International Institute of Information Technology, Bangalore

CS 816 / Software Production Engineering

Online Grocery Application

By

Rishabh Kumar Rai	MT2023088	Rishabh.Rai@iiitb.ac.in
Souvik Ghosh	MT2023122	Souvik.Ghosh@iiitb.ac.in

Introduction

This is an Online grocery Order Application built in React and microservice architecture. We have used Spring Boot framework for Backend and MySQL for Database services.

GitHub Repo (Old): https://github.com/Pappu98/Online-Grocery-Application.git

GitHub Repo (New): https://github.com/Pappu98/GroceryApp.git

DevOps tools:

Source Control Management: Git and GitHub

Continuous Integration Pipeline: Jenkins

Containerization: Docker

Container Orchestration: Docker compose

Frontend: React

Backend: Spring Boot

Monitoring: ELK Stack

Database: MySQL

Testing: Junit

Features:

Login

- Users can login using OTP (One Time Password) sent via email.

Add To Cart

- Users can add their product to cart.

View Previous Orders

- Users can view their previous orders.

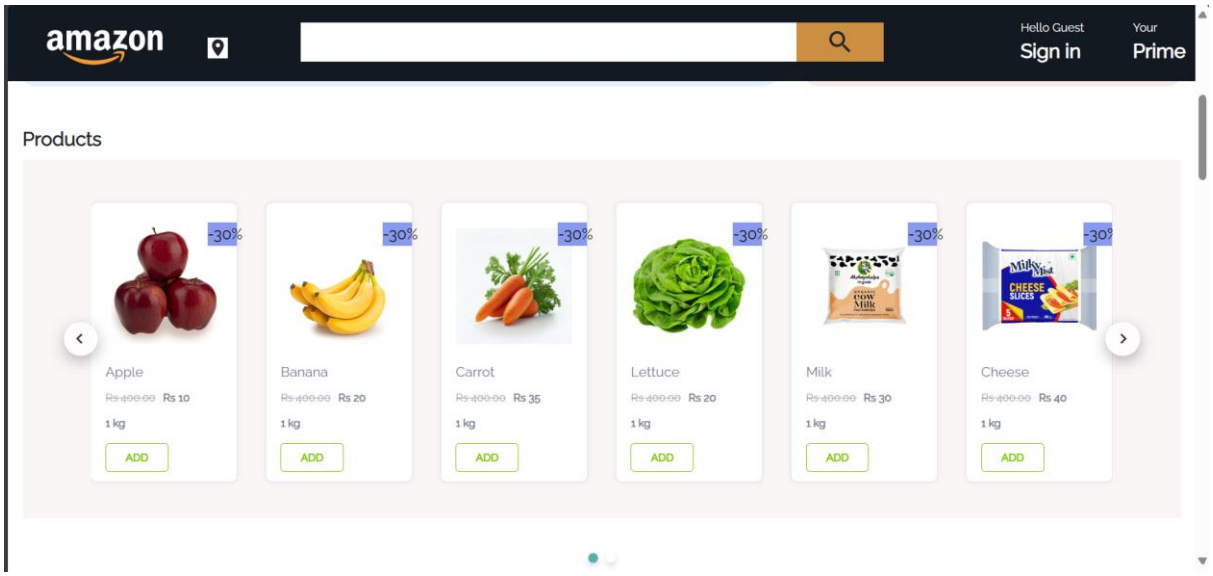
Add Address

- Users can add new addresses to place order.

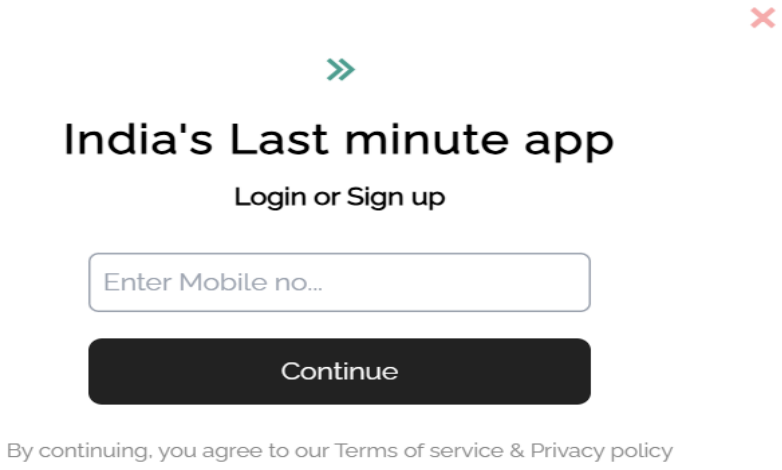
Place Order

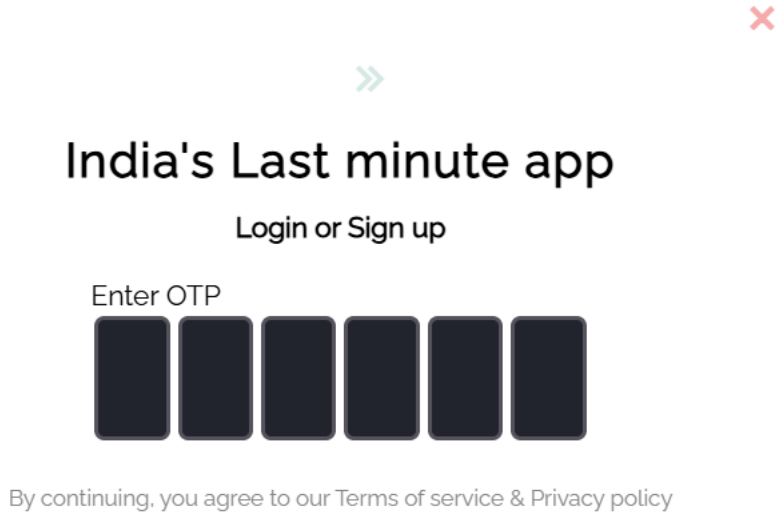
- Users need to add items to cart to place order.

List Of Products

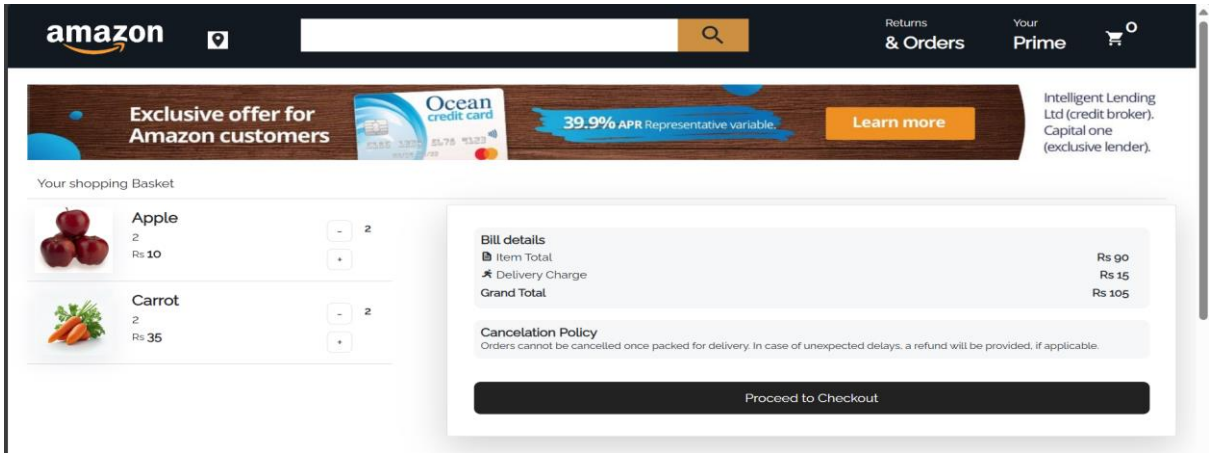


Login


India's Last minute app
Login or Sign up
Enter Mobile no...
Continue
By continuing, you agree to our Terms of service & Privacy policy


India's Last minute app
Login or Sign up
Enter OTP
By continuing, you agree to our Terms of service & Privacy policy

Cart


amazon
Returns
& Orders
Your
Prime
0
Exclusive offer for Amazon customers
Ocean credit card
39.9% APR Representative variable.
Learn more
Intelligent Lending Ltd (credit broker). Capital one (exclusive lender).
Your shopping Basket
Apple
2
Rs 10
2
Carrot
2
Rs 35
2
Bill details
Item Total
Rs 90
Delivery Charge
Rs 15
Grand Total
Rs 105
Cancelation Policy
Orders cannot be cancelled once packed for delivery. In case of unexpected delays, a refund will be provided, if applicable.
Proceed to Checkout

Address list

Delivery Address

Select Address

Souvik Ghosh, 9609119549

1B3, Alta Vista, Electronic City, , Bangalore, 560100

+ Add Another Address

Add New Address

Shipping
Please enter your shipping details.

FIRST NAME

LAST NAME

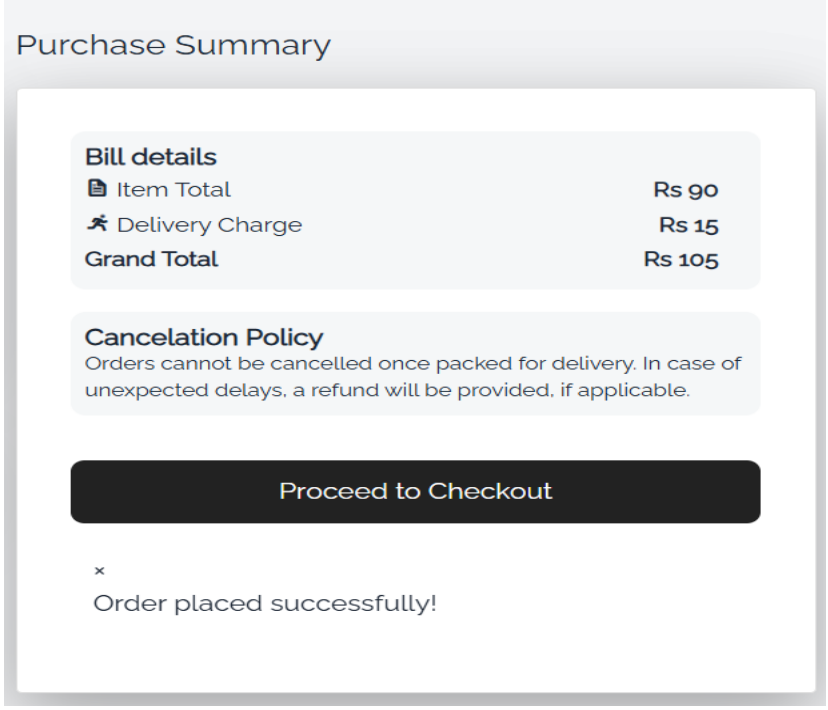
ADDRESS LINE 1

ADDRESS LINE 2

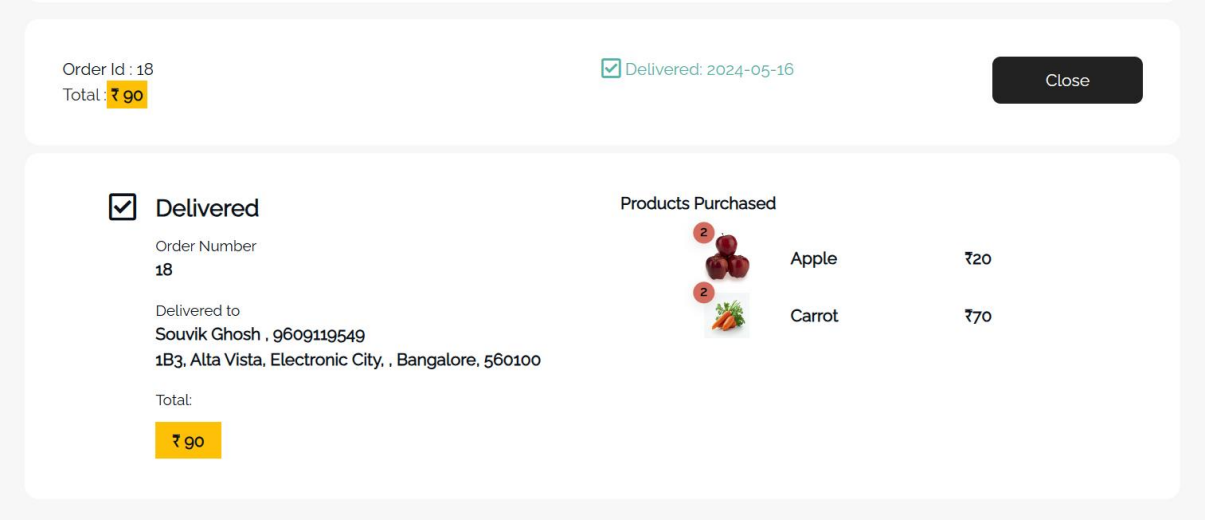
LANDMARK

PHONE NUMBER

Order Placed



Previous Orders



Microservices used:

Eureka Server: A service registry that helps in discovering and managing microservices within the application.

API Gateway: A single entry point that routes client requests to various microservices, providing load balancing and security.

Login Service: Handles user authentication.

OTP Service: Generates and verifies one-time passwords for user authentication.

Product Service: Manages the product catalog, including product listings, details, and inventory.

Cart Service: Manages users' shopping carts, including adding, removing, and updating items.

Order Service: Handles the creation and tracking of customer orders.

Junit Testing

JUnit testing is a Java-based framework for writing and executing unit tests. It employs annotations to define test methods and lifecycle management.

```java
@SpringBootTest
@AutoConfigureMockMvc
public class ProductControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetProduct() throws Exception {

        mockMvc.perform(get( urlTemplate: "/product-service/getProduct/2"))
                .andExpect(status().isOk())
                .andExpect(jsonPath( expression: "$.id").value( expectedValue: 2))
                .andExpect(jsonPath( expression: "$.productName").value( expectedValue: "Banana"))
                .andExpect(jsonPath( expression: "$.description").value( expectedValue: "Ripe yellow bananas"))
                .andExpect(jsonPath( expression: "$.price").value( expectedValue: 20.0))
                .andExpect(jsonPath( expression: "$.imageUrl").value( expectedValue: "https://www.shutterstock.com/image-photo/banana-cluster-i
                .andExpect(jsonPath( expression: "$.isAvailable").value( expectedValue: 1));
    }
}
```

ProductManagementApplication × ProductControllerTest ×

ProductControllerT 687 ms ✓ Tests passed: 1 of 1 test – 687 ms

Docker

Frontend Dockerfile:

amazon-clone > Dockerfile

```
FROM node:18
WORKDIR /app
COPY package.json ./
COPY package-lock.json ./
RUN npm install
COPY . ./
EXPOSE 3000
CMD ["npm", "start"]
```

Apigatewayservice Dockerfile:

D: > GroceryApp > Backend > API-GATEWAY > API-GATEWAY > Dockerfile

```
FROM openjdk:17
COPY target/apigateway.jar docker-apigateway.jar
ENTRYPOINT ["java","-jar","docker-apigateway.jar"]
```

Eurekaservice Dockerfile:

D: > GroceryApp > Backend > EurekaServer > EurekaServer > Dockerfile

```
FROM openjdk:17
COPY target/eurekaserver.jar docker-eurekaserver.jar
ENTRYPOINT ["java","-jar","docker-eurekaserver.jar"]
```

Loginservice Dockerfile:

D: > GroceryApp > Backend > LoginManagement > Dockerfile

```
FROM openjdk:17
COPY target/loginmanagement.jar docker-loginmanagement.jar
ENTRYPOINT ["java","-jar","docker-loginmanagement.jar"]
```

Otpservice Dockerfile:

D: > GroceryApp > Backend > OtpManagement > Dockerfile

```
FROM openjdk:17
COPY target/otpmanagement.jar docker-otpmanagement.jar
ENTRYPOINT ["java","-jar","docker-otpmanagement.jar"]
```

Orderservice Dockerfile:

D: > GroceryApp > Backend > OrderManagement > Dockerfile

```
FROM openjdk:17
COPY target/ordermanagement.jar docker-ordermanagement.jar
ENTRYPOINT ["java","-jar","docker-ordermanagement.jar"]
```

Productservice Dockerfile:

D: > GroceryApp > Backend > ProductManagement > Dockerfile

```
FROM openjdk:17
COPY target/productmanagement.jar docker-productmanagement.jar
ENTRYPOINT ["java","-jar","docker-productmanagement.jar"]
```

Cartservice Dockerfile:

```
D: > GroceryApp > Backend > ShoppingCartManagement > Dockerfile
FROM openjdk:17
COPY target/shoppingcartmanagement.jar docker-shoppingcartmanagement.jar
ENTRYPOINT ["java","-jar","docker-shoppingcartmanagement.jar"]
```

Jenkins

We used Jenkins pipeline scm from GitHub. The pipeline script was cloned from the GitHub repository and the code was also cloned from the same repository.

The first step is to clone the repository.

```
stages {
    stage('Checkout') {
        steps{
            script{
                git branch: 'master', url: "${GITHUB_REPO_URL}"
            }
        }
    }
```

The second step is to build all the images and upload to Docker Hub.

```
stage('Build and Push Images') {
    parallel {
        stage('Product Service') {
            steps {
                script {
                    dockerImage = docker.build("${IMAGE_NAME_PREFIX}-productservice", "./Backend/ProductManagement")
                    sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                    sh "docker tag ${IMAGE_NAME_PREFIX}-productservice ${DOCKERHUB_USERNAME}/speproject-productservice:latest"
                    sh "docker push ${DOCKERHUB_USERNAME}/speproject-productservice:latest"
                }
            }
        }

        stage('Cart Service') {
            steps {
                script {
                    dockerImage = docker.build("${IMAGE_NAME_PREFIX}-cartservice", "./Backend/ShoppingCartManagement")
                    sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                    sh "docker tag ${IMAGE_NAME_PREFIX}-cartservice ${DOCKERHUB_USERNAME}/speproject-cartservice:latest"
                    sh "docker push ${DOCKERHUB_USERNAME}/speproject-cartservice:latest"
                }
            }
        }
```

```
    stage('OTP Service') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-otpservice", "./Backend/OtpManagement")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-otpservice ${DOCKERHUB_USERNAME}/speproject-otpservice:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-otpservice:latest"
            }
        }
    }

    stage('Login Service') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-loginservice", "./Backend/LoginManagement")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-loginservice ${DOCKERHUB_USERNAME}/speproject-loginservice:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-loginservice:latest"
            }
        }
    }
```

```
    stage('Order Service') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-orderservice", "./Backend/OrderManagement")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-orderservice ${DOCKERHUB_USERNAME}/speproject-orderservice:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-orderservice:latest"
            }
        }
    }

    stage('API Gateway Service') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-apigatewayservice", "./Backend/API-GATEWAY/API-GATEWAY")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-apigatewayservice ${DOCKERHUB_USERNAME}/speproject-apigatewayservice:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-apigatewayservice:latest"
            }
        }
    }
```

```
    stage('Eureka Server') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-eurekaserver", "./Backend/EurekaServer/EurekaServer")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-eurekaserver ${DOCKERHUB_USERNAME}/speproject-eurekaserver:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-eurekaserver:latest"
            }
        }
    }

    stage('Frontend') {
        steps {
            script {
                dockerImage = docker.build("${IMAGE_NAME_PREFIX}-frontend", "./Frontend/amazon-clone")
                sh "docker login --username ${DOCKERHUB_USERNAME} --password ${DOCKERHUB_PASSWORD}"
                sh "docker tag ${IMAGE_NAME_PREFIX}-frontend ${DOCKERHUB_USERNAME}/speproject-frontend:latest"
                sh "docker push ${DOCKERHUB_USERNAME}/speproject-frontend:latest"
            }
        }
    }
}
```

Using Ansible, we can start the docker images by assigning the port and declaring the environment variables.

```
stage('Ansible Deployment') {
    steps {
        script {
            sh 'ansible-playbook -i inventory.ini deploy.yaml'
        }
    }
}
```

deploy.yaml has the permissions to the docker-compose and start docker images.

```
D: > GroceryApp > ! deploy.yaml
---
- name: Deploy Spring Boot Application
  hosts: all
  vars:
    ansible_python_interpreter: /usr/bin/python3

  tasks:
    - name: Copy Docker Compose file
      copy:
        src: docker-compose.yaml
        dest: "docker-compose.yaml"

    - name: Run Docker Compose
      command: docker-compose up --build -d
```

Using docker-compose

While using docker images, we used the mysql docker image to connect to the database. These images connect between them using a network name “springboot-mysql-network”.

This `docker-compose.yaml` file sets up a multi-service architecture for the application. It uses Docker to manage containerized services, enabling easy deployment and scaling. Here’s a brief description of each service and component:

1. Eureka Server (`eurekaserver`):

- Exposed on port 8761.

```yaml
eurekaserver:
  image: souvikiiitb/speproject-eurekaserver:latest
  ports:
    - "8761:8761"
  networks:
    - springboot-mysql-network
```

2. API Gateway (`apigatewayservice`):

- Depends on Eureka Server and runs on port 8765.

```yaml
apigatewayservice:
  image: souvikiiitb/speproject-apigatewayservice:latest
  ports:
    - "8765:8765"
  depends_on:
    - eurekaserver
  environment:
    - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
  networks:
    - springboot-mysql-network
```

3. Product Service (`productservice`):
- Depends on MySQL database and API Gateway, exposed on port 8081.

```yaml
services:
  productservice:
    image: souvikiiitb/speproject-productservice:latest
    ports:
      - "8081:8081"
    depends_on:
      - mysqldb
      - apigatewayservice
    environment:
      - SPRING_DATASOURCE_URL=jdbc:mysql://mysqldb:3306/groceryapp
      - SPRING_DATASOURCE_USERNAME=root
      - SPRING_DATASOURCE_PASSWORD=Pappu@1999
      - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
    networks:
      - springboot-mysql-network
    volumes:
      - /home/gsouvik/Desktop/SPE/GroceryApp/Backend/ProductManagement/app.log:/usr/share/filebeat/logs/app.log:rw
```

4. Cart Service (`cartservice`):

- Depends on Product Service, exposed on port 8083.

```yaml
cartservice:
  image: souvikiiitb/speproject-cartservice:latest
  ports:
    - "8083:8083"
  depends_on:
    - productservice
  environment:
    - SPRING_DATASOURCE_URL=jdbc:mysql://mysqldb:3306/groceryapp
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=Pappu@1999
    - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
    - SPRING_PROFILES_ACTIVE=docker
    - productservice.url=http://productservice:8081/product-service
  networks:
    - springboot-mysql-network
```

5. OTP Service (`otpservice`):

- Depends on MySQL database and API Gateway, exposed on port 8084.

```yaml
otpservice:
  image: souvikiiitb/speproject-otpservice:latest
  ports:
    - "8084:8084"
  depends_on:
    - mysqldb
    - apigatewayservice
  environment:
    - SPRING_DATASOURCE_URL=jdbc:mysql://mysqldb:3306/groceryapp
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=Pappu@1999
    - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
  networks:
    - springboot-mysql-network
```

6. Login Service (`loginservice`):

- Depends on OTP Service, exposed on port 8085.

```
loginservice:
  image: souvikiiitb/speproject-loginservice:latest
  ports:
    - "8085:8080"
  depends_on:
    - otpservice
  environment:
    - SPRING_DATASOURCE_URL=jdbc:mysql://mysqldb:3306/groceryapp
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=Pappu@1999
    - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
    - otpservice.url=http://otpservice:8084/otp-service
  networks:
    - springboot-mysql-network
```

7. Order Service (`orderservice`):

- Depends on Product Service and Cart Service, exposed on port 8082.

```
orderservice:
  image: souvikiiitb/speproject-orderservice:latest
  ports:
    - "8082:8082"
  depends_on:
    - productservice
    - cartservice
  environment:
    - SPRING_DATASOURCE_URL=jdbc:mysql://mysqldb:3306/groceryapp
    - SPRING_DATASOURCE_USERNAME=root
    - SPRING_DATASOURCE_PASSWORD=Pappu@1999
    - eureka.client.serviceUrl.defaultZone=http://eurekaserver:8761/eureka/
    - productservice.url=http://productservice:8081/product-service
    - cartservice.url=http://cartservice:8083/cart-service
  networks:
    - springboot-mysql-network
```

8. MySQL Database (`mysqldb`):

- Exposed on port 3307, with data persistence through a Docker volume.

```yaml
mysqldb:
  image: mysql:latest
  ports:
    - "3307:3306"
  environment:
    - MYSQL_DATABASE=groceryapp
    - MYSQL_ROOT_PASSWORD=Pappu@1999
  volumes:
    - mysql-data:/var/lib/mysql
  networks:
    - springboot-mysql-network
```

9. Frontend (`frontend`):

- Depends on API Gateway and runs on port 3000.

```yaml
frontend:
  image: souvikiiitb/speproject-frontend:latest
  ports:
    - "3000:3000"
  depends_on:
    - apigatewayservice
  networks:
    - springboot-mysql-network
```

10. Elasticsearch (`elasticsearch`):

- A search engine for indexing and searching application data.

- Runs on port 9200.

```yaml
elasticsearch:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.17.6
  container_name: elasticsearch
  environment:
    - discovery.type=single-node
  ports:
    - "9200:9200"
  networks:
    - springboot-mysql-network
```

11. Logstash (`logstash`):

- A data processing pipeline for ingesting logs and sending them to Elasticsearch.

- Uses a custom configuration file and runs on port 5044.

```
logstash:
  image: docker.elastic.co/logstash/logstash:7.17.6
  container_name: logstash
  volumes:
    - /home/gsouvik/Desktop/SPE/GroceryApp/Backend/logstash/pipeline/logstash.conf:/usr/share/logstash/pipeline/logstash.conf
  ports:
    - "5044:5044"
  networks:
    - springboot-mysql-network
```

12. Kibana (`kibana`):

- A visualization tool for viewing and analyzing logs stored in Elasticsearch.

- Runs on port 5601.

```
kibana:
  image: docker.elastic.co/kibana/kibana:7.17.6
  container_name: kibana
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
  ports:
    - "5601:5601"
  networks:
    - springboot-mysql-network
```

13. Filebeat (`filebeat`):

- A log shipper that forwards logs from the application to Logstash.

- Uses a custom configuration file and depends on Logstash.

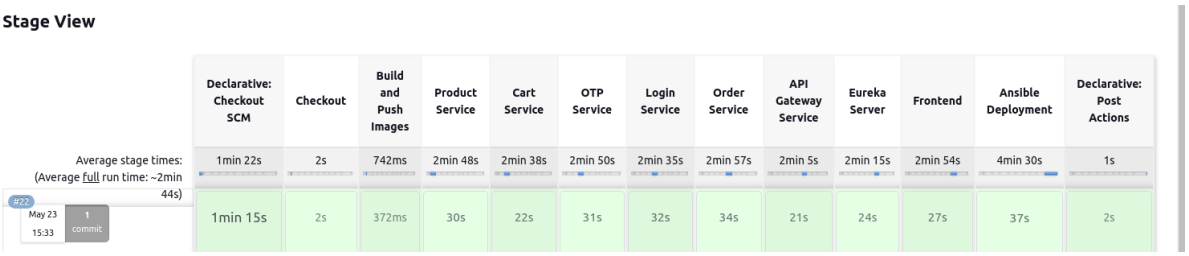
```
filebeat:
  image: docker.elastic.co/beats/filebeat:7.17.6
  container_name: filebeat
  volumes:
    - /home/gsouvik/Desktop/SPE/GroceryApp/Backend/filebeat.yaml:/usr/share/filebeat/filebeat.yaml:ro
    - /home/gsouvik/Desktop/SPE/GroceryApp/Backend/ProductManagement/app.log:/usr/share/filebeat/logs/app.log:ro
  networks:
    - springboot-mysql-network
  depends_on:
    - logstash
```

Networks and Volumes:

Network (`springboot-mysql-network`): Ensures that all services can communicate with each other within the same Docker network.

Volume (`mysql-data`): Provides persistent storage for MySQL database data, ensuring data is not lost between container restarts.

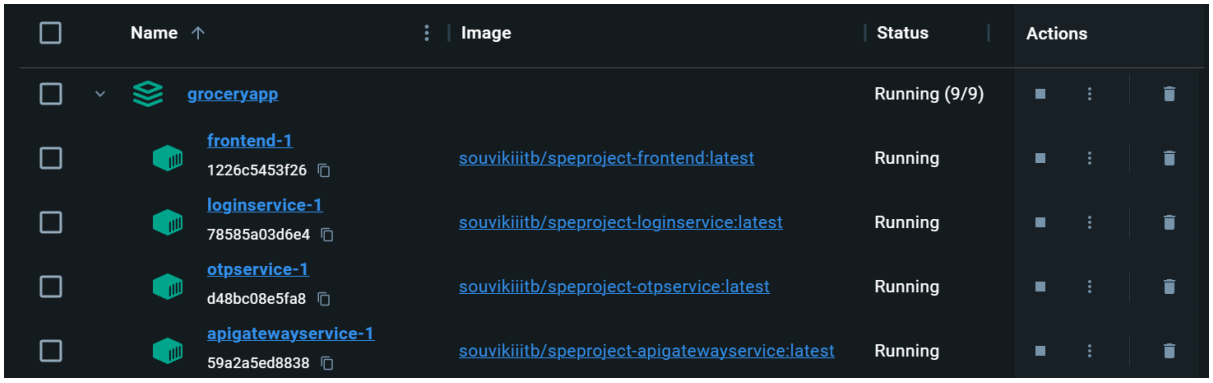
Jenkins Pipeline Result



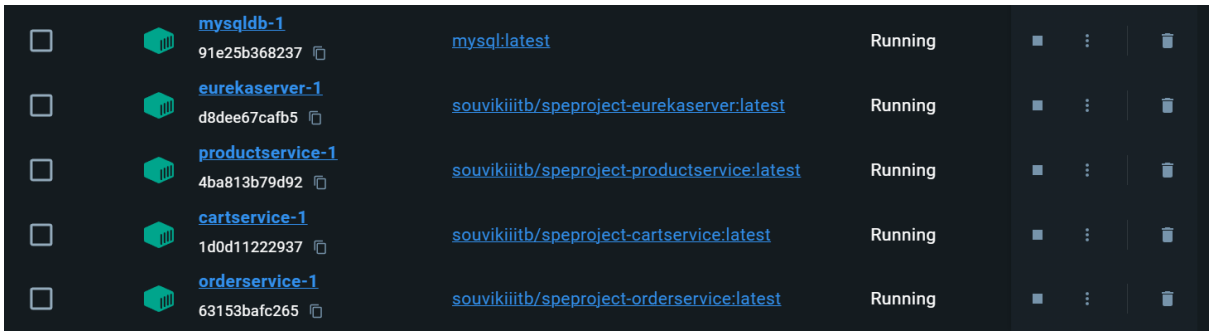
Stage View

	Declarative: Checkout SCM	Checkout	Build and Push Images	Product Service	Cart Service	OTP Service	Login Service	Order Service	API Gateway Service	Eureka Server	Frontend	Ansible Deployment	Declarative: Post Actions
Average stage times: (Average full run time: ~2min 44s)	1min 22s	2s	742ms	2min 48s	2min 38s	2min 50s	2min 35s	2min 57s	2min 5s	2min 15s	2min 54s	4min 30s	1s
#22 May 23 15:33 1 commit	1min 15s	2s	372ms	30s	22s	31s	32s	34s	21s	24s	27s	37s	2s

Deployment from docker-compose

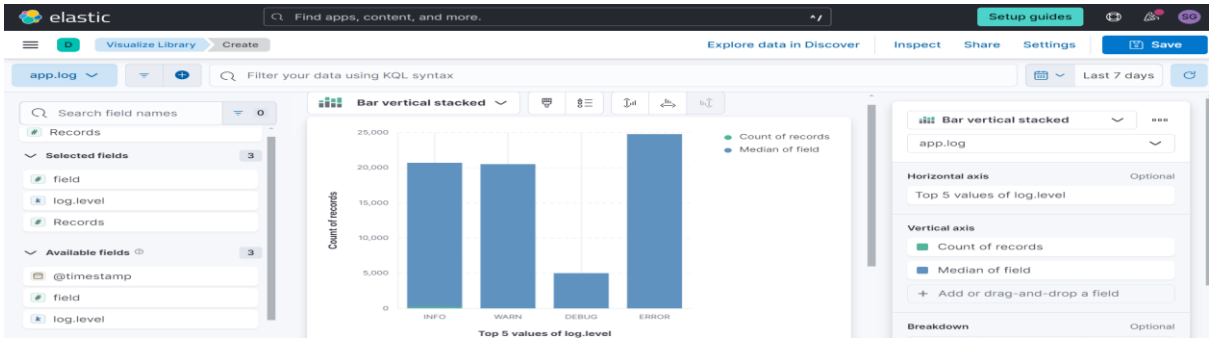


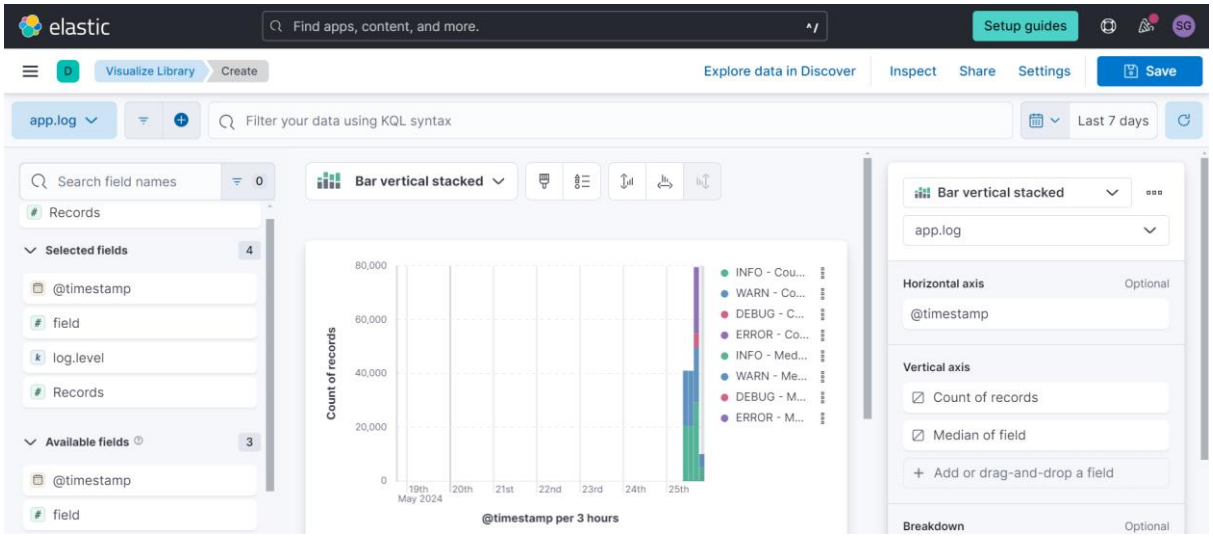
Name	Image	Status	Actions
groceryapp		Running (9/9)	
frontend-1 1226c5453f26	souvikiiitb/speproject-frontend:latest	Running	
loginservice-1 78585a03d6e4	souvikiiitb/speproject-loginservice:latest	Running	
otpservice-1 d48bc08e5fa8	souvikiiitb/speproject-otpservice:latest	Running	
apigatewayservice-1 59a2a5ed8838	souvikiiitb/speproject-apigatewayservice:latest	Running	

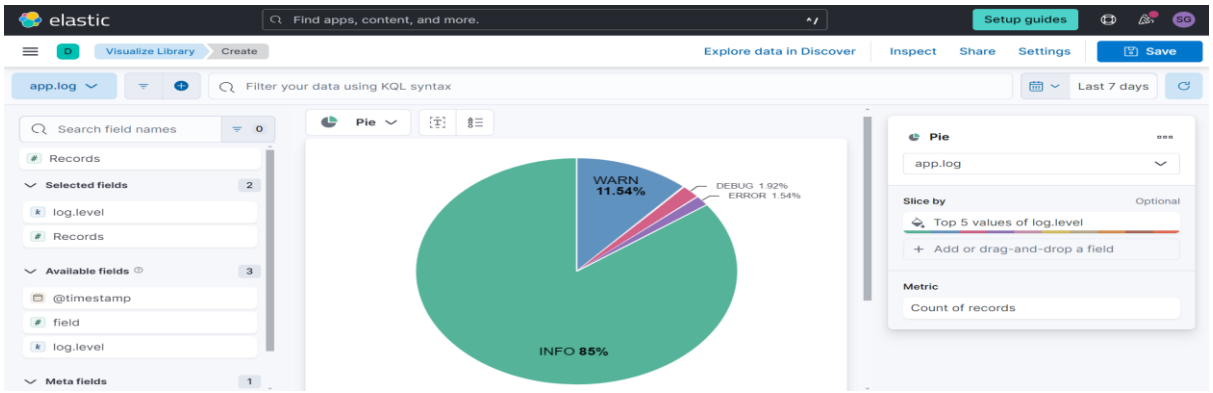


mysqldb-1 91e25b368237	mysql:latest	Running	
eurekaserver-1 d8dee67cafb5	souvikiiitb/speproject-eurekaserver:latest	Running	
productservice-1 4ba813b79d92	souvikiiitb/speproject-productservice:latest	Running	
cartservice-1 1d0d11222937	souvikiiitb/speproject-cartservice:latest	Running	
orderservice-1 63153bafc265	souvikiiitb/speproject-orderservice:latest	Running	

ELK dashboard visualisation of logs




elastic
Find apps, content, and more.
Setup guides
Visualize Library
Create
Explore data in Discover
Inspect
Share
Settings
Save
app.log
Filter your data using KQL syntax
Last 7 days
Search field names
Records
Selected fields
@timestamp
field
log.level
Records
Available fields
@timestamp
field
Bar vertical stacked
Count of records
80,000
60,000
40,000
20,000
0
19th
May 2024
20th
21st
22nd
23rd
24th
25th
@timestamp per 3 hours
INFO - Cou...
WARN - Co...
DEBUG - C...
ERROR - Co...
INFO - Med...
WARN - Me...
DEBUG - M...
ERROR - M...
Bar vertical stacked
app.log
Horizontal axis
Optional
@timestamp
Vertical axis
Count of records
Median of field
Add or drag-and-drop a field
Breakdown
Optional


elastic
Find apps, content, and more.
Setup guides
Visualize Library
Create
Explore data in Discover
Inspect
Share
Settings
Save
app.log
Filter your data using KQL syntax
Last 7 days
Search field names
Records
Selected fields
log.level
Records
Available fields
@timestamp
field
log.level
Meta fields
Pie
WARN
11.54%
DEBUG 1.92%
ERROR 1.54%
INFO 85%
Pie
app.log
Slice by
Optional
Top 5 values of log.level
Add or drag-and-drop a field
Metric
Count of records

API Documentation

Sno	API URL	Description
1.	/product-service/getAllProducts	Get all registered products
2.	/product-service/getProduct/{id}	Get details of a particular product
3.	/product-service/getAllCategories	Get all product categories
4.	/product-service/getSetOfProducts	Get product list based on category
6.	/login-service/login	Login a user verifying OTP
7.	/order-service/placeOrder	Place order
8.	/order-service/getAllOrders/{id}	Get previous orders of a user
9.	/order-service/getOrderDetails/{id}	Get details of a particular order
10.	/order-service /getAllAddresses/{id}	Get all addresses of a user
11.	/otp-service/otpVerification/{id}	Send OTP to a user
12.	/otp-service /getOtp/{id}	Fetch OTP of a user
13.	/cart-service/getCart/{id}	Get cart of a user
14.	/cart-service/updateCart	Add/Remove items from cart of a user