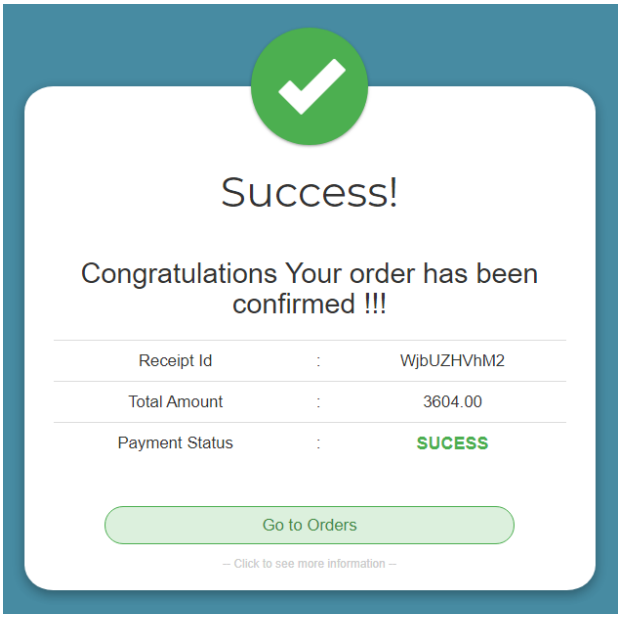

Fig4.7 Payment Status

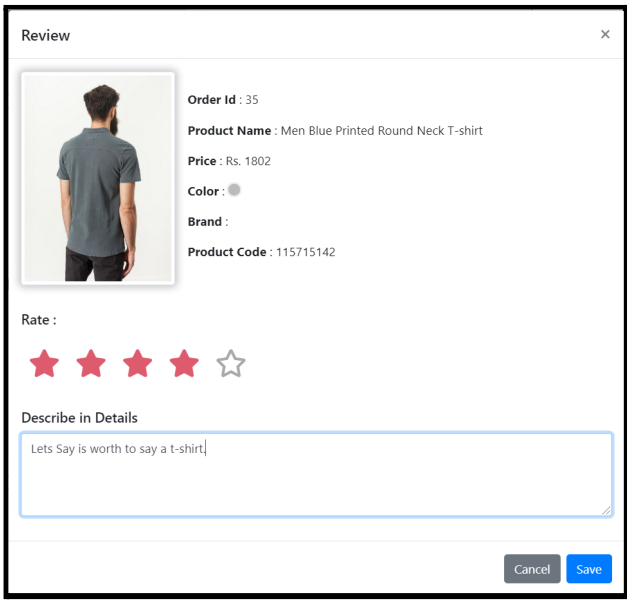

Fig 4.8 Write review window

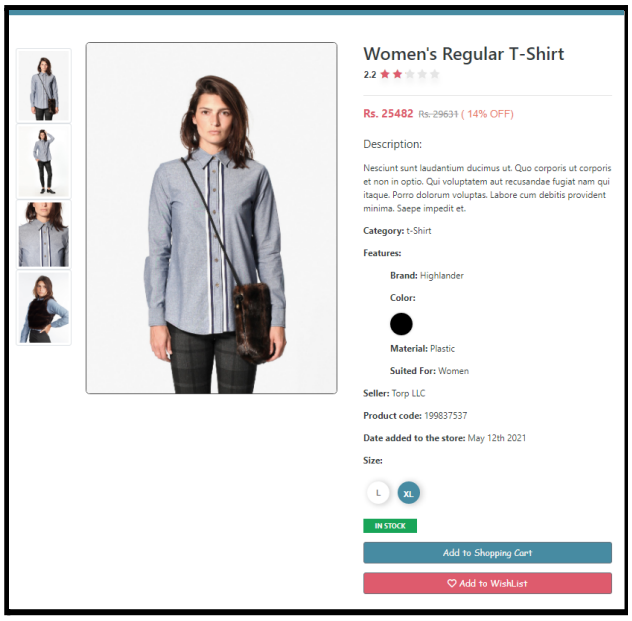

Fig 4.9 Product Detail page

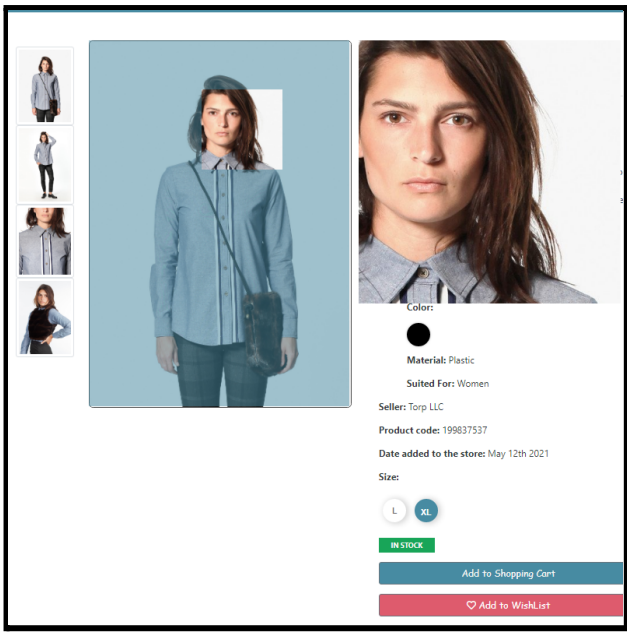

Fig 4.10 Zoom on hover

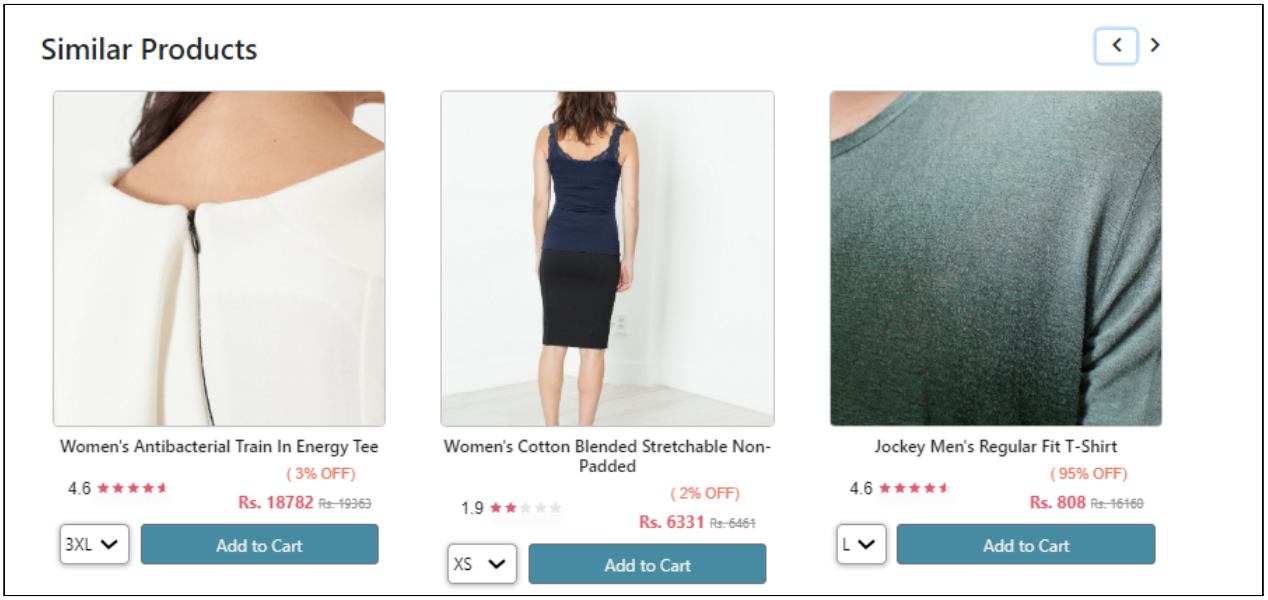

Fig 4.11 Similar Products on Product Page

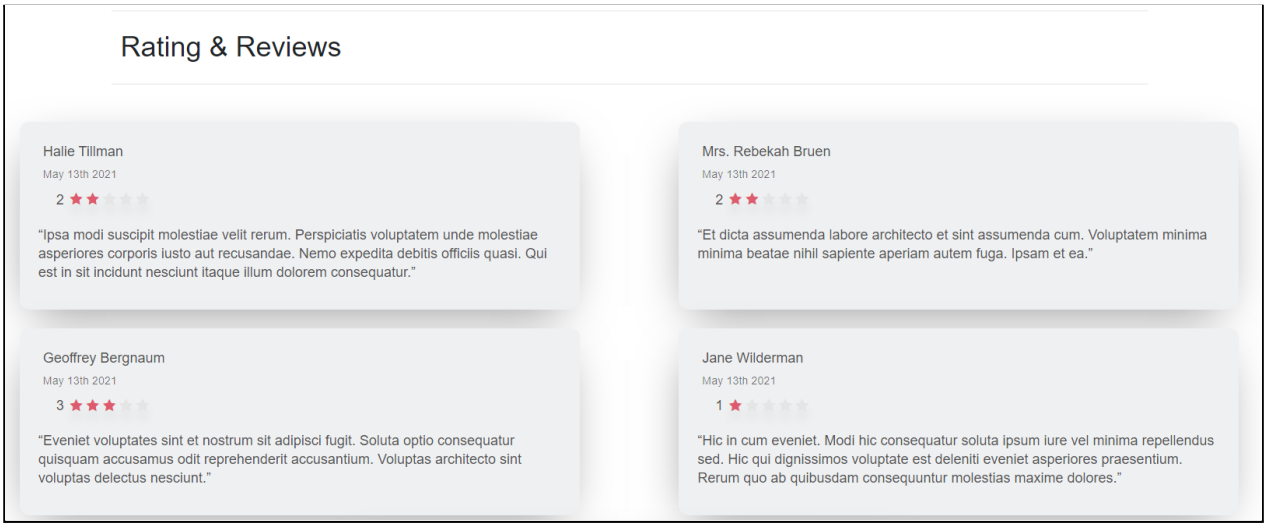

Fig 4.12 Reviews of the Products

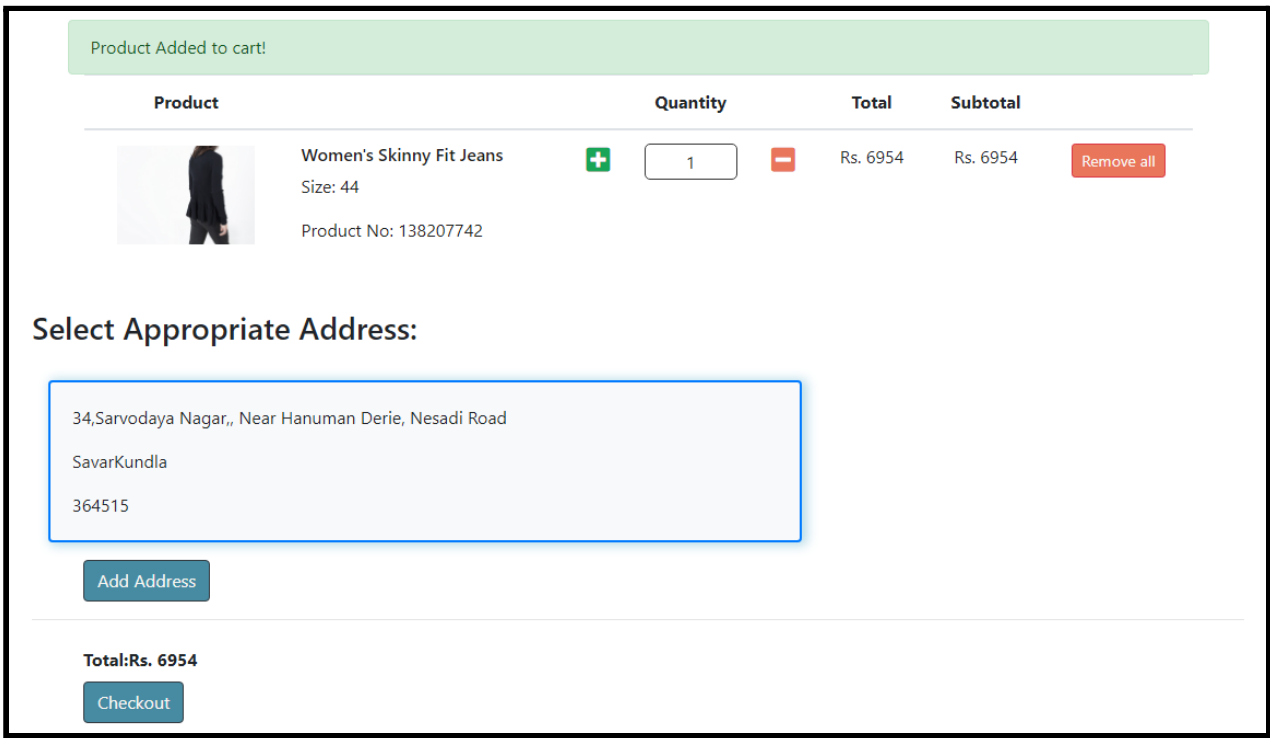

Fig 4.13 Cart Page

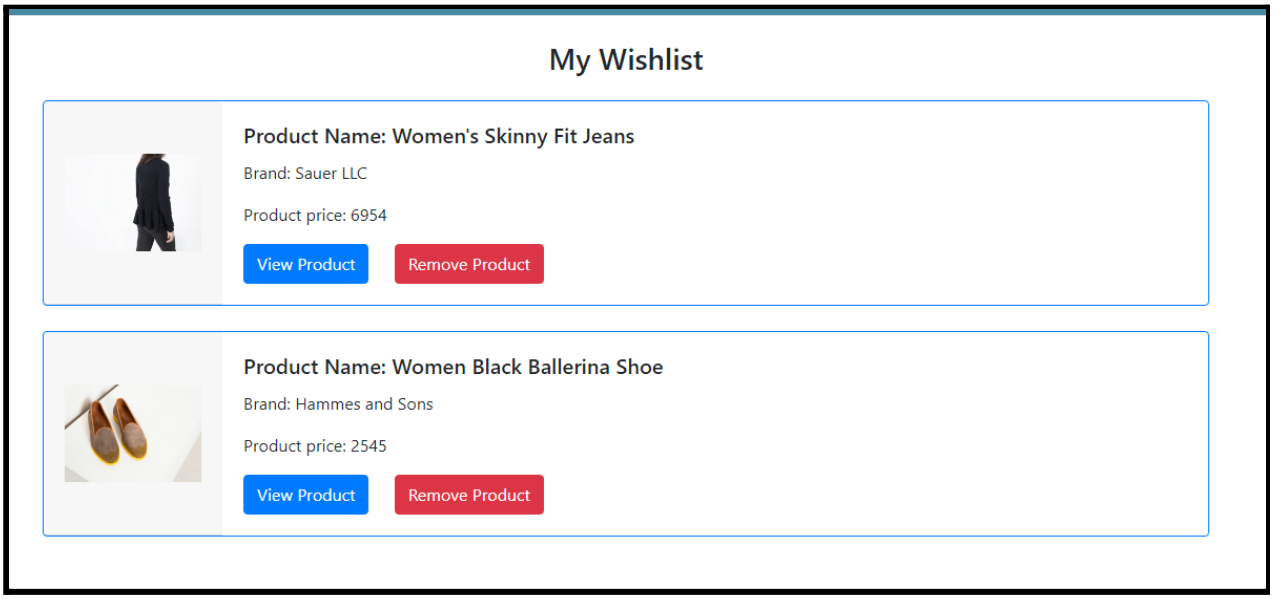

Fig 4.14 Wishlist Page

Chapter 5: Testing

Software testing is a crucial part to process in software/web development. In testing one try to perform some intended action to find any errors. It Provides the quality assurance and also it is like a final review of the previous stages like software design, coding etc.

5.1 Unit Testing

Unit testing highlights the testing of the smallest unit of the software. Unit testing is performed parallely with the coding phase where each service or code is compiled and tested at the time of coding. It tests the individual modules not the whole system before integration.

I have performed unit testing in each and every service individually. After the completion of the coding of the service, every possible input and output is checked for finding any bugs or errors. This action is performed until the service is running correctly without any flows.

Other than valid inputs and outputs, service functionality is also checked after completion of service. Each and every services' functionality is checked individually for the quality assurance using POSTMAN.

In the Product Catalogue Service it has been tested that auto suggestions won't be displayed for words with length less than 3. User hit enter with an empty query then it returned all products. Only a seller can create products. Sellers can not upload more than 4 and less than 1 images while creating the products. Only an admin can add a category. Sellers can't create or choose non existing categories for his/her product.

In Similarity Service it has been tested that all service won't return the product itself that has been provided as an input. Service always returns a maximum of 10 similar products. If the Product didn't exist then it returned nothing.

In Email Service it has been tested that email with html text body rendered on the recipient side. If an error occurred during sending email then it displays the error in the

console. If the same email has to be sent to multiple recipients then the same email is sent individually to all the recipients one by one.

In the Rating Review Service it has been tested that any user can write a review after ordering the product. User has to at least provide a rating to submit the review. Review is reflected on the product page as soon as the user submit the review. users can edit the review after submitting the review and also can delete the review. Users can't rate the product more than 5 and less than 1. Also users can not edit/delete other peoples reviews

In Order Fulfillment Service it has been tested that as the payment confirmation is received it sets a message to order service to change the status of the order based on the status of the payment status. On order confirmation it generates the order invoice for the email service to send to the user. When it receives the order cancellation or returned message then it generates a refund message for the payment service to initiate the refund. On the payment confirmation it sent a message to the inventory service to reduce the stocks of the ordered product and on order cancellation it sent a message to increase the stocks of the ordered products.

In Payment Service it has been tested that when the payment is done it sends the payment status to the order fulfillment service and on receiving the message of the order cancellation it initiates the refund also. And we can see the payment status by using the receipt Id. payment status is reflected on the my order page.

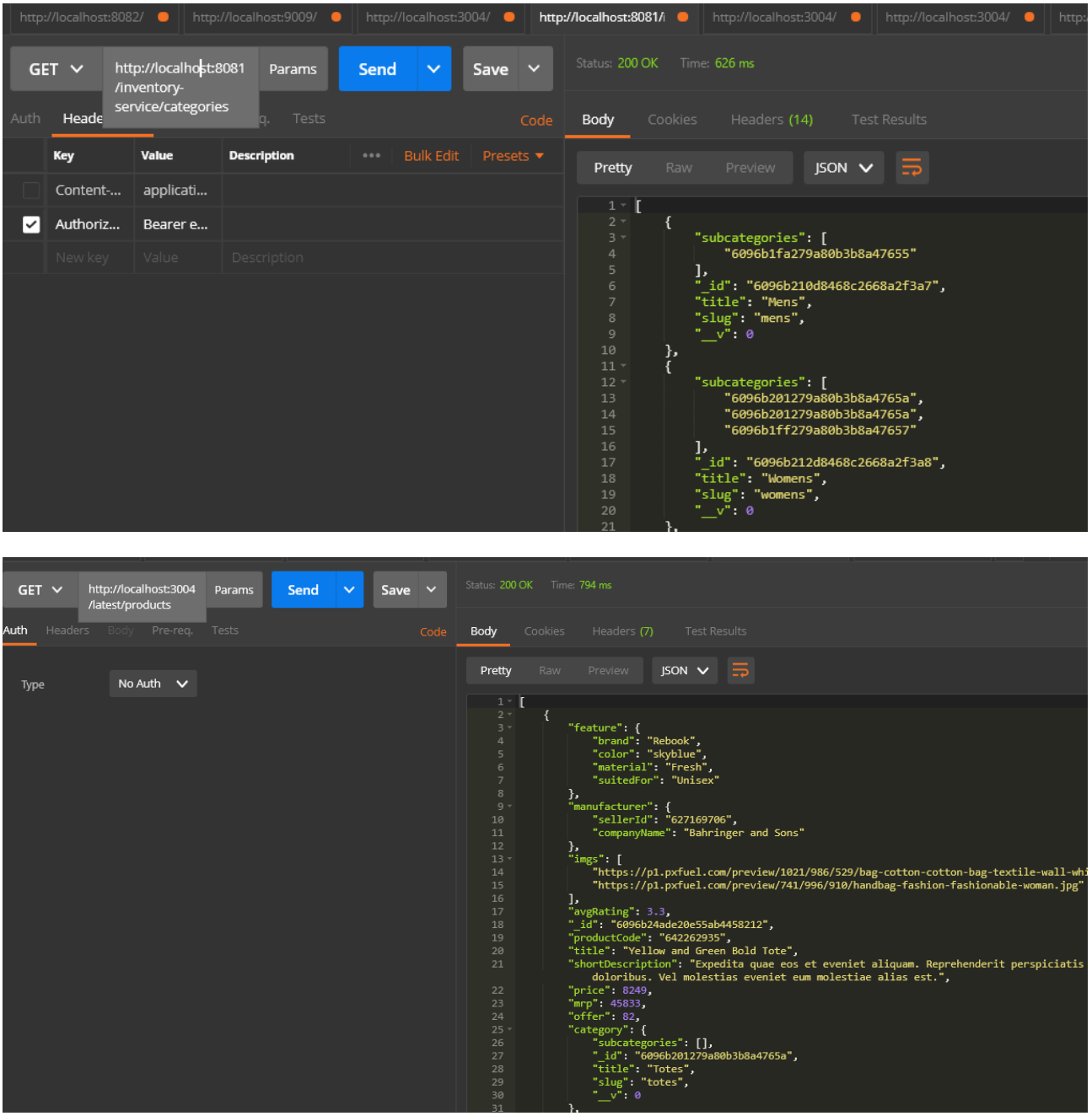

Fig: 5.1 Some unit testing on Inventory Service using POSTMAN

5.2 Integration Testing

In integration testing a system consisting of some of the modules integrated together to build a system that fulfills one or more system requirements is tested for any errors while integrating the modules after unit testing.

In my project I have done unit testing first and after that based on the system requirements, I started integrating the services one by one and after each integration again tested the sub module that generated after integrating the services, for finding the errors from service integration .

5.3 Validation Testing

Validation testing assures the system fulfills the functional and nonfunctional requirements. Validation testing is performed after the whole system is integrated successfully. In this testing Black-Box testing techniques are used. There are 3 primary parts in validation testing which are as follows,

- Validation measures
- Configuration survey
- A & B testing

Here A&B testing means alpha and beta testing in which alpha testing is done by the developer before the deployment and the beta testing is done after the deployment. As of now the project is not deployed online the beta testing hasn't been conducted. but I have done the Alpha testing after the Integration testing.

Here is an example of one of the test case I used for Testing,

Test Case - 1

- Adding the same product with the same size multiple times to cart increases the quantity of the product instead of creating a new row.
- Adding the same product with different sizes multiple times creates different rows for the different sizes.
- Making the quantity of the product in the cart, deletes the product from the cart.
- Any update in the cart reflected the Total Bill amount on the bottom of the cart.

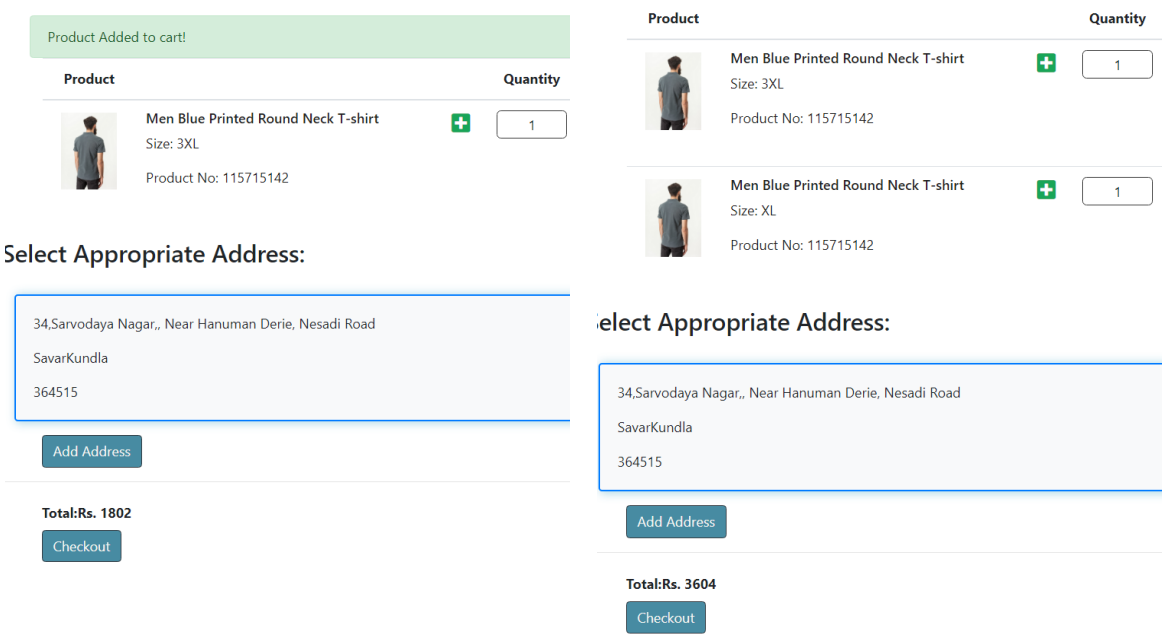

Fig 5.2 Checking above test case 1

5.4 Whitebox Testing

In whitebox testing the internal workings of the system is tested. This test assures the internal working of the system is performed as per the specifications. In white box testing test cases are generated to test the logical walkthrough of the system. These test cases assures that,

- All individual pathe and endpoints are tested at least once.
- Test all the decision making points.
- Test all loops for their boundaries.
- Test internal communication between services.

I have performed this type of testing parallely in each and every step in the coding phase. For an example in orderfullfiment service I have tested bellow scenarios,

- Order placed, Payment failed
- Order placed, payment succeed
- Order placed, payment succeed, product out of stock
- Order placed, payment succeeded then order cancelled by user.

Chapter 6: Conclusion

As part of feature improvement for the application we wish to implement a hybrid version of content-based and collaborative recommender systems. This will allow the application to recommend products with a greater accuracy. It will also lead to better content discovery for user.

6.1 Summary

We have implemented an E-commerce web application using the microservices architecture. All the services have been developed independent of each other and have their own service specific database. We have made use of secure authentication and authorization protocols like JWT-based authorization. For interservice communication we have made use of Kafka topics. This allows for asynchronous communication between services leading to faster performance. The services have been developed in Spring Boot and Express frameworks following best practices. For storing data, we have made use of MySQL and Mongo databases. After completing this project, we can surely say that we have better understood the principles behind the microservices architecture and efficient use of different tools and technologies.

6.2 Scope of Future work

As part of feature improvement for the application we wish to implement a hybrid version of content-based and collaborative recommender systems. This will allow the application to recommend products with a greater accuracy. It will also lead to better content discovery for users.

References :

1. https://www3.nd.edu/
2. Inverted Index - GeeksforGeeks
3. https://docs.atlas.mongodb.com/
4. https://developer.paytm.com/
5. https://stackoverflow.com/
6. https://github.com/
7. https://docs.spring.io/