A Project Report on

Krishisetu: Connecting Farmers and Consumers

Submitted in partial fulfillment of the requirements for the award
of the degree of

Bachelor of Engineering

in

Information Technology

by

Shreyas Chorge(17104022)
Vedangi Naigaonkar(16104046)
Abhijit Ambre(17104030)

Under the Guidance of

Prof. Anagha Aher
Prof. Neha Deshmukh

Department of Information Technology
NBA Accredited
A.P. Shah Institute of Technology
G.B.Road,Kasarvadavli, Thane(W), Mumbai-400615
UNIVERSITY OF MUMBAI

Academic Year 2020-2021

Approval Sheet

This Project Report entitled ***"Krishi Setu:Connecting Farmers and Consumers"*** Submitted by ***"Shreyas Chorge"(17104022), "Vedangi Naigaonkar"(16104046),"Abhijit Ambre"(17104030)***is approved for the partial fulfillment of the requirement for the award of the degree of ***Bachelor of Engineering*** in ***Information Technology*** from ***University of Mumbai***.

(Prof. Neha Deshmukh)
Co-Guide

(Prof. Anagha Aher)
Guide

Prof. Kiran Deshpande
Head Department of Information Technology

Place: A.P.Shah Institute of Technology, Thane
Date:

CERTIFICATE

This is to certify that the project entitled ***"Krishi Setu: Connecting Farmers and Consumers "*** submitted by ***"Shreyas Chorge" (17104022),"Vedangi Naigaonkar" (16104046),"Abhijit Ambre" (17104030)*** for the partial fulfillment of the requirement for award of a degree ***Bachelor of Engineering*** in ***Information Technology***,to the University of Mumbai,is a bonafide work carried out during academic year 2020-2021.

(Prof. Neha Deshmukh)
Co-Guide

(Prof. Anagha Aher)
Guide

Prof. Kiran Deshpande
Head Department of Information Technology

Dr. Uttam D.Kolekar
Principal

External Examiner(s)

1.

2.

Place: A.P.Shah Institute of Technology, Thane
Date:

Acknowledgement

We have great pleasure in presenting the report on **Krishi Setu: Connecting Farmers and Consumers** We take this opportunity to express our sincere thanks towards our guide **Prof. Anagha Aher** & Co-Guide **Prof. Neha Deshmukh** Department of IT, APSIT thane for providing the technical guidelines and suggestions regarding line of work. We would like to express our gratitude towards his constant encouragement, support and guidance through the development of project.

We thank **Prof. Kiran B. Deshpande** Head of Department,IT, APSIT for his encouragement during progress meeting and providing guidelines to write this report.

We thank **Prof. Vishal S. Badgujar** BE project coordinator, Department of IT, APSIT for being encouraging throughout the course and for guidance.

We also thank the entire staff of APSIT for their invaluable help rendered during the course of this work. We wish to express our deep gratitude towards all our colleagues of APSIT for their encouragement.

Shreyas Chorge
17104022

Vedangi Naigaonkar
16104046

Abhijit Ambre
17104030

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

_____

(Shreyas Chorge,17104022)

_____

(Signature)

_____

(Vedangi Naigaonkar,16104046)

_____

(Signature)

_____

(Abhijit Ambre,17104030)

Date:

Abstract

Distributed messaging structures are the core for Modern applications build with micro services architecture. Messaging technologies are every increasing need for modern applications which have to deal with a heavy traffic. These technologies are crucial for micro services, cloud native applications and data streaming applications.

With ever increasing traffic scalability of the application is important factor. It must be done in such a way that it is provides all the necessary services to its users and should not cost more than necessary while maintaining security and the integrity of the users using the application.

Krishi Setu is an application that is build on the idea of loosely coupled micro services.

This report explains the architecture of the application, different approaches to solve a single problem and which one would be best suited to build an application and how the application is build.

Contents

List of Figures

List of Abbreviations

JSON: JavaScript Object Notation
JWT: JSON Web Token
HTTP: Hypertext Transfer Protocol
HTTPS: Hypertext Transfer Protocol Secure
Auth: Authentication
Srv: Service
depl: Deployment
S3: Simple Storage Service
API: Application Programming Interface
AWS: Amazon Web Services
HTML: HyperText Markup Language
URL: Uniform Resource Locator

Chapter 1

Introduction

Farmers these days do not get proper value for the farm produce. They strive so hard yet are compelled to sell their produce at a low price. The middleman forces the farmer to sell his produce at a low rate.

These middlemen then sell the same farm produce in the cities or to distributors at a high cost. Thus, the middlemen are at an advantage while the farmers and vendors suffer a huge amount of loss.

Sometimes, these middlemen also indulge in malpractices. Instead of going through the chain of middlemen, farmers can directly sell their goods to the distributors of major cities.

Krishisetu allows the farmers to register themselves and sell their goods online.

Chapter 2

Literature Review

The papers referred are mentioned below:

[1] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, Ferhat Khendek "Kubernetes as an Availability Manager for Micro service Applications"

The authors have presented a brief overview of Kubernetes architectural components and architectures for deploying micro services-based application with Kubernetes.The authors have investigated the impact of adding redundancy on the availability of micro service-based applications and performed experiments under Kubernetes default configuration and its most responsive one. They have also performed a comparative evaluation with the Availability Management Framework (AMF), a proven solution as a middleware service for managing high-availability.
In Kubernetes, we have different master node components i.e Kube API-server, etcd, Kube-scheduler due to any failure if this single master node fails this cause a big impact on business. so to solve this issue we deploy multiple master nodes to provides high availability for a single cluster and improves performance.

[2] L Magnoni "Modern Messaging for Distributed Systems"

L Magnoni has discussed the Importance of loosely coupled communication, Connection-oriented communication, Messaging for loosely coupled communication, Messaging scenario, Messaging middleware. Author has also explained the basic Messaging Terminologies and compared different messaging services available and also their use cases.

[3] Pooja J Bhat, Priya D "Modern Messaging Queues - RabbitMQ, NATS and NATS Streaming"

Authors have explained in detail about the Messaging Technologies and Terminologies. Authors have also compared and explained the features of these technologies and provided the benchmark for RabbitMQ and NATS. They have also explained the architecture of RabbitMQ and NATS

Chapter 3

Project Design

3.1 Architecture of Krishisetu

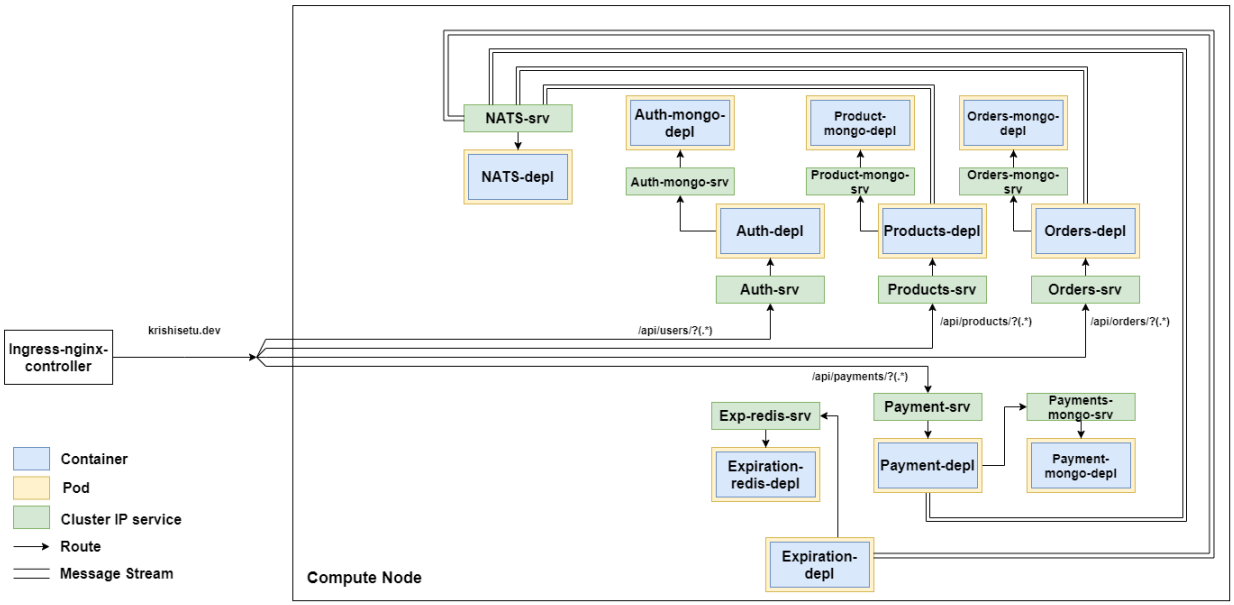

Figure 3.1: Architecture of Krishisetu

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource. **Ingress-Nginx** is the entry point to our application. Nginx is used as a proxy server to route the traffic based on the path. for example, any traffic coming at krishisetu.dev/api/users/(.?*) will be routed to the authentication service. It exposes the application to outer world.

Auth-depl holds all the code logic related to authentication is running inside the docker container which interns runs inside a Pod.

Auth-srv is a Cluster-IP service responsible for incoming and outgoing traffic from the auth Pod.

Auth-mongo-depl is pod which is running the Container of mongodb image, it is used to store the users information.

Products-depl holds the product-microservice running inside the docker container which interns runs inside a Pod.

Products-srv is a Cluster-IP service responsible for incoming and outgoing traffic from the Product microservice.

Products-mongo-depl is pod which is running the Container of mongodb image, it is used to store the products information.

Orders-depl holds the order-microservice running inside the docker container which interns runs inside a Pod.

Orders-srv is a Cluster-IP service responsible for incoming and outgoing traffic from the Order microservice.

Orders-mongo-depl is pod which is running the Container of mongodb image, it is used to store the orders information.

Payments-depl holds the payments-microservice running inside the docker container which interns runs inside a Pod.

Payments-srv is a Cluster-IP service responsible for incoming and outgoing traffic from the Payments microservice.

Payments-mongo-depl is pod which is running the Container of mongodb image, it is used to store the Payments information.

NATS-depl is running a nats-streaming image inside a docker container which interns running inside a Pod. NATS is responsible for the communication between all these isolated micro-services.

NATS-srv is a Cluster-IP service responsible for incoming and out going traffic from the NATS-depl.

Expiration-depl holds expiration micro-service responsible for orders expiration.

Redis-depl is running a redis instance.

Chapter 4

Project Implementation

4.1 Authentication and Authorization Strategies

4.1.1 Approaches of Authentication in Micro-services

(A) Relying on Auth-Service for authentication

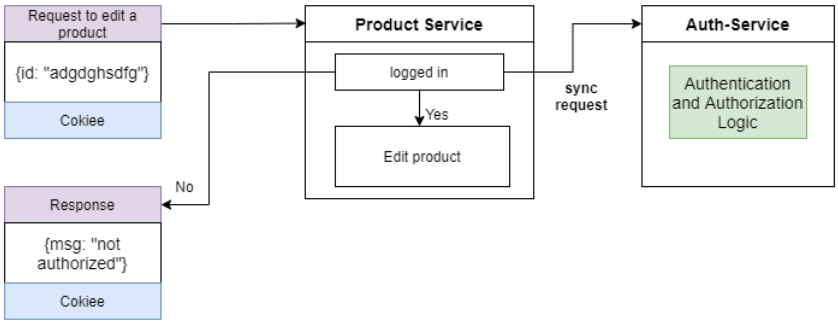

Figure 4.1: Authentication approach 1

Consider a scenario where a user wants to edit a product. Only an authorized user (i.e the one who has created a product) must be able to edit it. With this approach, considering a user is authenticated, whenever a user tries to edit a product we could make a synchronous request to Auth-Service to check weather a user is authorized or not.

In such a case Product-Service is entirely dependent on the Auth-Service. If Auth-Service ever went down, all services processing the request which needs authentication will have to wait until Auth-Service comes back up. This will stall the application and introduce down-time. This approach shares all the downsides of synchronous communication.

(B) Teaching Services how to authenticate a user

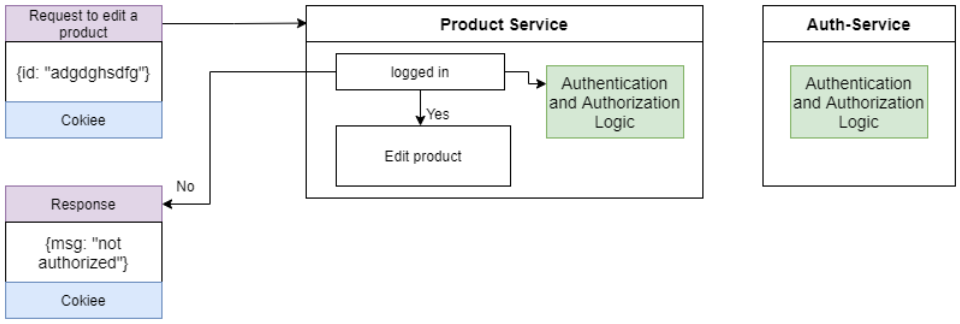

Figure 4.2: Authentication approach 2

Embedding the authentication logic inside the micro-service holds up to the idea of loosely coupled services. Considering the above scenario, in this case, even if the Auth-Service went down Product-Service will still be able to serve our users.

Teaching Services how to authenticate introduces code replication. Code replication/sharing will be covered in Section 4.4 titled Code Sharing.

May this approach seems like a go-to approach there is still a downside to this approach. Consider a scenario where we have a malicious user who authenticates himself/herself in past. To ban this user from accessing the application we could add a field to our database(eg. { isbanned = true }). This user will still have an old token/cookie that he/she can use to access the application. We need a token/cookie to expire after a certain time. Once the token/Cookie is expired the banned user has to authenticate himself/herself again through Auth-Service. This is where he/she will fail to access the application even though he/she could still have access until the token/cookie is expired. So we need this method build in within a token/cookie. This brings us to a discussion of the requirements for authentication.

4.1.2 Requirements for Authentication

A token/cookie
(a) Must tell us the details about the person.
(b) Must have build-in support to handle authorization.
(c) Must have build-in tamper-resistant expiration mechanism.
(d) Must have support for different languages.
(e) Must not require a backing data store.

JSON Web Token is an Internet proposed standard for creating data with optional signature and/or optional encryption whose payload holds JSON. It meets all the above mentioned requirements.

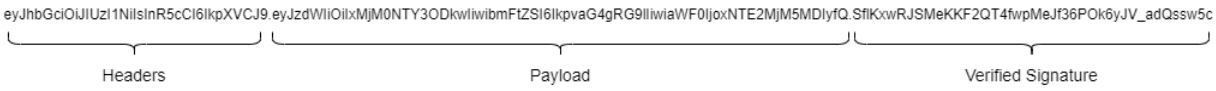

Figure 4.3: Example of JWT

Authentication Mechanism
So now that we have discussed what we could use to authenticate users, lets see how we could handle transferring JWT token over HTTP.

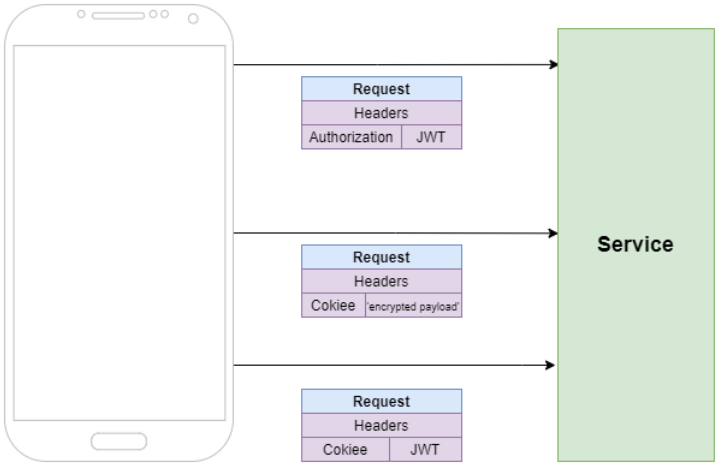

Figure 4.4: Authentication Mechanisms

(a) Sending a JWT token in an Authorization header
(b) Sending an encrypted cookie
(c) Sending a JWT token inside an encrypted cookie

Krishisetu uses approach (c) On the server side cookies is managed by 'cookie-session' npm module. 'cookie-session' is a simple session middleware.

- cookie-session does not require any database / resources on the server side, though the total session data cannot exceed the browser's max cookie size.

- cookie-session can simplify certain load-balanced scenarios.

- cookie-session can be used to store a "light" session and include an identifier to look up a database-backed secondary store to reduce database lookups.

4.2 Scalable Image Upload

Dealing with images is an essential part of any application. There are various approaches to store images. We could use storage services provided by multiple cloud providers. We are using AWS S3 for Krishisetu. AWS provides a service called S3 (Simple Storage Service) where we can store all their files. AWS S3 charges 0.023$ (1.7 Rupee) per GB. Thus making it one of the most affordable service to store data.

4.2.1 Image Upload Constraints

1. Only authenticated users can upload an image

It could be done on the front-end side of the application where we could simply do not show the upload button to the unauthenticated user. Even though this seems like a straight forward approach we must take into consideration that there will be malicious users who will try to exploit our endpoint. Thus we have to make API endpoint tamper-resistant.

2. The Uploaded image must be tied with the product that will be created

We must figure out a way to get the uploaded image from the S3 and associate it with the product inside the database

3. We should only allow images

We must also have to make sure that the user uploads only images to s3. Changing the extension of the files (.mp4 to .jpg) and uploading it to S3 must not be supported.

4.2.2 Approaches for Image Upload

(A) Two Hop Approach

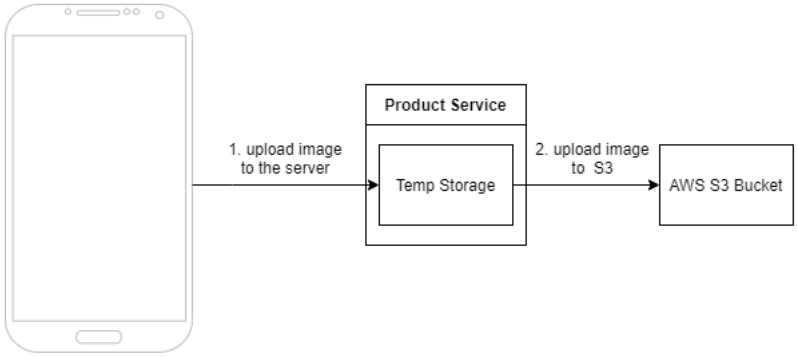

Figure 4.5: Two Hop Approach

In this approach, we will upload the image to our service and then the service will upload the image to AWS S3. It could make handling some of our image upload constraints much easier. When an image is uploaded to the service we could take a look at JWT token and verify whether a user is authenticated or not.We could easily associate our image with the

product so that they could refer to each other. Though it is an easy approach it has a downside to it.

Whenever we upload an image to service our node server takes up some computational resources. As the application grows images will be uploaded at a large scale this will take up more computational resources which in turn charge us an extra amount to use these resources.

(B) Pre-signed URL

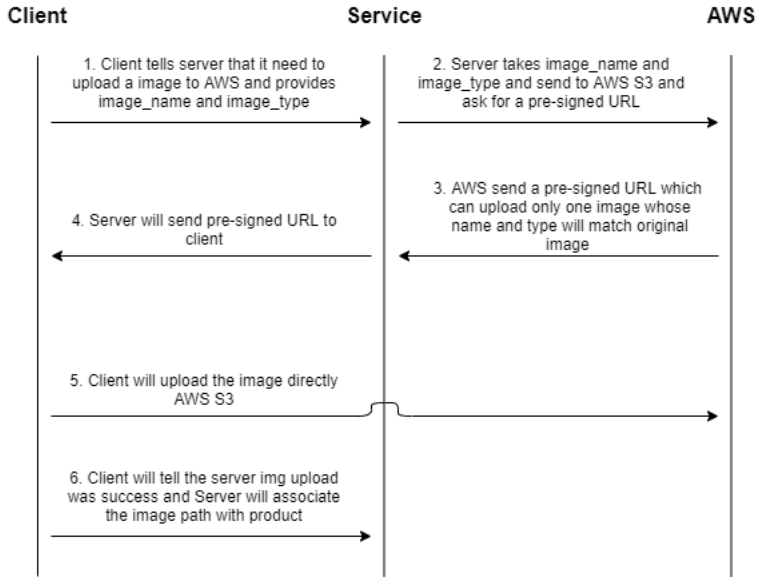

Figure 4.6: Flow of Image Upload

A user who does not have AWS credentials or permission to access an S3 object can be granted temporary access by using a presigned URL. A presigned URL is generated by an AWS user (Service) who has access to the object. The generated URL is then given to the unauthorized user (Client). The presigned URL can be entered in a browser or used by a program or HTML webpage or moblie application to be fetched as a network image. The credentials used by the presigned URL are those of the AWS user who generated the URL (Service). This way only our service can write data to a S3 bucket and not a random malicious user. If we allow a public write access to our S3 bucket a random user will store all their data to our S3 bucket which will serve no purpose to our users and we will end up paying for the resources used by a random malicious user.

Example of a pre-signed URL

https://krishisetu.s3.ap-south-1.amazonaws.com/test/apple.jpeg?Content-Type=image%2Fjpeg&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=MY_AWS_ACCESS_KEY_ID%2Fap-south-1%2Faws4_request&X-Amz-Date=20210408T002339Z&X-Amz-Expires=900&X-Amz-Signature=6092937d89d0-s-d0g0-09fdb2b7a9402229bc1bee24c628382bd183ba3dcb80f8&X-Amz-SignedHeaders=host

Domain	https://krishisetu.s3.ap-south-1.amazonaws.com
FileName	apple.jpeg
AWSAccessKeyID	MY_AWS_ACCESS_KEY_ID
Content-type	image%2Fjpeg => image/jpeg
ExpiresAt	900 => 15 min (default)
Bucket Region	ap-south-1
Signature	6092937d89d0-s-d0g0-09fdb2b7a9402229bc1bee24c628382bd183ba3dcb80f8

Figure 4.7: Example of pre-Signed URL

Pre-signed URL by default expires after 15 min but this can be customized.

Following is the code snippet to generate a pre-signed URL. getSignedUrlPromise() method generates a pre-signed URL. We send bucket-name, content-type and name of our file (key in this example) to S3 which is used by S3 to generate a signature. A Signature verifies that the URL is generated by using AWS API with proper credentials and it prevents the users from fabricating their own link and maliciously upload their own file.

```
AWS.config.update({
  region: 'ap-south-1',
  accessKeyId: process.env.AWS_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SECRET_ACCESS_KEY
})

const s3 = new AWS.S3();

router.get('/api/products/upload/', requireAuth, async (req: Request, res: Response) => {

  const key = `${req.currentUser!.id}/${uuid()}.jpeg`;

  const url = await s3.getSignedUrlPromise(
    'putObject',
    {
      Bucket: 'krishisetu',
      ContentType: 'image/jpeg',
      Key: key
    },
  );

  res.send({key, url}).status(200);
});
```

Figure 4.8: Code Snippet for generating pre-signed URL

Security Issues solved by pre-signed URL

- URL can be used to upload only a single file, thus preventing users from spaming our s3 bucket.

- URL encodes the file-name and file-type so users cannot request for one file and upload another file.

- URL expires after a certain duration, preventing users to reuse the same link.

- URL is generated by an AWS user who has access to the object.

- URL only works for a single bucket, preventing users from uploading their files to a different bucket

Krishisetu uses approach (b) pre-signed URL for image upload.

4.3 Message Streaming

4.3.1 Introduction to NATS Streaming

Stream protocols send a continuous flow of data.

NATS is a connective technology that powers modern distributed systems. A connective technology is responsible for addressing and discovery and exchanging of messages that drive the common patterns in distributed systems; asking and answering questions, aka services/micro services, and making and processing statements, or stream processing.

NATS Streaming is a data streaming system powered by NATS, and written in the Go programming language. The executable name for the NATS Streaming server is 'nats-streaming-server'. NATS Streaming embeds, extends, and inter operates seamlessly with the core NATS platform. The NATS Streaming server is provided as open source software under the Apache-2.0 license. Synadia actively maintains and supports the NATS Streaming server.

Features

- Enhanced message protocol
- Message/event persistence
- At-least-once-delivery
- Publisher rate limiting
- Rate matching/limiting per subscriber
- Historical message replay by subject
- Durable subscriptions

4.3.2 Working of node-nats-streaming

We create a client (also referred as Stan in NATS Documentation) which listens to a specific event i.e 'connect' into which we need to specify **ClusterId** - The cluster that the instance of client is going to connect. **ClientId** - Id of this instance of the client. This field has to be unique for a client across the NATS Streaming Server. **NATS URL** - URL of NATS Streaming server.

Once we have an instance of a client connected to the NATS Streaming Server, we can use it to subscribe to a topic and publish a message or listen to a message from that specific topic.

Default Behaviour of Messages : If a message consumed by the a service and we fail to handle the message. The message is lost. If we are handling a critical message we don't want it to get lost. To avoid this behaviour we can add an option **setManualAckMode(true)** to subscription options. If we do not send the acknowledgement NATS will send this same message to another service in a queue group or same service subscribed to a topic.

Queue Groups : Queue Groups are within the topic. They limit the number of messages that goes to a client subscribed to a queue group. Multiple clients would be subscribing to a Queue Group within the topic. It will send a message to only one client subscribed to it.

Durable Subscriptions : Enabling **setDeliverAllAvailable()** will send all the message to a service who just come back alive after being down. NATS persistence all the messages in memory (or optionally in file Store, SQL Store). Consider a scenario where a service went down and came back up, NATS would have thousands of millions of messages stored in it. Sending all these messages to a service which just came back alive and processing all these messages is not efficient. To avoid this, we use **setDurableName()**. Durable Subscriptions only sends those messages which is not processed for this to happen durable subscription works with queue group and setDeliverAllAvailable().

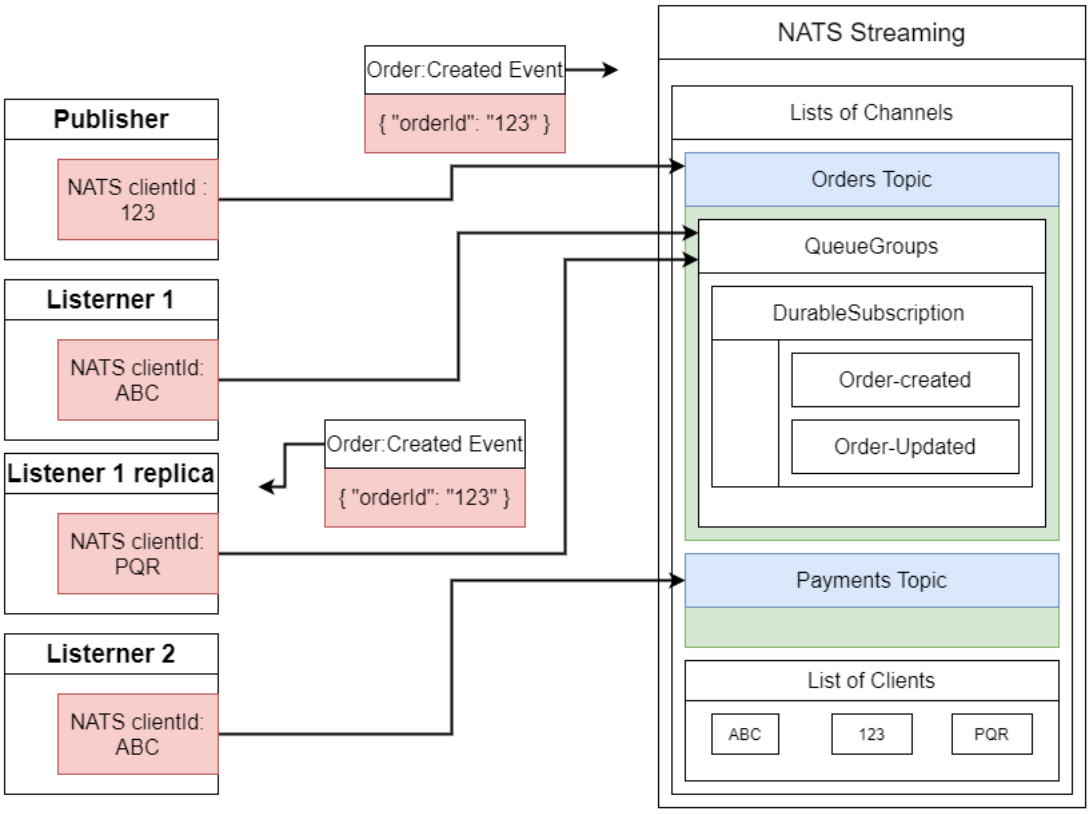

Figure 4.9: NATS Streaming Server

4.3.3 NATS Configurations

NATS ensures at least one-time delivery. NATS Streaming offers message acknowledgements between publisher and server (for publish operations) and between subscriber and server (to confirm message delivery). Messages are persisted by the server in memory or secondary storage (or other external storage) and will be redelivered to eligible subscribing clients as needed. This means that for the very first time NATS will send all the messages to a service in a sequence as they arrived. If a service fails to process that message NATS will keep sending all this message untill it gets acknowledgement from the service.

If a subscription goes down NATS server holds on to that event and wait for a specific period. If the same subscription comes alive it will pass on the event. So for a brief period channel holds one extra subscription within its list of subscriptions. That's why the message is temporarily lost.

We can have some additional configurations to NATS Streaming Server to let it know when to let go of the dead subscription.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nats-depl
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nats
  template:
    metadata:
      labels:
        app: nats
    spec:
      containers:
        - name: nats
          image: nats-streaming:0.17.0
          args:
            [
              '-p',
              '4222',
              '-m',
              '8222',
              '-hbi',
              '5s',
              '-hbt',
              '5s',
              '-hbf',
              '2',
              '-cid',
              'krishisetu',
            ]
---
```

Figure 4.10: NATS Streaming Server

Arguments

-p:4222 - Port used by clients to connect to NATS Server
-m:8222 - Port used for http monitoring
-hbi = 5s - Interval at which server sends heartbeat to a client
-hbt = 5s - How long server waits for a heartbeat response
-hbf = 2 - Number of failed heartbeats before server closes the client connection
-cid = krishisetu - Cluster ID

4.4 Code Sharing

While developing micro-services there is a some common logical code that needs to be shared across different services for example every service that needs user authentication in order to access its services need to have same common code for authentication.

We could simply copy paste code from one service to another. Downside to this approach is as the platform grows there will be different versions of code in different services. This could be a potential problem as event data flowing in our application have to be consistent across the application. To solve this we need to have a central repository from which we can pull the latest code into our different services.

In Krishi Setu we are using npm public registory.

4.4.1 npm registory

To resolve packages by name and version, npm talks to a registry website that implements the CommonJS Package Registry specification for reading package info. npm is configured to use the npm public registry at https://registry.npmjs.org by default. Use of the npm public registry is subject to terms of use available at https://www.npmjs.com/policies/terms. You can configure npm to use any compatible registry you like, and even run your own registry. Use of someone else registry may be governed by their terms of use. npm's package registry implementation supports several write APIs as well, to allow for publishing packages and managing user account information.

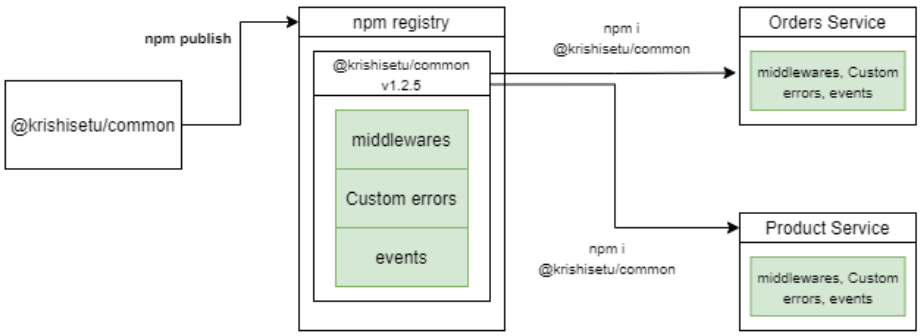

Figure 4.11: Code Sharing Approach

4.5 Cross-Service Data Replication

Every micro service is inside an isolated environment. Product service is responsible for creating a product. Orders service is responsible for creating an order. We can make a synchronous request to products service, but this approach will have all the downsides of synchronous communication. The second approach would be we can replicate the product in the orders database. So being in an isolated environment we need to make orders-service aware every time a product is created. This is where NATS comes into the picture. Whenever a product is created product service will publish a message containing that product to the product:created topic. Orders Service listeners will consume that message (product) and replicate that product inside orders-mongo-depl. This way we can hold up the idea of loosely coupled micro services.

Followings are the diagrams shows different micro services affected by different events.

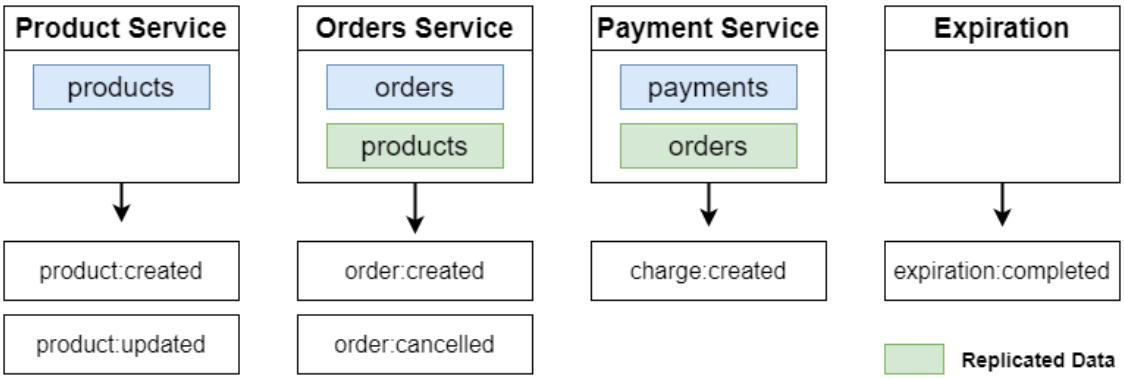

Figure 4.12: Events published by each service

product:created - Orders needs to know the valid products that can be purchased and the price of each product.

product:updated - Orders service needs to know when the price of a product has changed and a product has successfully been reserved.

order:created - Products service needs to be told that one of its product has been reserved, and no further edits to that product should be allowed, Payments service needs to know there is a new order that a user might submit a payment for and Expiration service needs to start a 5 minute timer to eventually time out this order.

order:cancelled - Product service should unreserve a product if the corresponding order has been cancelled so this product can be edited again. Payments should know that any incoming payments for this order should be rejected.

charge:created - Orders service needs to know that an order has been paid for.

expiration:completed - Orders service needs to know that an order has gone over the 5 minute time limit. It is up to the orders service to decide whether or not to cancel the order (it might have already been paid!!!)

4.6 Version Control

Krishi Setu uses Github for version control.

We are using Github Actions to automate testing. Automated tests get triggered every time we create a new pull request. After the testing is done we get the results of all the passed and failed test, if any test is a failing test we get a warning before we merge the code base with the master branch.

```
Shreyaschorge, 2 months ago • Create tests-orders.yml
name: tests-orders

on:
  pull_request:
    paths:
      - 'orders/**'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: cd orders && npm install && npm run test:ci
```

Figure 4.13: Github Actions for Orders Service

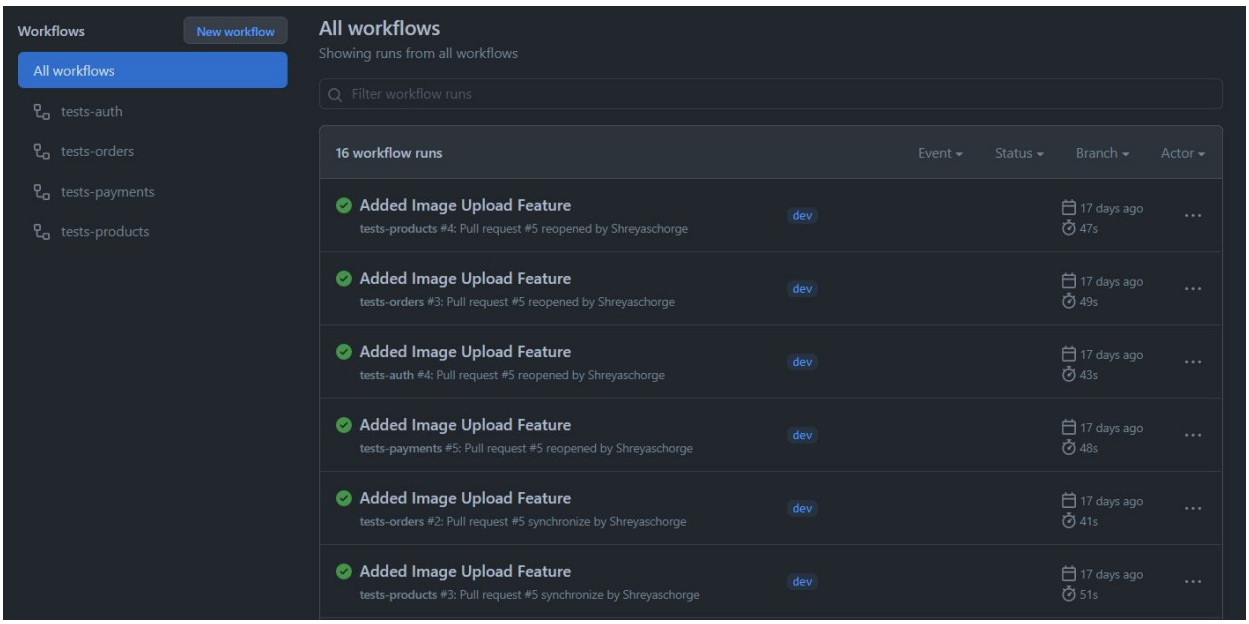

Figure 4.14: Github Actions Workflows

4.7 API endpoints

4.7.1 Authentication Service Endpoints

Auth Service		
Endpoints	Method	Purpose
/api/users/signup	POST	Signup user to the application
/api/users/signin	POST	Sign user into the application
/api/users/signout	GET	Signout user from the application
/api/users/currentuser	GET	Get the current user

Figure 4.15: Authentication Service Endpoints

4.7.2 Product Service Endpoints

Products Service		
Endpoints	Method	Purpose
/api/products	GET	Fetch all products
/api/products/:id	GET	Fetch single product
/api/products	POST	Create a new Product
/api/product/:id	PUT	Edit Product
/api/products/upload	GET	Get Pre-Signed URL from S3
/api/products/user-products	GET	Fetch products of a single user

Figure 4.16: Product Service Endpoints

4.7.3 Orders Service Endpoints

Orders Service		
Endpoints	Method	Purpose
/api/orders	GET	Fetch all orders of a single user
/api/orders	POST	Create an order
/api/orders/:orderId	GET	Fetch single order
/api/orders/:orderId	DELETE	Delete Order

Figure 4.17: Orders Service Endpoints

4.7.4 Payments service Endpoint

Payments Service		
Endpoints	Method	Purpose
/api/payments	POST	Make a payment

Figure 4.18: Payments service Endpoint

Chapter 5

Testing

5.1 Scope of Testing

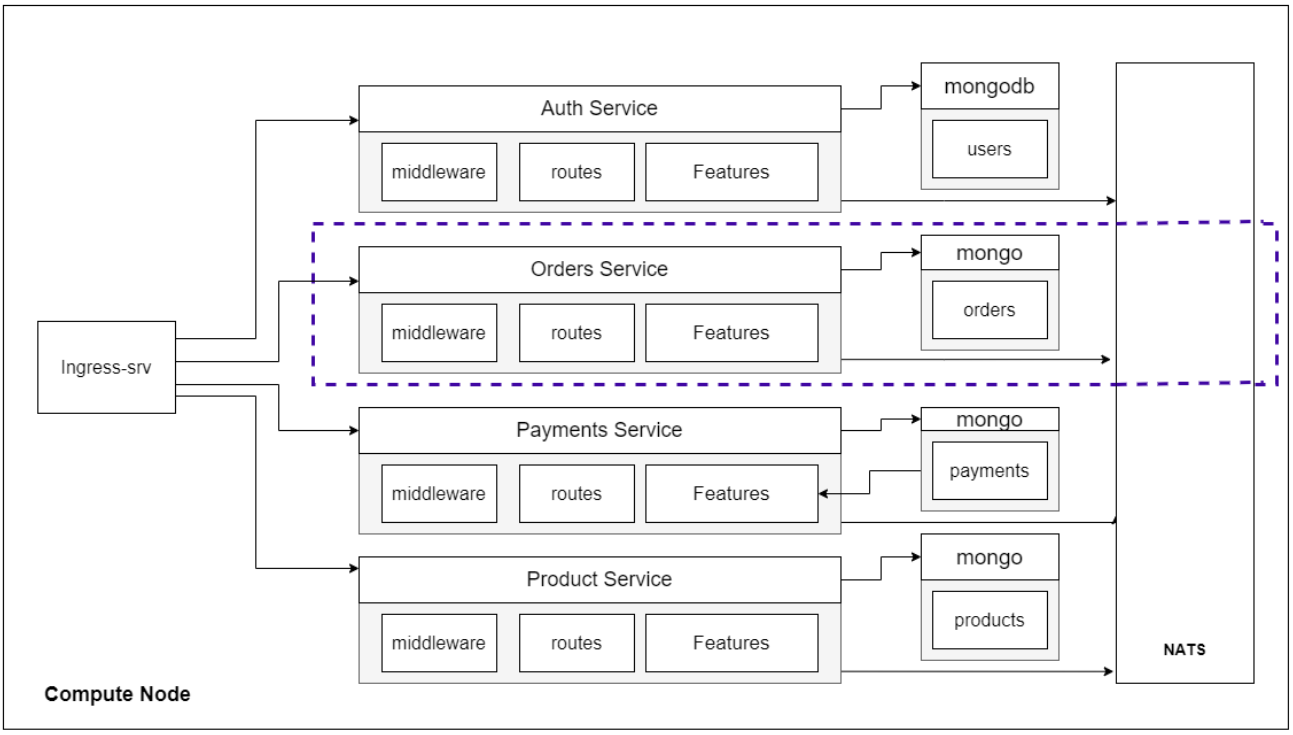

Figure 5.1: Testing Scope

Krishisetu focuses on integration testing throughout the application.

5.1.1 Testing Results

A. Products Service

```
Run cd products && npm install && npm run test:ci
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1,
but package-lock.json was generated for lockfileVersion@2. I'll try to do my best
```

```
with it!

> core-js@3.9.1 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/products/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"


> mongodb-memory-server@6.9.3 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/products/node_modules/mongodb-memory-server
> node ./postinstall.js

mongodb-memory-server: checking MongoDB binaries cache...
Downloading MongoDB 4.0.14: 0 % (0mb / 99.4mb)
mongodb-memory-server: binary path is /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/products/node_modules/.cache/mongodb-memory-server
/mongodb-binaries/4.0.14/mongod
npm WARN products@1.0.0 No description
npm WARN products@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.3.2
(node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for
fsevents@2.3.2: wanted {"os":"darwin","arch":"any"}
(current: {"os":"linux","arch":"x64"})

added 773 packages from 660 contributors and audited 784 packages in 14.693s

31 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities


> products@1.0.0 test:ci /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/products
> jest

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
(Use `node --trace-deprecation ...` to show where the warning was created)
PASS src/routes/__test__/update.test.ts (9.569s)
   Console

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25
```

```
    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:updated

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:updated

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
PASS src/routes/__test__/new.test.ts
   Console

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
PASS src/events/listeners/__test__/order-created-listener.test.ts
   Console

    console.log
      Event published to subject product:updated
```

```
      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:updated

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:updated

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
PASS src/events/listeners/__test__/order-cancelled-listener.test.ts
  Console

    console.log
      Event published to subject product:updated

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
PASS src/routes/__test__/show.test.ts
  Console

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

(node:1712) DeprecationWarning: Listening to events on the Db class has been
deprecated and will be removed in the next major version.
PASS src/routes/__test__/index.test.ts
  Console

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25
```

```
    console.log
      Event published to subject product:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25


Test Suites: 6 passed, 6 total
Tests:       22 passed, 22 total
Snapshots:   0 total
Time:        14.153s
Ran all test suites.
```

B. Authentication Service

```
Run cd auth && npm install && npm run test:ci
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1,
but package-lock.json was generated for lockfileVersion@2. I'll try to do my
best with it!

> mongodb-memory-server@6.9.6 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/auth/node_modules/mongodb-memory-server
> node ./postinstall.js

mongodb-memory-server: checking MongoDB binaries cache...
Downloading MongoDB 4.0.14: 0 % (0mb / 99.4mb)
mongodb-memory-server: binary path is /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/auth/node_modules/.cache/mongodb-memory-server
/mongodb-binaries/4.0.14/mongod
npm WARN authentication-service@1.0.0 No description
npm WARN authentication-service@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY:
fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform
for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"}
(current: {"os":"linux","arch":"x64"})

added 748 packages from 600 contributors and audited 758 packages in 15.469s

30 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities


> authentication-service@1.0.0 test:ci /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/auth
> jest
```

```
PASS src/routes/__test__/signup.test.ts (6.441s)
PASS src/routes/__test__/signin.test.ts
PASS src/routes/__test__/current-user.test.ts
PASS src/routes/__test__/signout.test.ts

Test Suites: 4 passed, 4 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        8.957s
Ran all test suites.
```

C. Orders Service

```
    Run cd orders && npm install && npm run test:ci
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1,
but package-lock.json was generated for lockfileVersion@2. I'll try to do my
best with it!

> core-js@3.9.1 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/orders/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"


> mongodb-memory-server@6.9.6 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/orders/node_modules/mongodb-memory-server
> node ./postinstall.js

mongodb-memory-server: checking MongoDB binaries cache...
Downloading MongoDB 4.0.14: 0 % (0mb / 99.4mb)
mongodb-memory-server: binary path is /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/orders/node_modules/.cache/mongodb-memory-server
/mongodb-binaries/4.0.14/mongod
npm WARN products@1.0.0 No description
npm WARN products@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY:
fsevents@2.3.2 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY:
Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"}
(current: {"os":"linux","arch":"x64"})

added 758 packages from 601 contributors and audited 769 packages in 16.475s

31 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
> products@1.0.0 test:ci /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/orders
> jest

PASS src/events/listeners/__test__/product-updated-listener.test.ts (6.404s)
PASS src/routes/__test__/new.test.ts
   Console

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

PASS src/events/listeners/__test__/expiration-complete-listener.test.ts
   Console

    console.log
      Event published to subject order:cancelled

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:cancelled

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:cancelled

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

PASS src/routes/__test__/delete.test.ts
   Console

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:cancelled
```

```
      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:cancelled

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

PASS src/routes/__test__/index.test.ts
   Console

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

PASS src/events/listeners/__test__/product-created-listener.test.ts
PASS src/routes/__test__/show.test.ts
   Console

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

    console.log
      Event published to subject order:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25


Test Suites: 7 passed, 7 total
```

```
Tests:       17 passed, 17 total
Snapshots:   0 total
Time:        12.468s
Ran all test suites.
```

D. Payments Service

```
    Run cd payments && npm install && npm run test:ci
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@1,
but package-lock.json was generated for lockfileVersion@2. I'll try to do my
best with it!

> core-js@3.6.5 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/payments/node_modules/core-js
> node -e "try{require('./postinstall')}catch(e){}"


> mongodb-memory-server@6.5.2 postinstall /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/payments/node_modules/mongodb-memory-server
> node ./postinstall.js

mongodb-memory-server: checking MongoDB binaries cache...
Downloading MongoDB 4.0.14: 0 % (0mb / 99.4mb)
mongodb-memory-server: binary path is /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/payments/node_modules/.cache/mongodb-memory-server
/mongodb-binaries/4.0.14/mongod
npm WARN payments@1.0.0 No description
npm WARN payments@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY:
fsevents@2.1.3 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY:
Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"}
(current: {"os":"linux","arch":"x64"})

added 748 packages from 604 contributors and audited 757 packages in 18.123s

19 packages are looking for funding
  run `npm fund` for details

found 23 high severity vulnerabilities
  run `npm audit fix` to fix them, or `npm audit` for details

> payments@1.0.0 test:ci /home/runner/work/krishisetu-backend-api
/krishisetu-backend-api/payments
> jest

PASS src/routes/__test__/new.test.ts (8.445s)
   Console
```

```
    console.log
      price 99811

      at src/routes/__test__/new.test.ts:73:11

    console.log
      Event published to subject payment:created

      at node_modules/@krishisetu/common/build/events/base-publisher.js:15:25

PASS src/events/listeners/__test__/order-cancelled-listener.test.ts
PASS src/events/listeners/__test__/order-created-listener.test.ts

Test Suites: 3 passed, 3 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        10.343s
Ran all test suites.
```

Chapter 6

Result

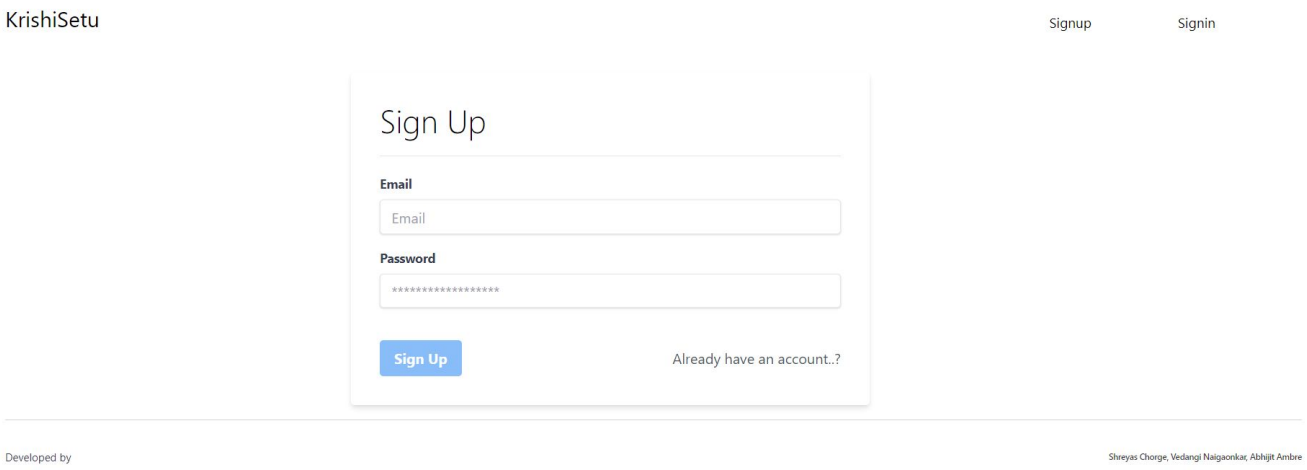

Figure 6.1: SignUp Page

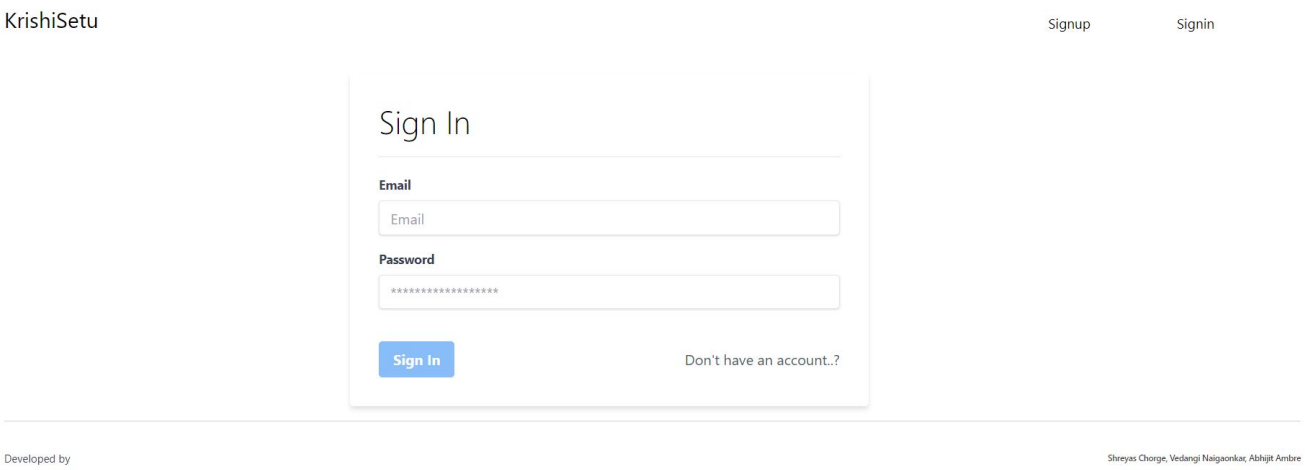

Figure 6.2: SignUp Page

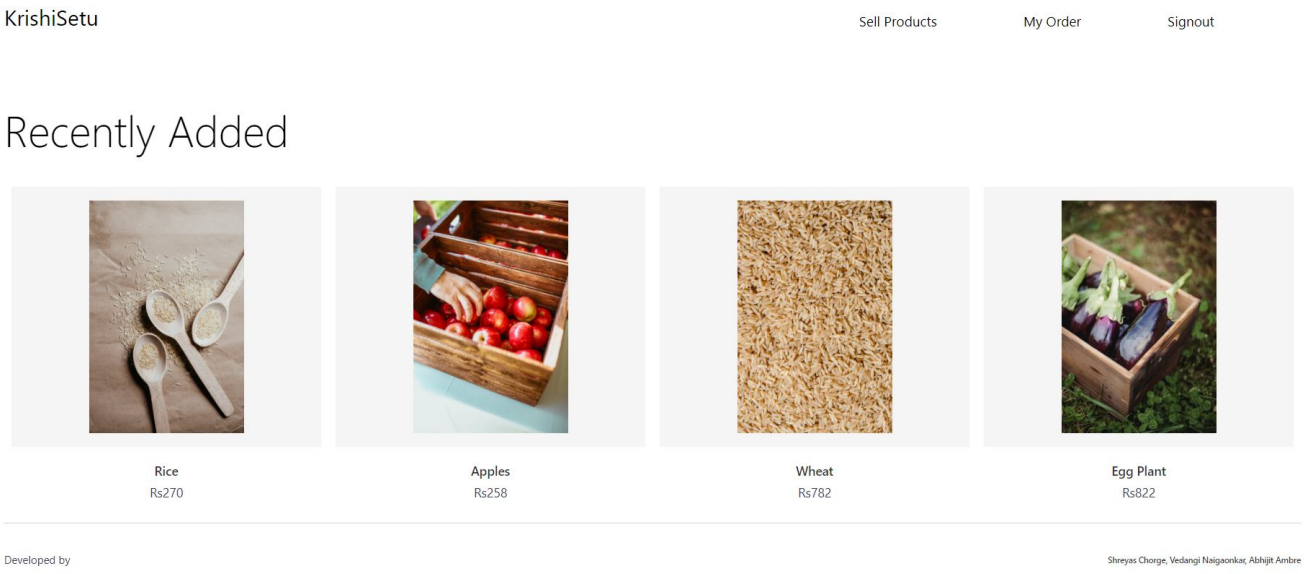

Figure 6.3: Home Page

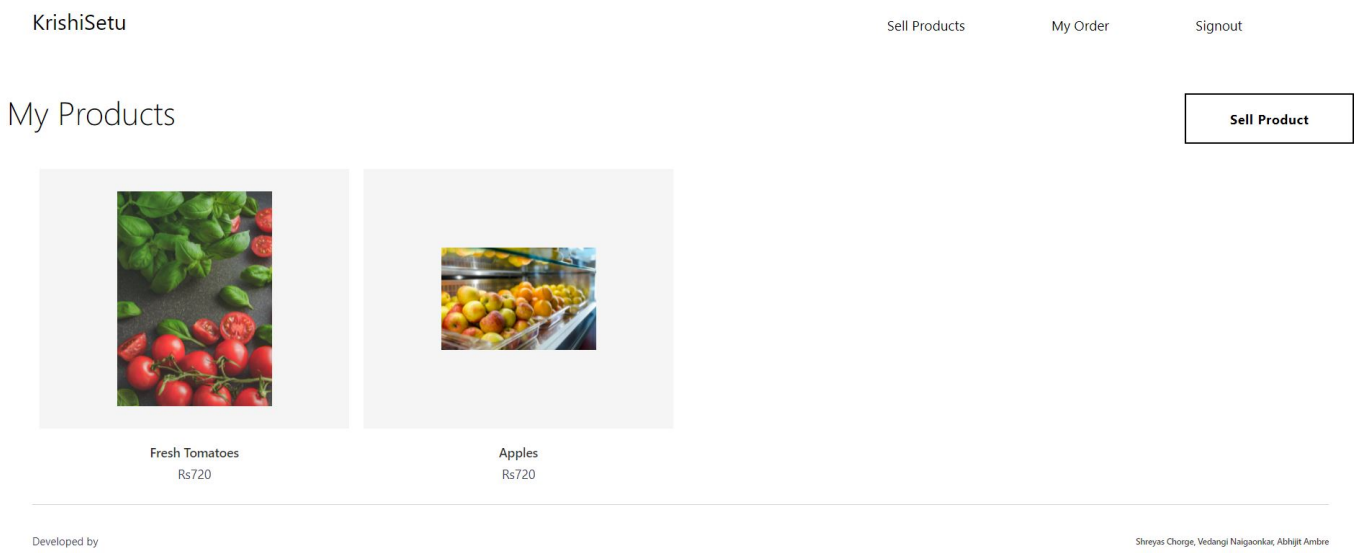

Figure 6.4: Sell Products Page

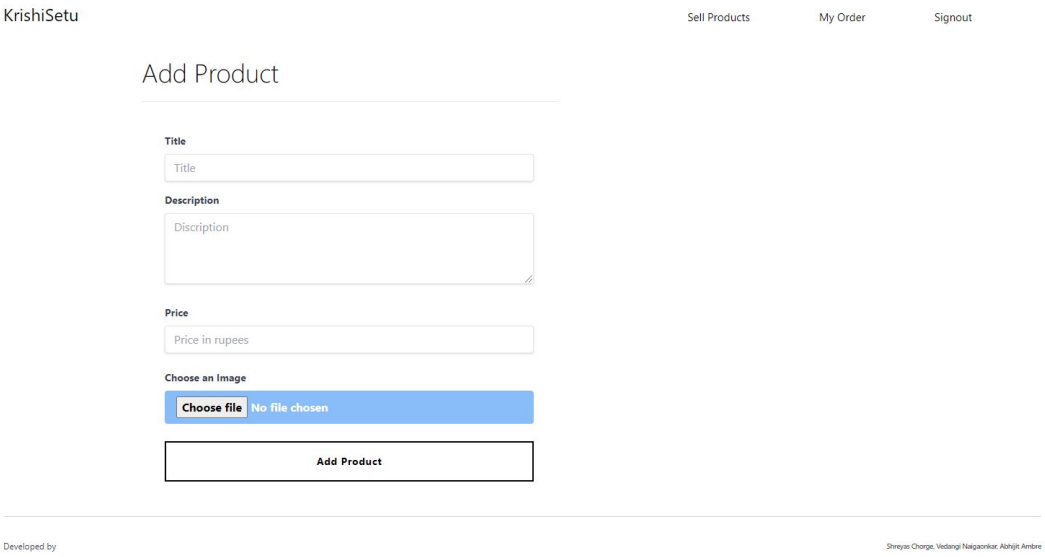

Figure 6.5: Add Products Page

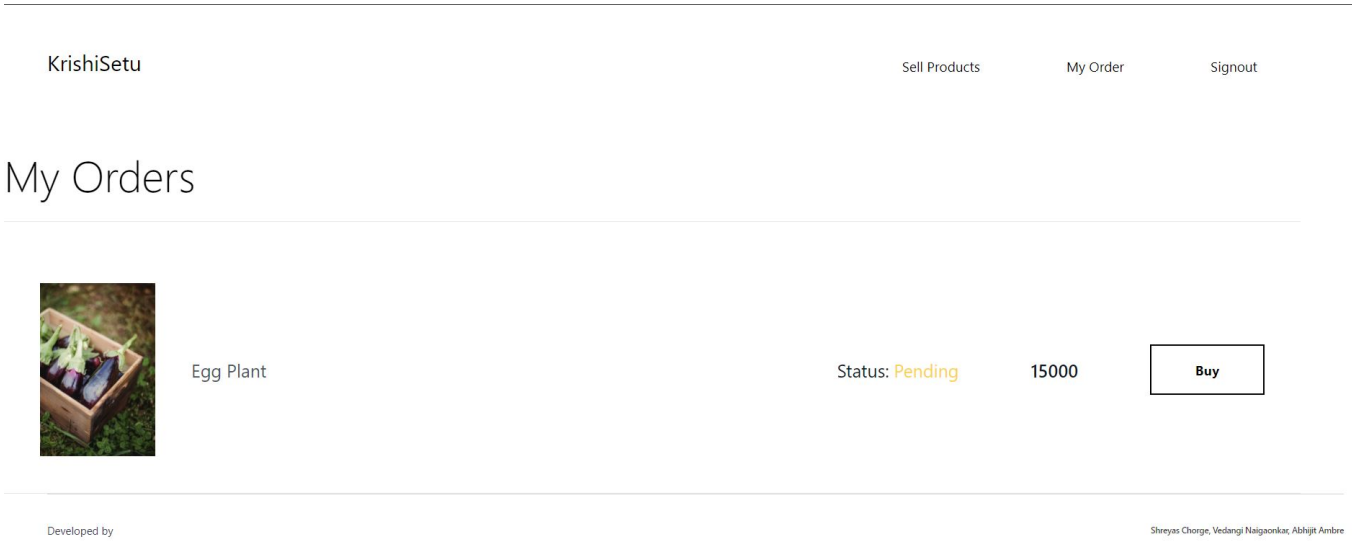

Figure 6.6: Orders Page

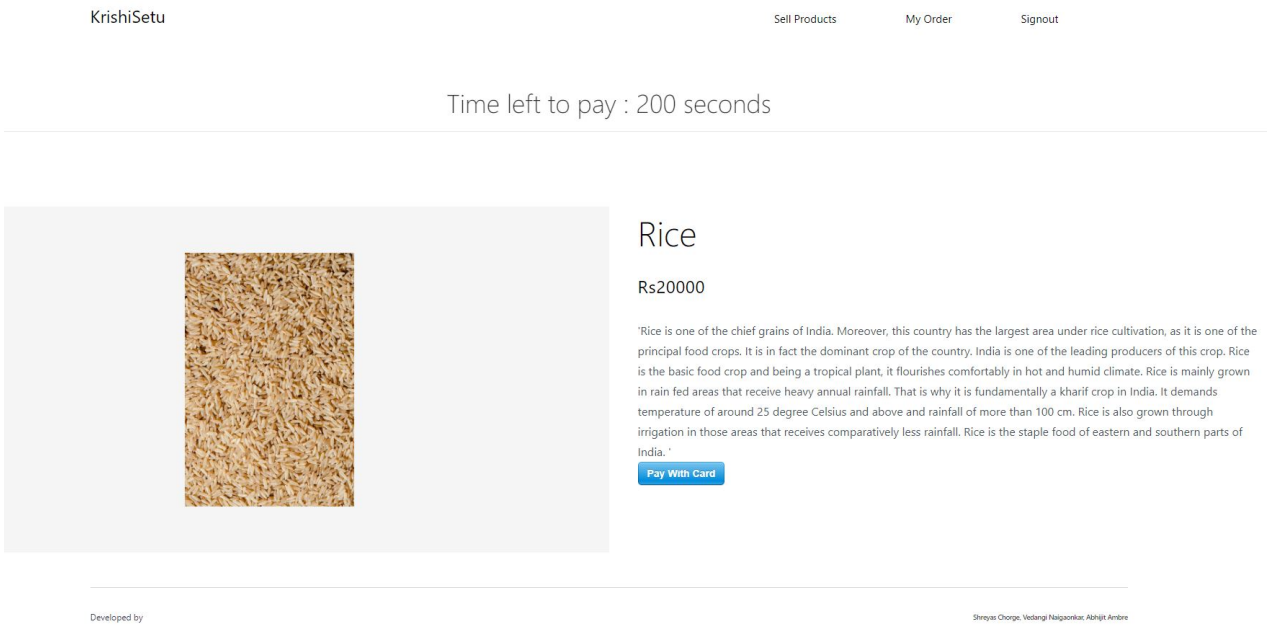

Figure 6.7: Payments Page

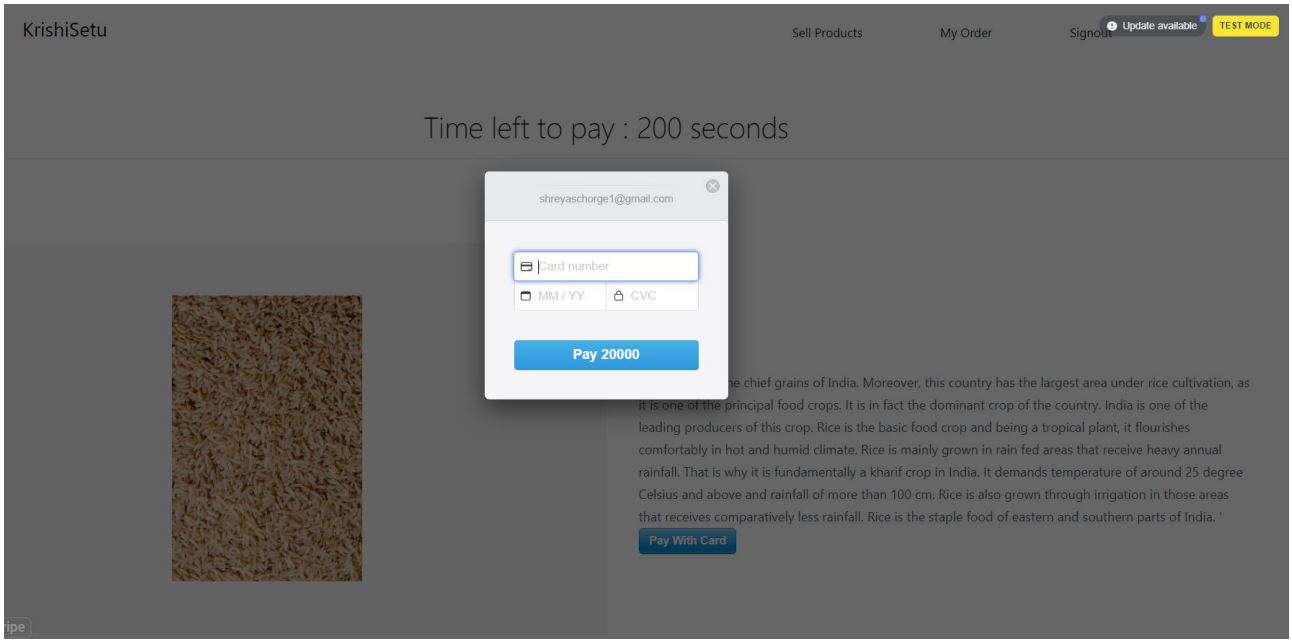

Figure 6.8: Checkout

Chapter 7

Conclusions and Future Scope

The main objective to build this application is to eliminate all the middlemen from the chain of distribution cycle by providing them with the platform on which they can sell their goods directly to the local distributors of major cities thus giving them the value for their goods.

Also making sure that the platform could maintain high traffic and scale up and scale down whenever necessary and it is always available to serve its users.

Also the entire application is leveraging Open Source technologies. And the application is Open Source thus encouraging the developers community to contribute to the growth of the application and making it more reliable to its users.

For the future, we could add more services like a recommendation system, maps, comments, integration of continuous monitoring plugin like Prometheus, Sharded Clusters and replica sets for high availability of the data.

Bibliography

[1] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, Ferhat Khende, "Kubernetes as an Availability Manager for Microservice Applications", unpublished.

[2] L Magnon "Modern Messaging for Distributed Systems", Journal of Physics Conference Series 608(1):012038

[3] Poojya J Bhat, Priya D, "Modern Messaging Queues - RabbitMQ, NATS and NATS Streaming", International Journal of Recent Technology and Engineering (IJRTE)

Appendices

Appendix-A: Installation Guide

1. Install Docker

2. Enable Kubernetes

3. Install Skaffold

4. Setup an aws bucket, change bucket configurations in uploadImage route in Products-service

5. Setup a stripe accout

6. git clone https://github.com/Shreyaschorge/krishisetu-backend-api.git

7. cd krishisetu-backend-api

8. Install node_modules in respective folders

9. Using kubectl, create generic secrets for JWT_KEY, STRIPE_KEY, AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY

10. skaffold dev

Publication

Paper entitled **"Krishi Setu: Connecting Farmers and Consumers"** is presented at **"ICIRCA : 3rd IEEE International Conference on Inventive Research in Computing Applications"** by **"Shreyas Chorge", "Vedangi Naigaonkar", "Abhijit Ambre".**