# International Institute of Information Technology, Bangalore

## (IIIT Bangalore)

### Software Production Engineering(CS 816)

### Project Report

### Couch Potato- Online Grocery Ordering !

**Under the Guidance of Prof. B. Thangaraju**

**Teaching Assistant: Aman Gupta**


Ram Rodwal MT2022087

Vartika Chaturvedi MT2022130

**Table of contents**

# Abstract

The rise of e-commerce and the convenience it offers to customers. More and more people are turning to online grocery ordering to save time and avoid the hassle of visiting physical stores. This trend has been accelerated by the COVID-19 pandemic, which has led to an increased demand for contactless delivery and online shopping.

Online grocery ordering allows customers to browse products and place orders from the comfort of their homes. They can choose from a wide variety of products and brands, compare prices, and read reviews before making a purchase. Online grocery stores often offer promotions, discounts, and loyalty programs to attract and retain customers.

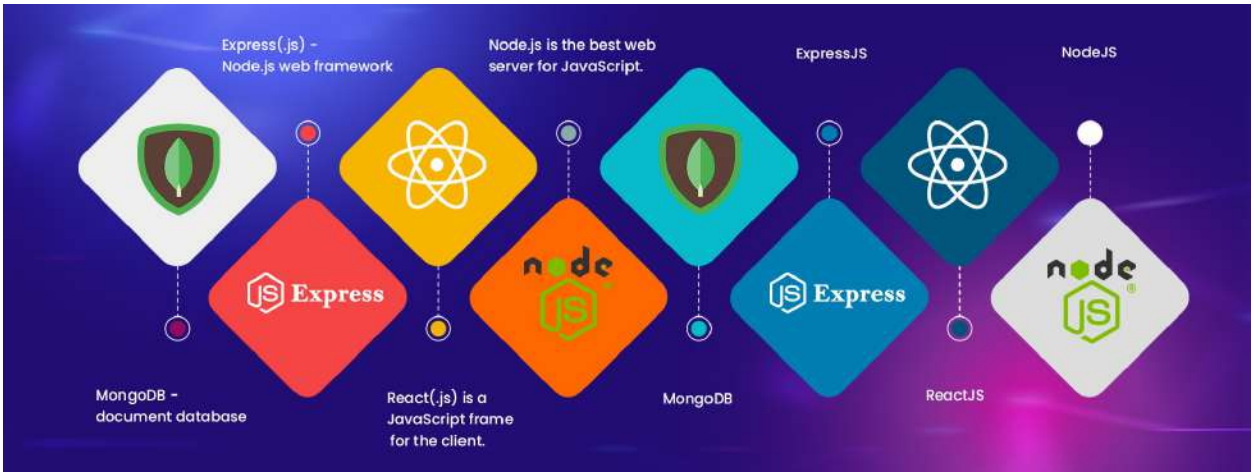

**Couch Potato** is such a grocery ordering website wherein the users can order the groceries amongst an enormous amount of options. The further steps have also been automated using DevOps.

## Introduction

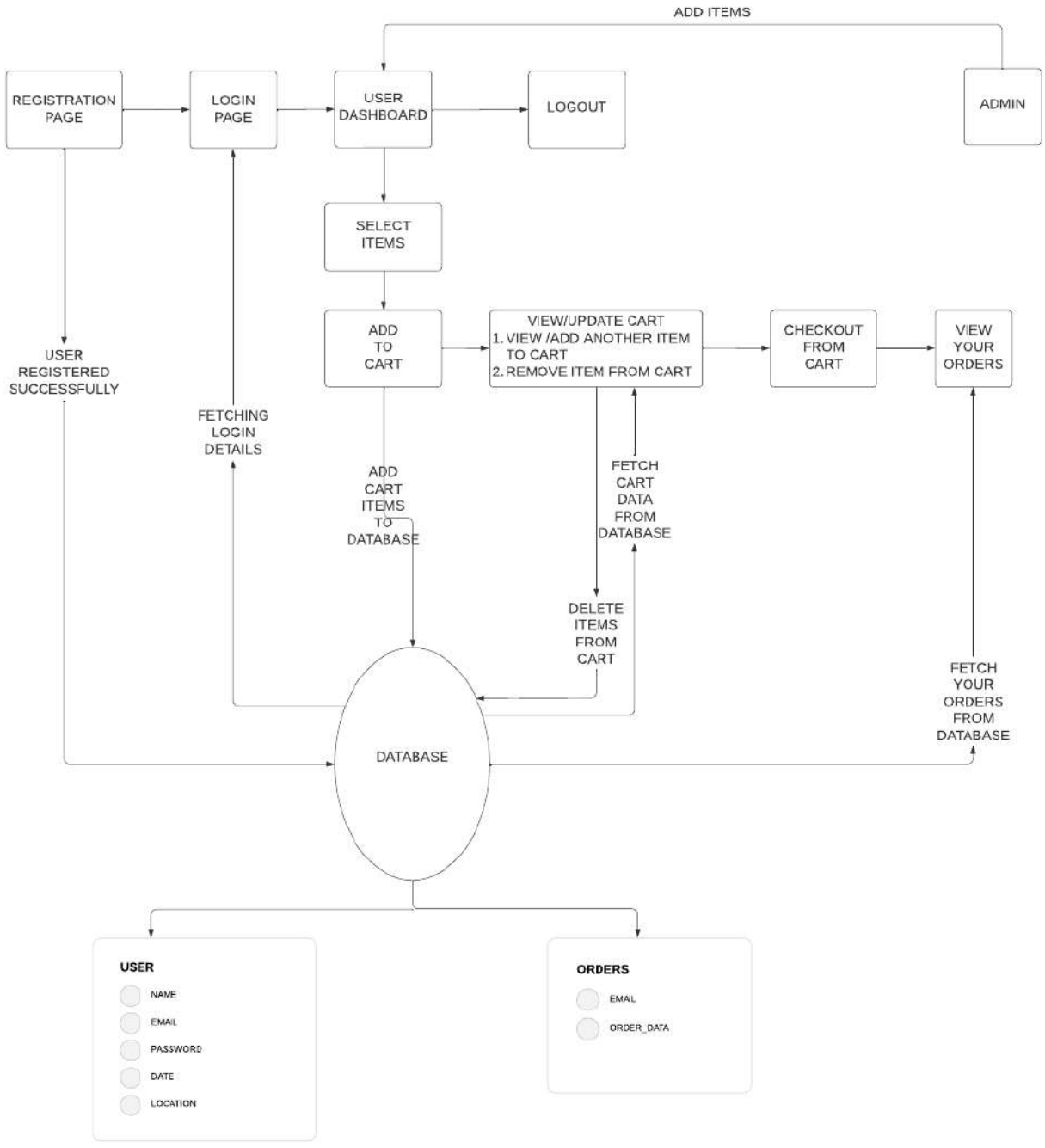
Couch Potato is an online e-commerce website that allows users to order groceries online and get it delivered to their address.

The groceries are divided into different categories so that it becomes easier for the user.

TechStack used → MERN (MongoDB, ExpressJS, ReactJS, NodeJS)



## Architecture Diagram



## Features

1. **Login/SignUp**
    a. Account creation and login for users.

2. **Home Page**
    a. Shows all the varieties of grocery

3. **Search Groceries**
    a. User can search the groceries amongst the plethora of options.

4. **Adding/Viewing the groceries to cart**
    a. User can add their choice of grocery and also the amount of groceries to be added can be chose (the quantity of groceries)
    b. The user can view their cart where all the selected options have been saved and then can check out to place the order.
    c. The final pricing is also calculated at the end, before the check out.
    d. The user can delete the goods that they do not wish to order anymore but have added in their cart.
    e. The price gets adjusted based on the goods that are currently present in their cart.

5. **View Orders**
    a. The user can see the orders that they have placed till now .
    b. The orders are arranged based on the date when they were placed.

# Software Development Life Cycle

# Installations

## React JS

React JS is a popular JavaScript library used for building user interfaces. It was developed by Facebook and is widely used in web development. React allows developers to create reusable UI components and manage the state of these components in an efficient way. It is also known for its virtual DOM, which allows for fast rendering of updates to the UI.

To use React, developers must install it as a dependency in their project and then import the necessary modules. They can then create components using JSX syntax and render them to the DOM using ReactDOM.

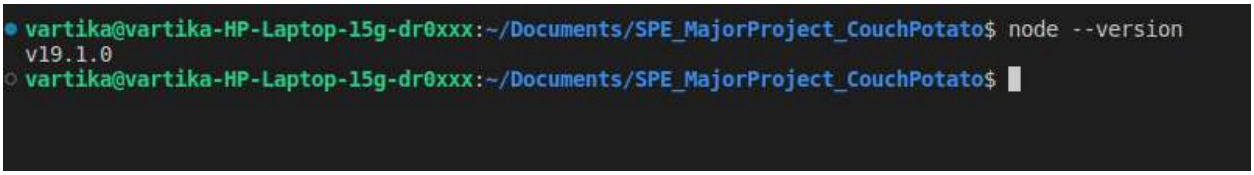
Overall, React is a powerful tool for building complex and dynamic user interfaces that are easy to maintain and update over time.

Update local before installing:

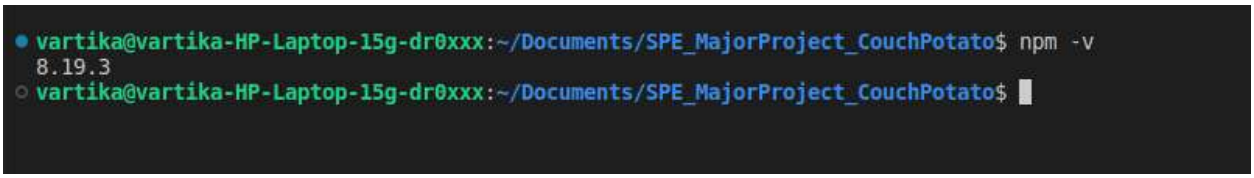
```
sudo apt-get update
```

Install NodeJS and NPM:

```
sudo apt-get install nodejs
```




```
sudo apt-get install npm
```

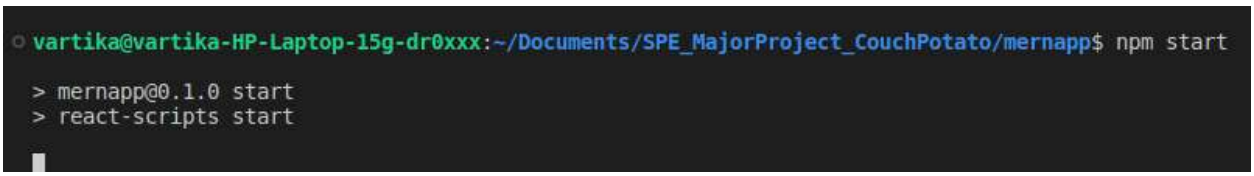



React App

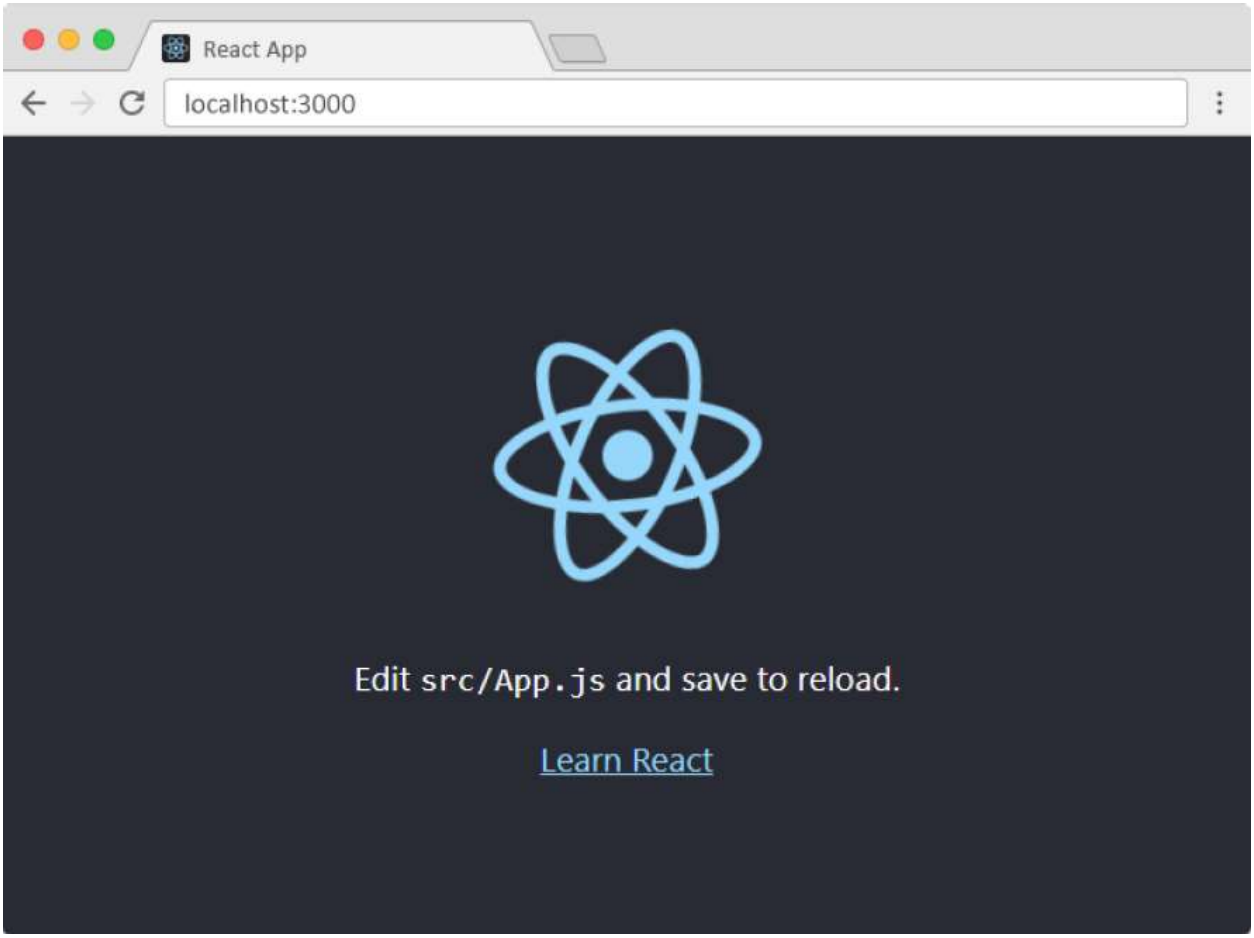
```
npx create-react-app mernapp
```

Now check whether your react app is created or not

```
npm start
```




Now go to **localhost:3000** and you will see the react app running




## Express JS

Express JS is a popular web framework for Node.js that simplifies the process of building web applications. It provides a set of tools and features for handling HTTP

requests and responses, routing, middleware, and more. Express allows developers to create APIs and web services quickly and easily, and is widely used in the development of web applications.

To use Express, developers must install it as a dependency in their project and then import the necessary modules. They can then define routes and middleware functions using the Express API, and use the built-in functionality to handle HTTP requests and responses.

Overall, Express is a powerful tool for building scalable and efficient web applications using Node.js.

Express App

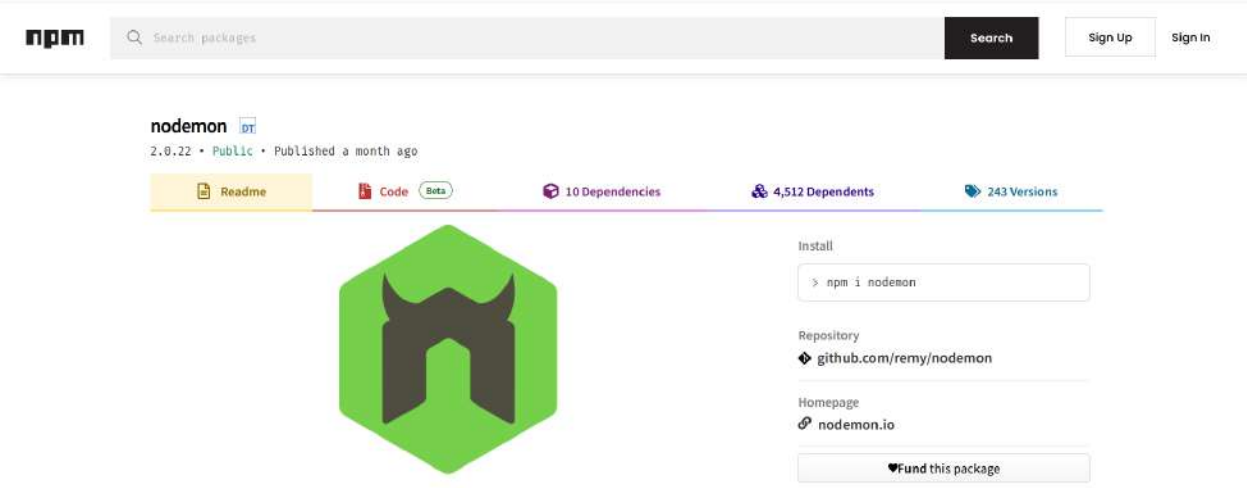
```
npm init -y
```

-y automates the config with node default values i.e without going through the interactive process.

Running the Node Application locally:

```
node index.js
```

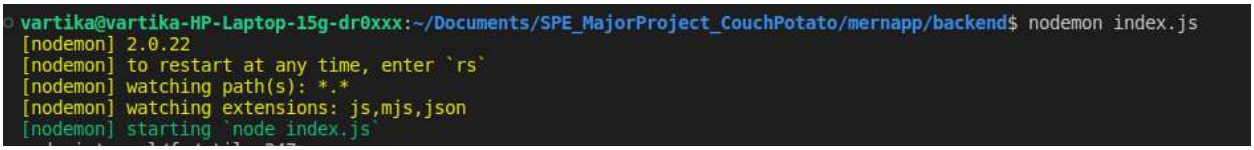
Running the Node Application locally with nodemon:




Nodemon automatically restarts the node application whenever a file change in the directory is detected.

`npm install nodemon`

`nodemon index.js`




# Source Control Management (SCM)

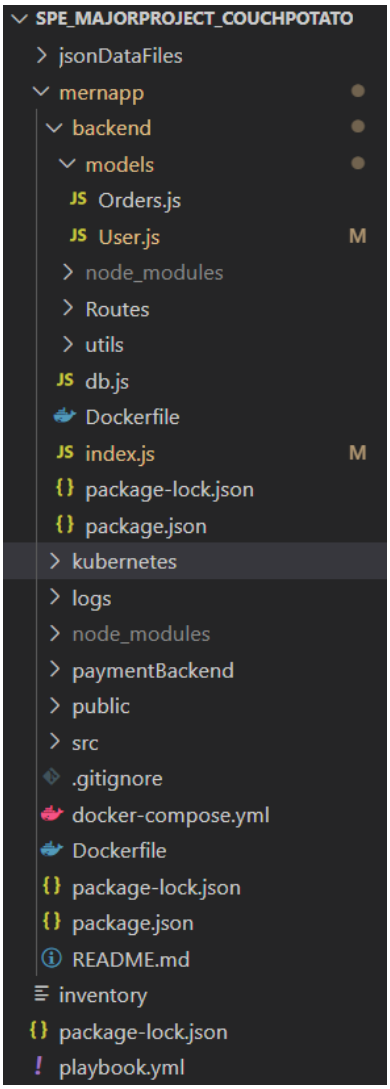
The code for this project has been stored in a GitHub repository. The repository will be organized into branches, and changes will be made using pull requests. Code reviews will be required for all pull requests, and the master branch will be protected to prevent direct commits.

The repository will be linked to a continuous integration and delivery (CI/CD) pipeline, which will automatically build and test the code on every change. The pipeline will also deploy the code to a staging environment for further testing, and then to a production environment once it passes all tests.

Repository Link: https://github.com/ramrodwal/SPE_MajorProject_CouchPotato.git

The frontend is created in the mernapp/ directory and the backend is created in the mernapp/backend/ directory

**Code Structure:**




Initializing the project with git:

```
git init
```

```
git remote add origin https://github.com/ramrodwal/SPE_MajorProject_CouchPotato.git
```

Workflow:

```
git add <files>
```

```
git commit -m "commit message"
```

```
git pull origin master
```

```
git push origin master
```

The code is first pulled before making a push to make sure that our project is in the

latest stage and to avoid merge conflicts. For the above, the pull and push is done on the “master” branch.

Working on a feature/issue:

```
git checkout master
```

```
git checkout -b "<your_branch_name>"
```

After creating a branch, required changes are done and a pull request to the main is created.

```
git add <files>
```

```
git commit -m "commit message"
```

```
git pull origin master
```

```
git push origin master
```

Then merge the pull request from Github.

## Containerization with Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, the developer can be sure that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon.

To containerize an application using docker, you must write a Dockerfile that specifies the application's environment and dependencies. The Dockerfile is then used to build a Docker image, which can be used to run Docker containers.

**Frontend Dockerfile:**

```
# Use node's alpine variant to save on space and version number is 16.19.1
FROM node:16.19.1-alpine

# Set the working directory to /
WORKDIR /usr/src/app

COPY package*.json ./

# Install any needed packages
RUN npm install

# Copy the current directory contents into the container at /
COPY . .

# Make port 3000 available to the world outside this container
EXPOSE 3000

# Run index.js when the container launches
CMD ["npm", "start"]
```

Once the Dockerfile has been created, it can be used to build a Docker image:

```
docker build -t vartikach02/client .
```

This command builds a Docker image with the tag `vartikach02/client` using the Dockerfile in the current directory (`.`).

To run a Docker container from the image, use the following command:

```
docker run -p 3000:3000 vartikach02/client
```

This command runs a Docker container from the `vartikach02/client` image, mapping port 3000 on the host to port 3000 in the container.

**Backend Dockerfile:**

```
# This line specifies the base image to use for the Docker image.
# In this case, it is using the official Node.js image with the version
# 16.19.1-alpine,
# which is a lightweight version of the image that uses Alpine Linux as the
# base operating system.
```

```
FROM node:16.19.1-alpine

# This line sets the working directory for the rest of the commands in the
# Dockerfile to /usr/src/app.
WORKDIR /usr/src/app

# This line copies the package.json and package-lock.json files from the local directory
# to the Docker image.
COPY package*.json ./

# This line runs the npm install command in the Docker image to install the dependencies
# listed in the package.json file.
RUN npm install

RUN mkdir logs && chown -R node:node logs

# This line installs the cors package, which is a middleware for enabling Cross-Origin
# Resource Sharing (CORS) in the Node.js application.
RUN npm install cors

# This line copies all the files in the local directory to the Docker image.
COPY . .

EXPOSE 5000

# This line specifies the command to run when the Docker container is started.
#  In this case, it is running the npm start command to start the Node.js application.
CMD ["npm", "start"]
```

Once the Dockerfile has been created, it can be used to build a Docker image:

```
docker build -t vartikach02/server .
```

This command builds a Docker image with the tag `vartikach02/server` using the Dockerfile in the current directory (`.`).

To run a Docker container from the image, use the following command:

```
docker run -p 5000:5000 vartikach02/server
```

This command runs a Docker container from the `vartikach02/server` image, mapping port 5000 on the host to port 5000 in the container

# Docker-Compose

Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define their application's services, networks, and volumes in a single file, and then spin up all the services with a single command.

A `docker-compose.yml` file can be used to specify the different services that make up an application, as well as any required configuration options. Docker Compose can then be used to start, stop, and manage the application's containers.

In the case of Couch Potato, we can use Docker Compose to define the frontend and backend services, as well as any required databases or other services. This will make it easier to manage the application's infrastructure and ensure that it is consistent across different environments.

Here is an example `docker-compose.yml` file that defines the frontend and backend services for Couch Potato:

```
# This specifies the version of the Docker Compose file format that the file uses.
version: '3'

# This section defines the Docker services that will be created.
services:

  # This is the name of the first service, which will be a backend API server.
  api-server:
  # This specifies that the Dockerfile for this
  # service is located in the ./backend directory,
  # and Docker should build the image for the service
  #from that Dockerfile.
    build:
      context: /home/vartika/Documents/SPE_MajorProject_CouchPotato/mernapp/backend
      dockerfile: Dockerfile
    # This line maps the port 5000 on the container
    #to the port 5000 on the host machine, which
    #allows incoming requests to the API server.
    image: vartikach02/server:latest
    ports:
      - "5000:5000"
      # This specifies that the api-server service
      # will be connected to the mern-app network,
      #  which will allow it to communicate with
      #other services on that network.
    volumes:
      - ./logs:/usr/src/app/logs
    networks:
      - mern-app
    # This sets the name of the container running
    #the api-server service to node-app.
    container_name: server
```

```
    # This command will be executed when the
    #container starts, which will start the
    #API server using the npm start command.
    command: npm start
  # This is the name of the second service,
  # which will be a frontend React application.
  react-app:
  # This specifies that the react-app service
  #will use an existing image named temp,
  #which should already be built and available.
    build:
      context: /home/vartika/Documents/SPE_MajorProject_CouchPotato/mernapp
      dockerfile: Dockerfile
    image: vartikach02/client:latest
    # This sets stdin_open to true, which means
    # that the container will keep stdin open
    #even if not attached. This can be useful for debugging.
    stdin_open: true
    ports:
      - "3000:3000"
    networks:
      - mern-app
    command: npm start
# This section defines the mern-app network, which is a
# bridge network that will be used to connect the api-server
#and react-app services.
networks:
  mern-app:
    driver: bridge

#use the original api(for backend) : http://localhost:5000/
```

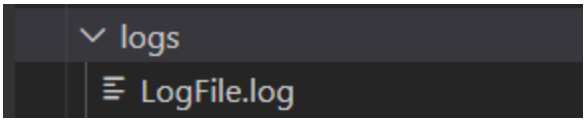
This file defines two services: `couchpotato-frontend(react-app)`, `couchpotato-backend(api-server)`. The `couchpotato-frontend` service builds the frontend application and exposes port 3000. The `couchpotato-backend` service builds the backend application and exposes port 5000.

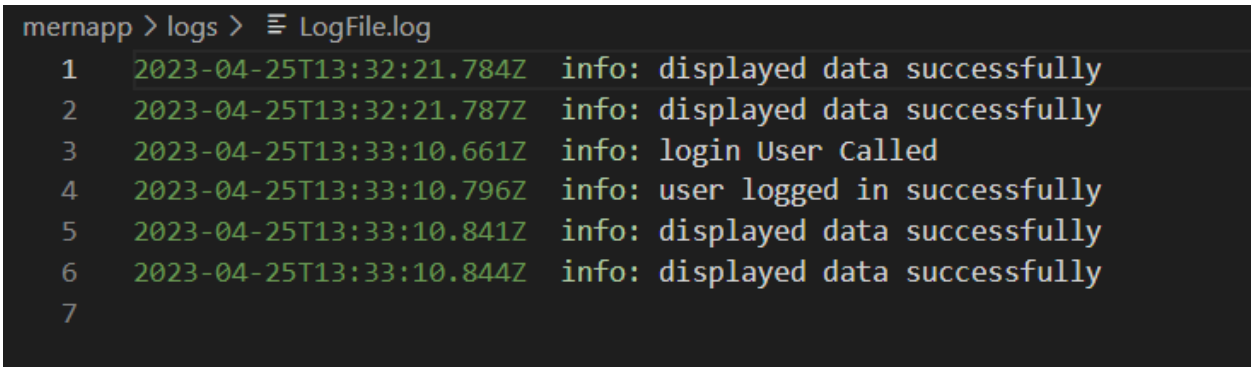
To start the application using Docker Compose, navigate to the directory containing the `docker-compose.yml` file and run the following command:

```
docker-compose up
```

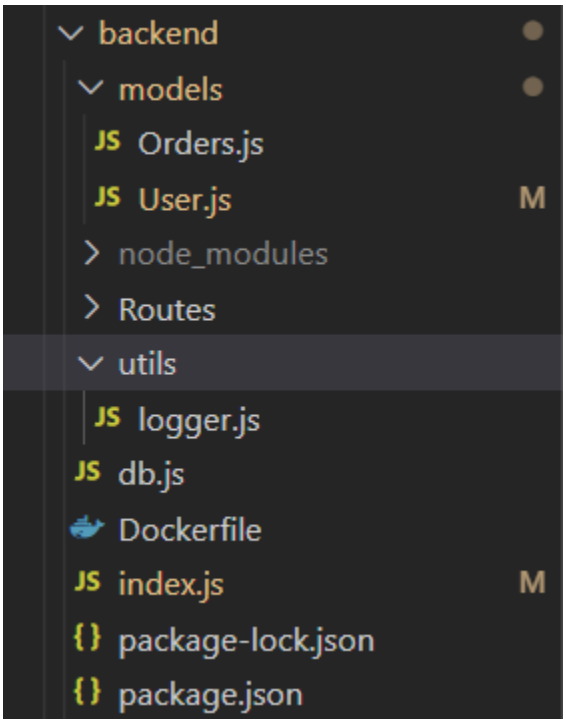
This will start all the services defined in the `docker-compose.yml` file and output their logs to the console. To stop the application, press `Ctrl-C` in the terminal.

The logs are also getting created due to the volumes specification that we added in the Docker-Compose file and can be used to directly send to ELK for the continuous monitoring.




The file name of log is defined in server side code , so the logs generated here are for server side therefore we have used volumes in api-server service.




logger.js is responsible for generating the logs and we are using Winston to generate our logs.

### logger.js in utils folder

```
const winston = require("winston");
const { createLogger, format, transports } = require("winston");

const { combine, timestamp, label, printf } = format;

const myFormat = printf(({ level, message, label, timestamp }) => {
  return `${timestamp} ${label} ${level}: ${message}`;
});

const logger = createLogger({
  format: combine(label({ label: "" }), timestamp(), myFormat),
  transports: [
    new transports.Console(),
    new winston.transports.File({
      filename: "/usr/src/app/logs/LogFile.log",
      level: "info",
    }),
  ],
});

module.exports = logger;
```

# Kubernetes

Kubernetes is an open-source container orchestration platform that helps automate the deployment, scaling, and management of containerized applications. It provides a platform-agnostic way to deploy and manage applications, making it easier to move applications between different cloud providers or on-premises data centers.

Kubernetes can be used to manage a wide range of containerized applications, including stateless applications, stateful applications, and batch jobs. It provides a powerful set of features for managing containerized applications at scale, including automatic scaling, rolling updates, and self-healing.

To deploy an application on Kubernetes, you need to create a Kubernetes deployment that specifies the container image to use and the desired number of replicas. You can then expose the deployment as a Kubernetes service to make it accessible from outside the cluster.

Kubernetes also provides a powerful set of tools for managing the configuration of your applications, including secrets, config maps, and environment variables. These tools make it easier to manage the configuration of your applications in a scalable and secure way.

Overall, Kubernetes is a powerful tool for managing containerized applications at scale, and is widely used in production environments to deploy and manage cloud-native applications.

We have also implemented kubernetes in our project. We made a separate folder for kubernetes and added the frontend-deployment.yaml, frontend-service.yaml, backend-deployment.yaml and backend-service.yaml in the folder. The files are mentioned below:

**Kubernetes Installation**

# Exercise 1: K8s Installation

You can refer to this link if you get stuck anywhere
https://kubernetes.io/docs/tasks/tools/

**Step 1:** Install kubectl

1.1. Download the latest release with the command:
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

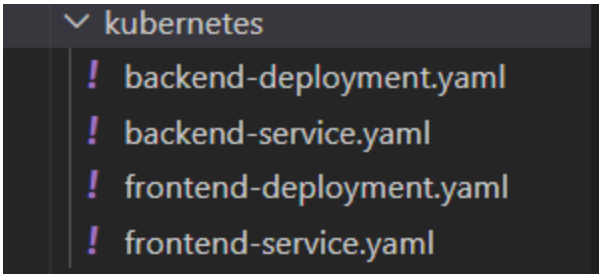
1.2. Install kubectl
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

1.3. Test to ensure the version you installed is up-to-date:
kubectl version --client

**Step 2:** Install minikube

To install the latest minikube **stable** release on **x86-64 Linux** using **binary download**:

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube




1. **frontend-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
```

```
      containers:
      - name: frontend
        image: ramrodwal/client:latest
        ports:
        - containerPort: 3000
        resources:
            limits:
              cpu: "1"
              memory: "1Gi"
            requests:
              cpu: "0.5"
              memory: "500Mi"
```

2. **frontend-service.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 3000
  type: NodePort
```

3. **backend-deployment.yaml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
```

```
      - name: backend
        image: ramrodwal/server:latest
        ports:
        - containerPort: 5000
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
          requests:
            cpu: "0.5"
            memory: "500Mi"
```

4. **backend-service.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
  - name: http
    protocol: TCP
    port: 90
    targetPort: 5000

  type: NodePort
```

- A **Kubernetes Deployment** tells Kubernetes how to create or modify instances of the pods that hold a containerized application. Deployments can help to efficiently scale the number of replica pods, enable the rollout of updated code in a controlled manner, or roll back to an earlier deployment version if necessary. Kubernetes deployments are completed using kubectl, the command-line tool that can be installed on various platforms, including Linux, macOS, and Windows.
- With a deployment, you declare a single object in a **YAML** file. This object is responsible for creating the pods, making sure they stay up to date, and ensuring there are enough of them running.
- A **Kubernetes Service** is a logical abstraction for a deployed group of pods in a cluster(which all perform the same function).

- Since pods are ephemeral, a service enables a group of pods, which provide specific functions (web services, image processing, etc.) to be assigned a name and unique IP address (clusterIP). As long as the service is running that IP address, it will not change.
- Services also define policies for their access.

Now after creating the above four files , you need to apply each of them

```
kubectl apply -f kubernetes/bacend-deployment.yaml
```

```
kubectl apply -f kubernetes/bacend-service.yaml
```

```
kubectl apply -f kubernetes/frontend-deployment.yaml
```

```
kubectl apply -f kubernetes/frontend-service.yaml
```

# Ansible

Ansible is an open-source automation tool that can help streamline IT configuration management and application deployment, as well as automate repetitive tasks. With Ansible, you can define and manage your infrastructure as code, making it easier to scale and maintain your IT systems.

Ansible uses a simple and powerful language called YAML to define configuration files. You can use it to manage servers, network devices, and other infrastructure components. It also provides a powerful set of features for managing configuration files, including templates, variables, and conditionals.

Overall, Ansible is a powerful tool for automating IT operations and can help organizations improve efficiency, reduce errors, and increase reliability.

We are going to be pulling the Docker Hub image to the host system for Ansible

deployment.

Creating Inventory file

The inventory file is used to specify the list of managed hosts/server machines.

The inventory file looks as,

```
[client1]
172.16.144.201 ansible_ssh_user=vartika ansible_ssh_pass=root121 sudo=vartika ansible_sudo_pass=root121
```

Create this file in the root directory of your project repository with the name inventory.

Configuring OpenSSH Server

Install openssh-server on the host machine and because it is the Jenkins user that is going to be doing the pulling of Docker image, we need to SSH from the Jenkins user to the user that is specified in the inventory file, i.e. vartika.

`apt-get install openssh-server`

`service ssh restart`

`su jenkins`

`ssh-keygen -t rsa`

`ssh-copy-id <host-username>@<host-ip-address>`

`sudo chmod 666 /var/run/docker.sock`

The chmod command is required so that the Jenkins user has access to the Docker socket for performing the docker build and push operations.

**playbook.yml**

```
---
- name: Deploy Docker Images
  hosts: client1
  tasks:
    - name: Copy Docker Compose file from host machine to remote host
      copy:
        src: mernapp/docker-compose.yml
        dest: ./
    # Pull the Docker images from Docker Hub
    - name: Pull the Docker images specifed in docker-compose
      command: docker-compose pull
    # Detached mode is required, otherwise Jenkins build never exits
    # even though the docker-compose up command has successfully executed
    - name: Run the pulled Docker images in detached mode
      command: docker-compose up -d

    # This is added so that the Docker images of the previous builds
    # which will now become dangling images are removed
    - name: Prune the dangling Docker images
      command: docker image prune --force
```

This file will pull the images specified in docker-compose.yml file
After pulling the image using `docker-compose up` you can start the container and run the application

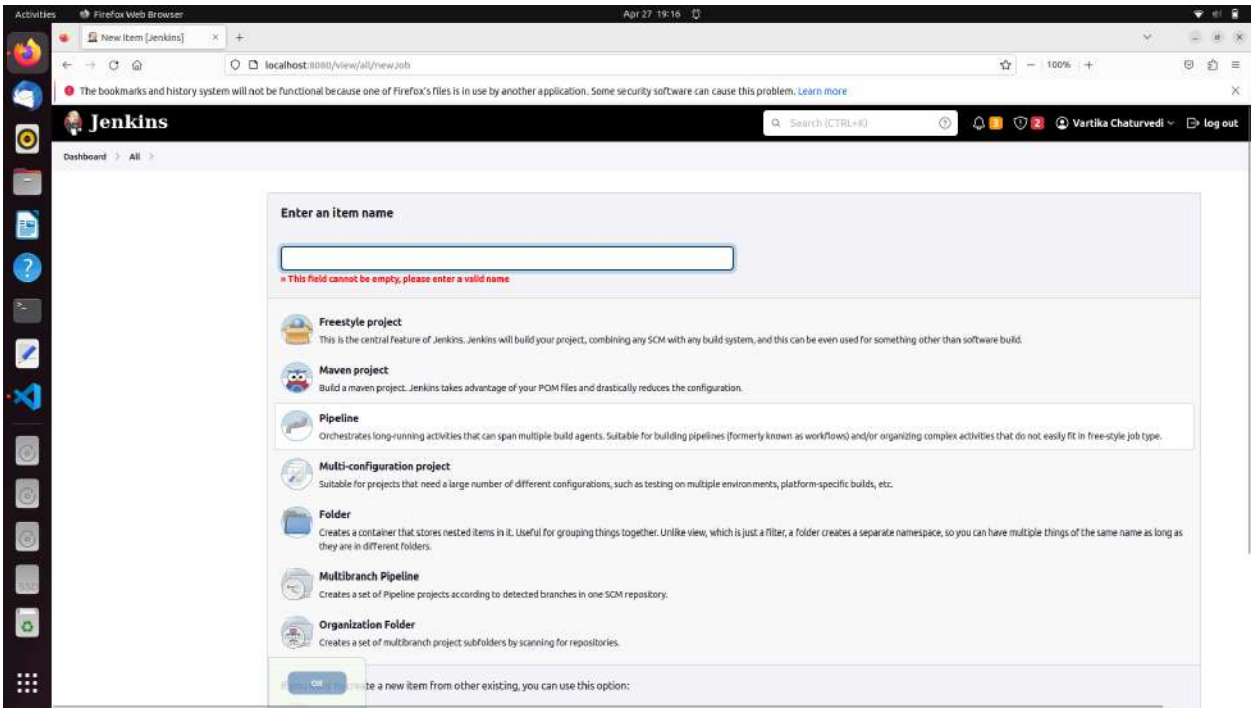
## Continuous Integration: Jenkins

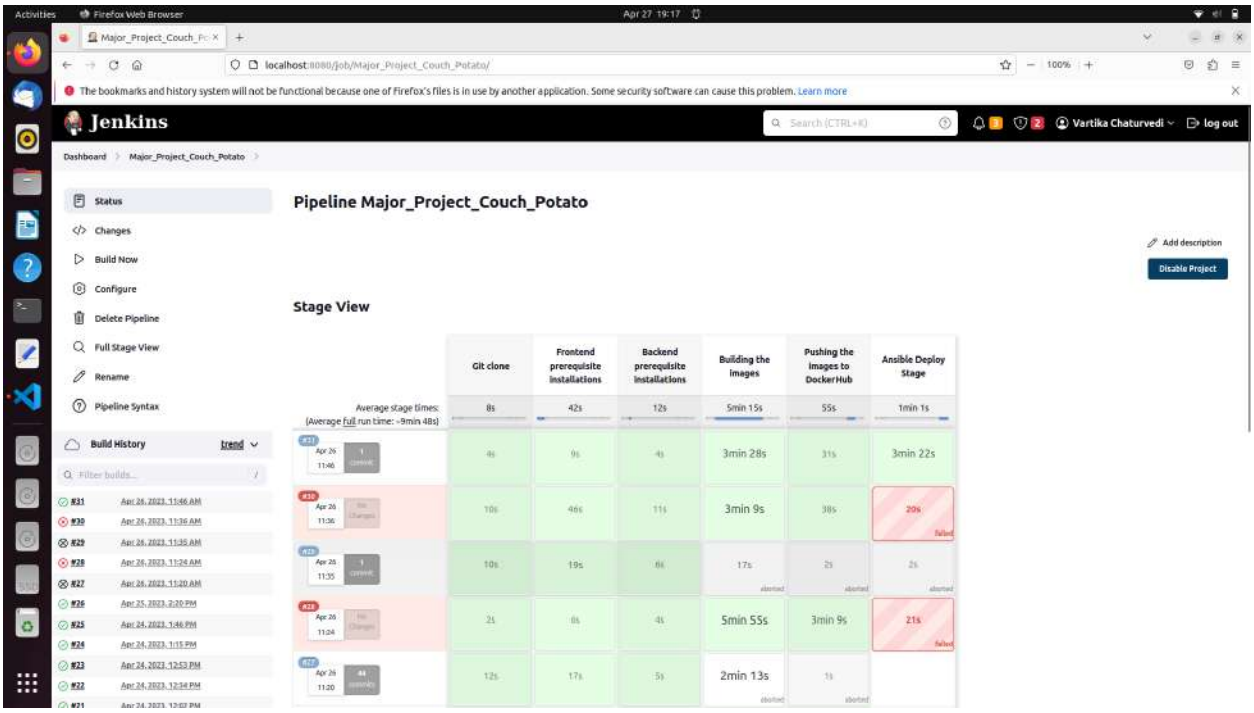
Jenkins is an open-source automation server that can be used to automate various aspects of software development, including building, testing, and deploying code. It integrates with a wide range of tools and technologies, making it a versatile tool for continuous integration and continuous delivery (CI/CD) pipelines.

To use Jenkins for continuous integration, we first need to install and configure it on a server or cloud instance. Once Jenkins is up and running, we can create a new job that defines the build process for your application. This job can include steps for checking out code from a repository, building the application, running tests, and deploying the application to a staging or production environment.

Jenkins provides a powerful set of features for managing the build process, including support for plugins, pipelines, and workflows. It also provides a web-based interface for managing builds and monitoring the status of your applications.

By using Jenkins for continuous integration, we can automate the process of building, testing, and deploying your applications, making it easier to maintain and scale your software development projects.

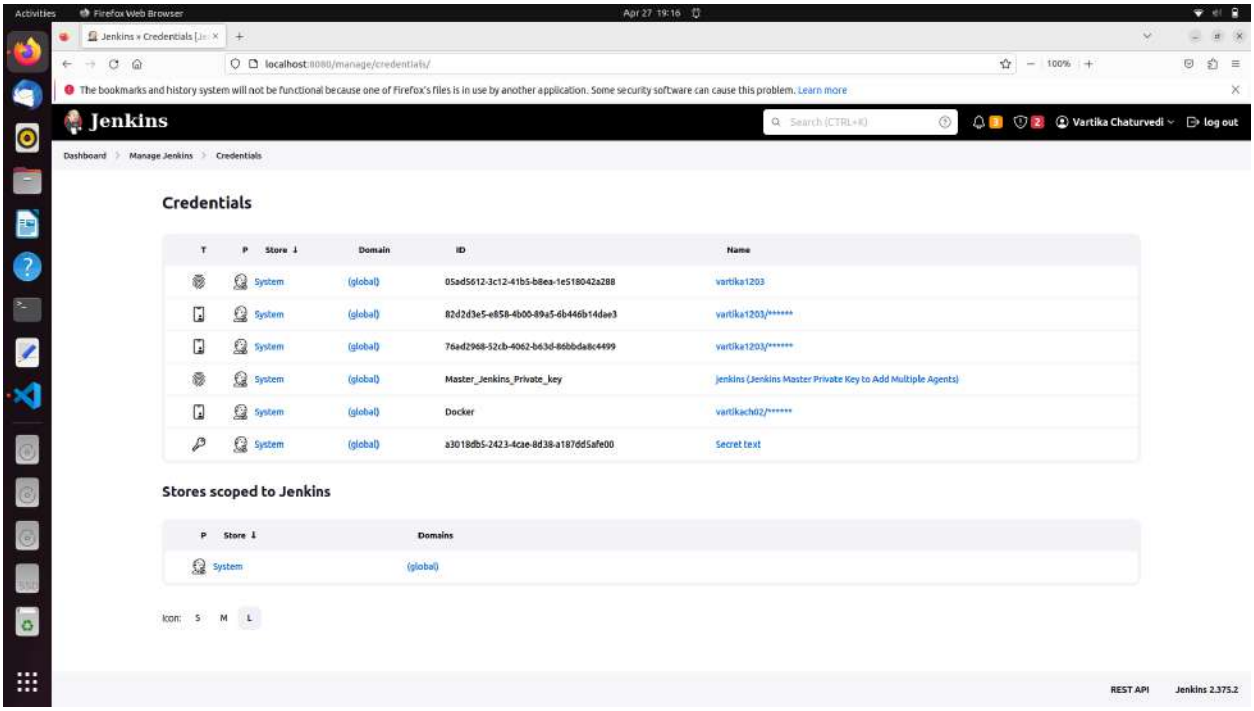

Activities
Firefox Web Browser
Apr 27 19:16
New Item [Jenkins]
localhost:8080/view/all/newJob
The bookmarks and history system will not be functional because one of Firefox's files is in use by another application. Some security software can cause this problem. Learn more
Jenkins
Search (CTRL+K)
Vartika Chaturvedi
log out
Dashboard
All
Enter an item name
» This field cannot be empty, please enter a valid name
Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.
Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.
te a new item from other existing, you can use this option:


Activities
Firefox Web Browser
Apr 27 19:17
Major_Project_Couch_Fc
localhost:8080/job/Major_Project_Couch_Potato/
The bookmarks and history system will not be functional because one of Firefox's files is in use by another application. Some security software can cause this problem. Learn more
Jenkins
Search (CTRL+K)
Vartika Chaturvedi
log out
Dashboard
Major_Project_Couch_Potato
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
Rename
Pipeline Syntax
Build History
trend
Pipeline Major_Project_Couch_Potato
Add description
Disable Project
Stage View
Git clone
Frontend prerequisite installations
Backend prerequisite installations
Building the images
Pushing the images to DockerHub
Ansible Deploy Stage
Average stage times:
(Average full run time: ~9min 48s)
8s
42s
12s
5min 15s
55s
1min 1s
3min 28s
31s
3min 22s
3min 9s
38s
20s
17s
2s
5min 55s
3min 9s
21s
2min 13s

## Credentials & Jenkins

**Github :** For accessing private Github repositories.

**Docker Hub Credentials:** For logging into Docker Hub account and pushing Docker




# Pipeline Script

1. **Git Pull:** It pulls the repository from the GitHub
2. **Frontend prerequisite installations:** This step is for building our react app.
3. **Backend prerequisite installations:** This step is for building our server.
4. **Building the images:** It is used to create images in our local system of the frontend and backend separately.
5. **Pushing the images to DockerHub:** The build images are pushed into the public DockerHub repository so that they can be pulled by anyone later on or during docker-compose by us.
6. **Ansible Deploy:** This is the deployment stage were using the already build images and the concept of containerization we can now execute our app on any platform using the Ansible inventory file and the playbook files

### Jenkins Pipeline

```
pipeline {
agent any
stages {
stage("Git clone Stage") {
steps {
// git clone stage
git url:
'[https://github.com/ramrodwal/SPE_MajorProject_CouchPotato.git](https://github.com/ramrod
wal/SPE_MajorProject_CouchPotato.git)', branch: 'master'
}
}
stage("Frontend prerequisite installations stage") {
steps {
//front end installing the dependencies
dir('mernapp'){
sh 'npm install'
}
}
}
stage("Backend prerequisite installations stage") {
steps {
dir('mernapp/backend'){
sh 'npm install'
}
}
}
stage('Building the images stage'){
steps {
dir('client'){
//building the docker image for frontend
sh 'docker build -t vartikach02/client:latest .'
}
dir('server'){
//building the docker image for backend
sh 'docker build -t vartikach02/server:latest .'
}
}
}
stage('Pushing the images to DockerHub stage'){
steps {
script {`
withDockerRegistry([ credentialsId: "Docker", url: "" ]) {
// publishing the image created above for frontend
sh 'docker push vartikach02/client:latest'
}
withDockerRegistry([ credentialsId: "Docker", url: "" ]) {
// publishing the image created above for backend
sh 'docker push vartikach02/server:latest'
```

```
}
}
}
}
International Institute of Information Technology, Bangalore 30
stage('Ansible Deploy Stage') {
steps {
ansiblePlaybook becomeUser: 'null',
colorized: true,
installation: 'Ansible',
inventory: 'inventory',
playbook: 'playbook.yml',
sudoUser: 'null',
vaultCredentialsId: 'Ansible'
}
}
}
}
```

# Logs and Monitoring

When the application is deployed and running properly on the managed node, we also want to check whether there are any problems in the run time or not. To do that we can implement a monitoring system using the ELK stack.

1. **While using Docker-Compose:**

   The winston library helps us to maintain the logs of the user whenever the user tries to do anything(whether it is a success or not)

   These logs can then be either automatically sent to the ELK stack using the elastic-search, logstash and kibana pre-installed in your system or by manually uploading the file that has been created.

2. **While using Kubernetes**

   With Kubernetes we could either use the shell command to extract the logs of a particular pod or we could go to our mnikube dashboard by writing the following command on our terminal:

   `minikube dashboard`

After going to the dashboard, we can directly go to the pod for which we want to see the logs. From there, we can dowload the logs of the particular pod and then manually upload the log file on the ELK cloud to see and visualise the logs.

## ELK for continous monitoring

ELK (Elasticsearch, Logstash, and Kibana) is a popular open-source stack used for logging and monitoring. Elasticsearch is a search engine and analytics platform used to store and index logs. Logstash is a data processing pipeline used to collect, process, and forward logs to Elasticsearch. Kibana is a data visualization tool used to create dashboards and analyze log data.

In the case of Couch Potato, we can use ELK to monitor the logs generated by the application and gain insights into the application's performance and behavior. By analyzing the logs, we can identify any errors or issues that need to be addressed and optimize the application for better performance.

To use ELK with Couch Potato, we can configure Logstash to collect the logs generated by the application and send them to Elasticsearch for indexing and storage. We can then use Kibana to create visualizations and dashboards based on the log data, providing insights into the application's behavior and performance.

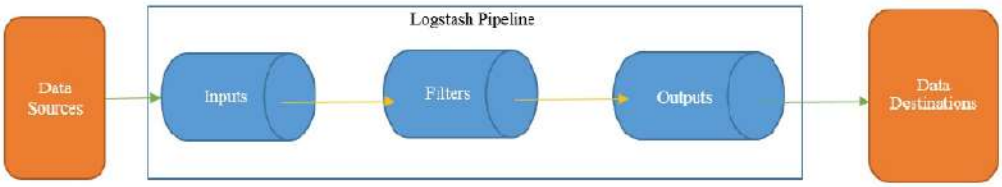
Overall, ELK is a powerful tool for monitoring and analyzing logs, and can help us gain valuable insights into the performance and behavior of our applications.

**ELK Stack**

1. **ElasticSearch**
   Elasticsearch is the living heart of what is today's the most popular log analytics platform
   — the ELK Stack (Elasticsearch, Logstash and Kibana). Elasticsearch's role is so central
   that it has become synonymous with the name of the stack itself. Primarily for search and
   log analysis, Elasticsearch is one of the most popular database systems available today.
   This Elasticsearch tutorial provides new users with the prerequisite knowledge and tools

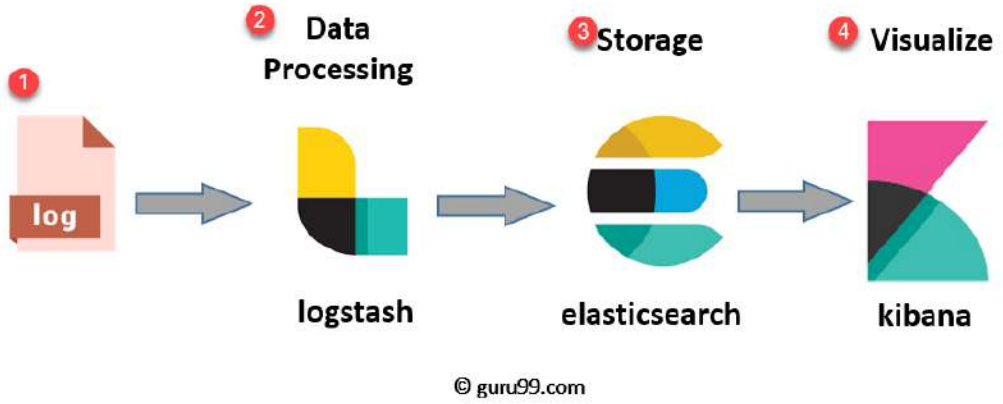
to start using Elasticsearch. It includes installation instructions, and initial indexing and
data handling instructions.


2. **Logstash**
   The classic definition of Logstash says it's an open-source, server-side data processing
   pipeline that can simultaneously ingest data from a wide variety of sources, then parse,
   filter, transform and enrich the data, and finally forward it to a downstream system.In most
   cases, the downstream system is Elasticsearch. These applications collect logs from
   different sources (software, hardware, electronic devices, API calls, etc.), process the
   collected data, and forwards it to a different application for further processing or storing.This makes Logstash essentially a data pipeline. But there's more to it than typical
   pipelines that data engineers develop.




3. **Kibana**

   Kibana is a data visualization and exploration tool used for log and time-series analytics,
   application monitoring, and operational intelligence use cases. It offers powerful and
   easy-to-use features such as histograms, line graphs, pie charts, heat maps, and built-in
   geospatial support.




## Installation:

1. **ElasticSearch**

```
wget -qO -
```

```
https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
```

```
echo "deb https://artifacts.elastic.co/packages/7.x/apt stable main" | sudo tee
```

```
/etc/apt/sources.list.d/elastic-7.x.list
```

```
sudo apt-get update
```

```
sudo apt-get install elasticsearch
```

```
sudo systemctl start elasticsearch
sudo systemctl status elasticsearch
```

2.**Logstash**

```
sudo apt-get install -y apt-transport-https sudo apt-get update && sudo apt-get
install -y
logstash
sudo systemctl start logstash
sudo systemctl status logstash
```

Now goto /etc/logstash/conf.d and add the logstash.conf file in order to add the pipeline
stages

Here you can add the grok pattern and the mutate also

3. **Kibana**

```
sudo apt-get update
sudo apt-get update
sudo systemctl start kibana
sudo systemctl enable kibana
sudo systemctl status kibana
```

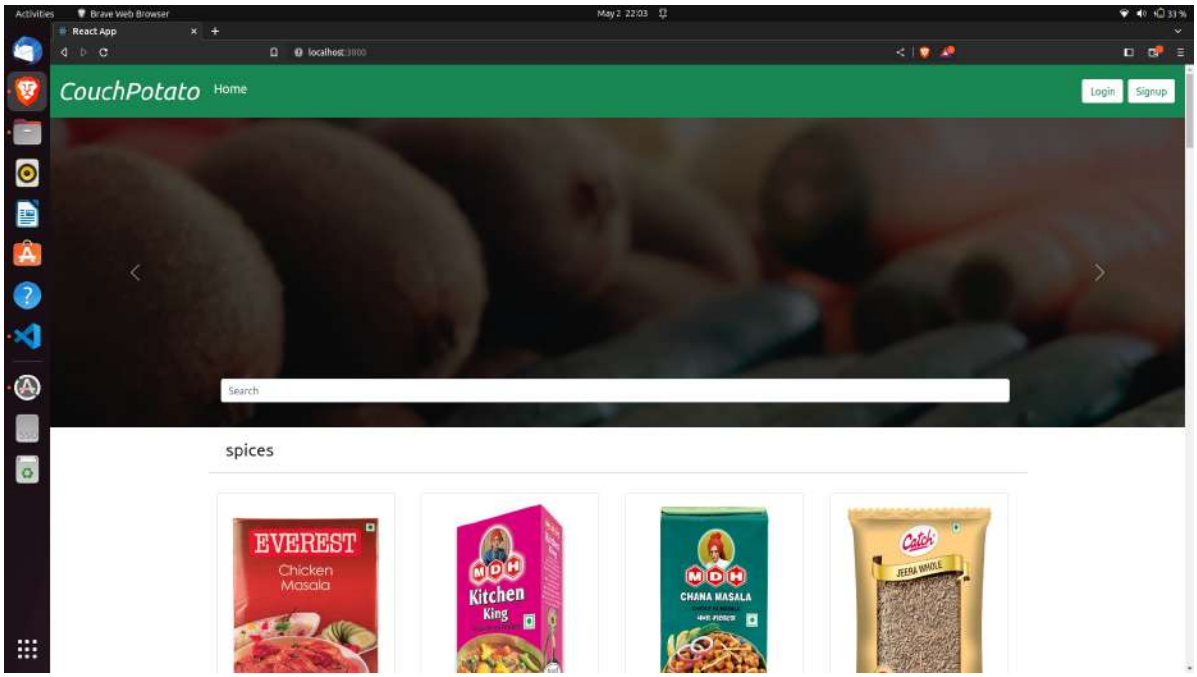
### Visualisation

Now start all three services (ELK) and goto `localhost:5601` , add index pattern and discover
the logs created by your application in real-time.

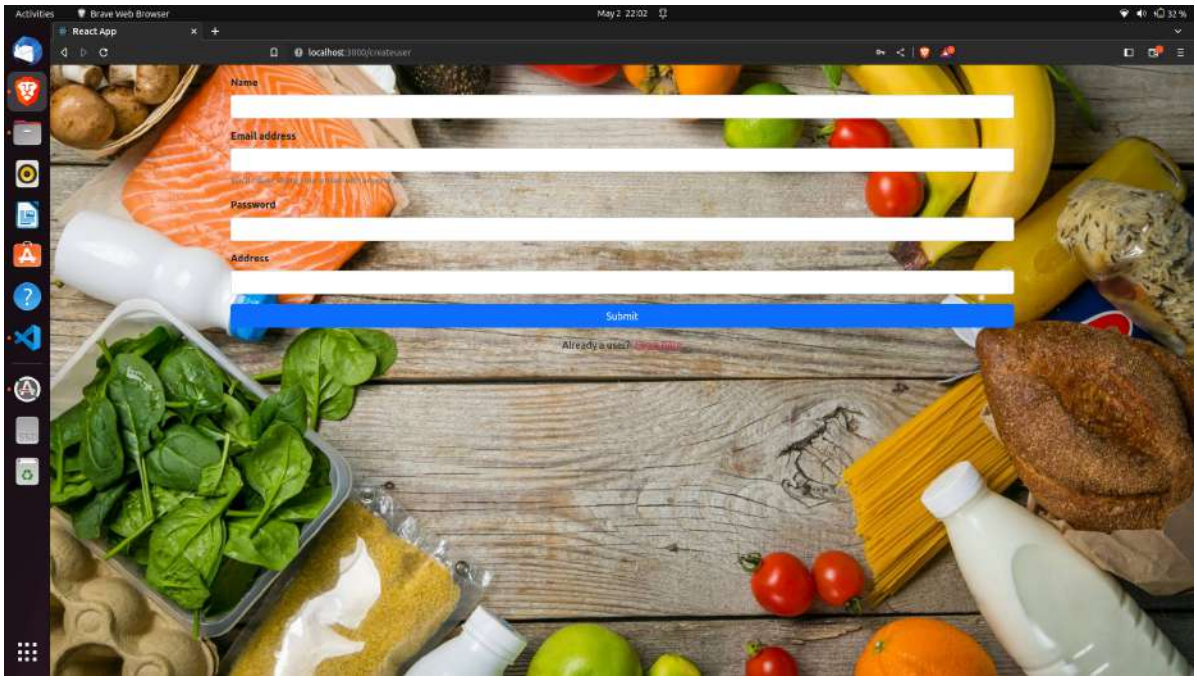
## Features and API

### Features

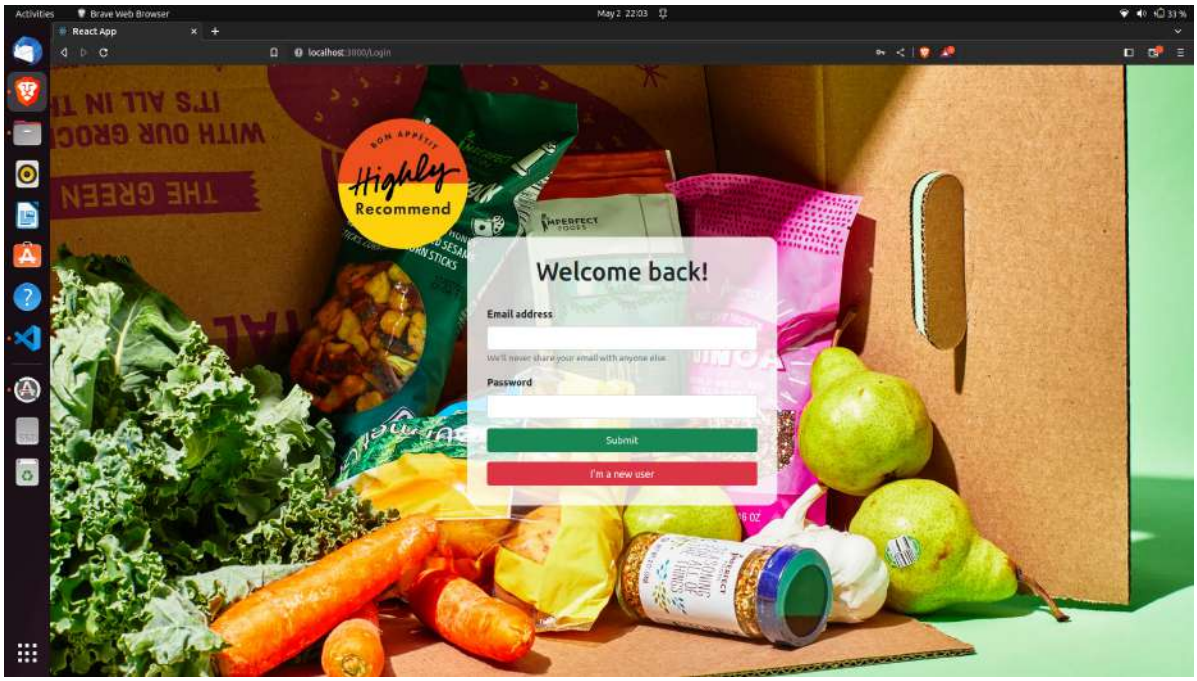
1. Landing Page



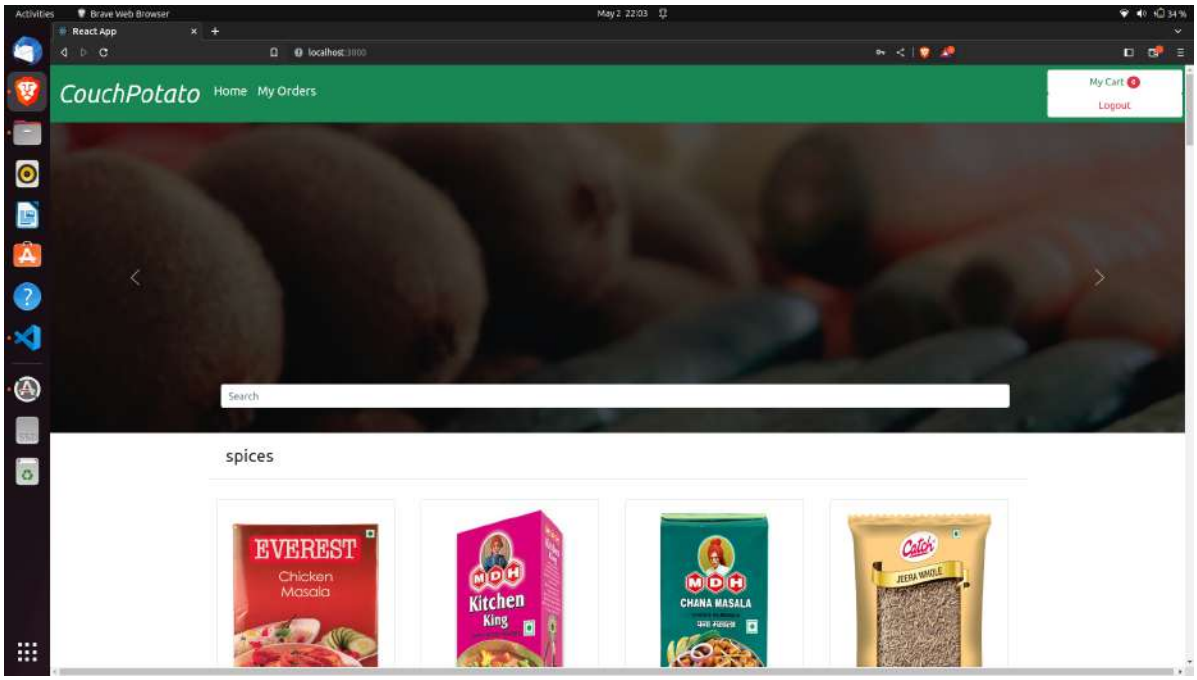

2. SignUp Page:



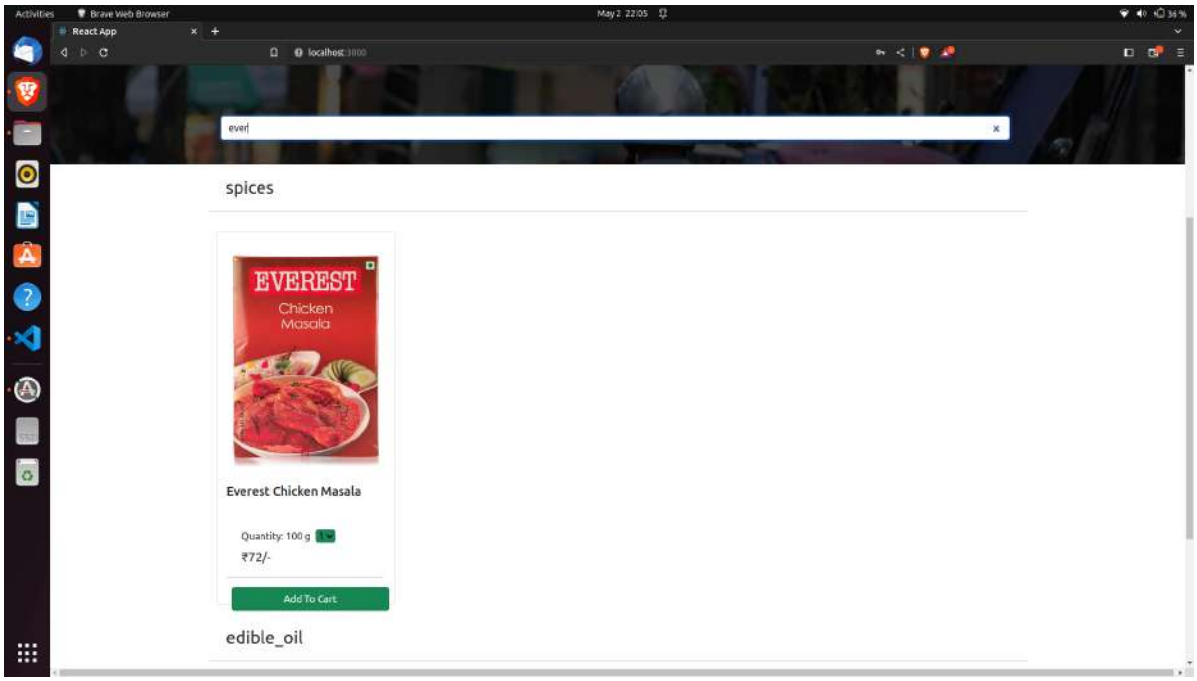

3. Login Page:



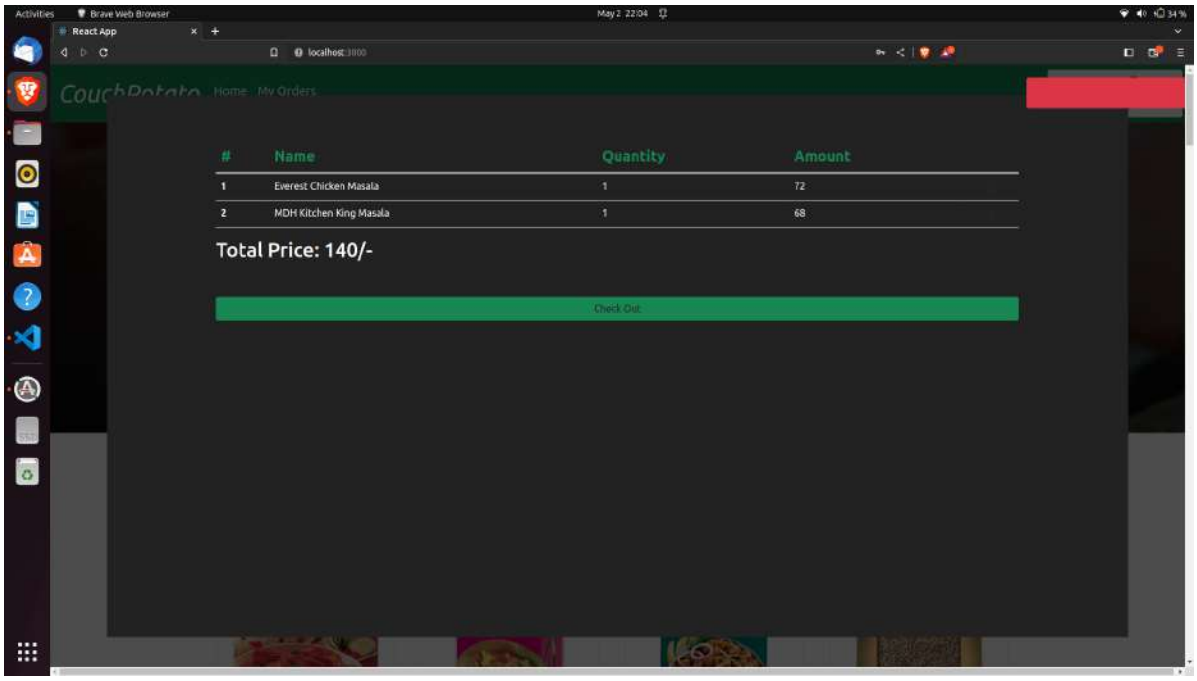

4. User Dashboard:



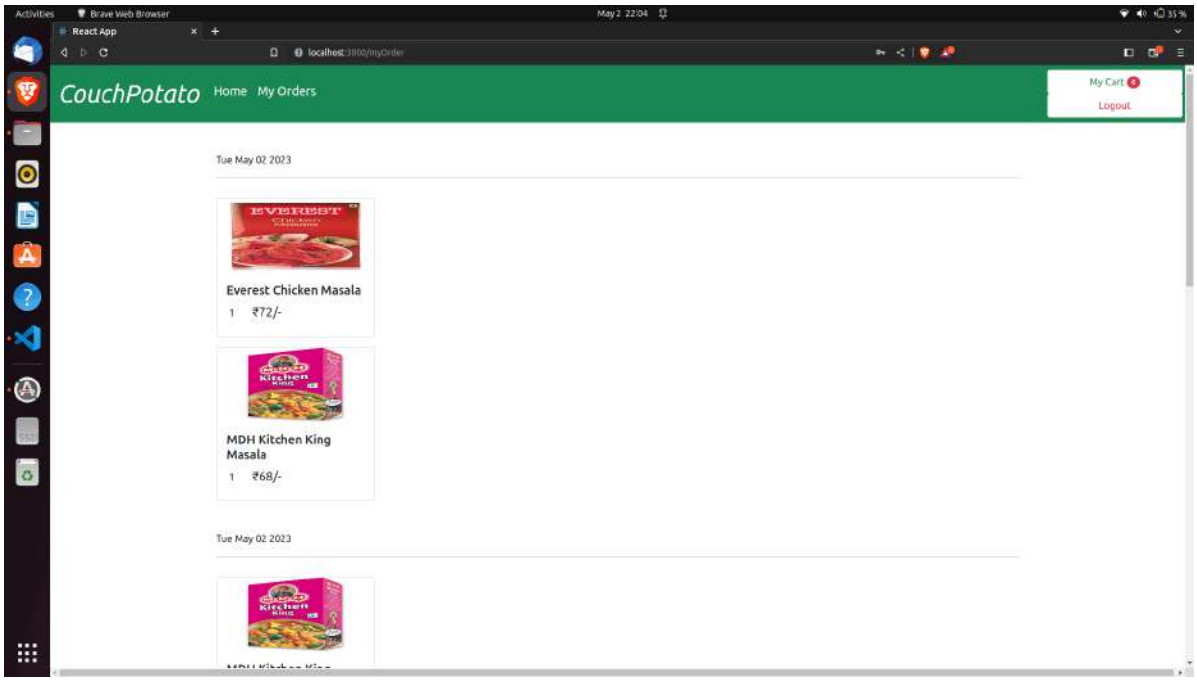

5. Search an item




6. Add to cart, view and update cart:




7. View your orders:



## APIs

| Serial no. | Use Case | API | Request Method | End-user |
|---|---|---|---|---|
| 1. | Register User | /api/CreateUser | POST | User |
| 2. | Login User | /api/loginuser | POST | User |
| 3. | Fetch the data | /api/food-data | POST | User |
| 4. | Display the fetched data | /api/DisplayData | POST | User |
| 5. | Place the order | /api/OrderData | POST | User |
| 6. | View all the orders placed based on the date | /api/myorderData | POST | User |
| 7. | Add new items to the inventory | /api/additem | POST | Admin |

## Code Snapshots

### index.js

```
// I faced an error here with nodemon - permission was disabled
//For that run windows powershell as administrator and then  "Set-ExecutionPolicy RemoteSigned"

const express = require('express')
const cors = require('cors');
const app = express()
const port = 5000
const mongoDB = require("./db");
// const { Next } = require('react-bootstrap/esm/PageItem');
mongoDB();
app.use(cors({
  origin: '*'
}));

app.use(cors({
  methods:['GET','POST','PUT','DELETE']
}));


//we did this because of CORS error coming when we were trying to submit a form
app.use((req, res, next)=> {
  // res.setHeader("Access-Control-Allow-Origin", "http://localhost:3000");
  res.setHeader("Access-Control-Allow-Origin", "*");
  res.header(
    "Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept,Auhtorization"
  );
  next();
});

app.get('/', (req, res) => {
  res.send('Hello World!')
})

//needed for hitting apis
app.use(express.json())
app.use('/api', require("./Routes/CreateUser"));
```

```
  app.use('/api', require("./Routes/DisplayData"));
  app.use('/api', require("./Routes/OrderData"));
  app.use('/api', require("./Routes/myorderData"));


  app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
  })
```

**Order.js**

```
const mongoose = require('mongoose')

const { Schema } = mongoose;

const OrderSchema = new Schema({
    email: {
        type: String,
        required: true,
        unique: true
    },
    order_data: {
        type: Array,
        required: true,
    },

});

module.exports = mongoose.model('order', OrderSchema)
```

**User.js**

```
const mongoose =require('mongoose');
const { Schema } = mongoose;

const UserSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true
    },
    location:{
        type: String,
        required: true
    },
    email:{
        type:String,
        required: true
    },
    password:{
        type: String,
        required: true
    },
    date:{
        type: Date,
        default: Date.now
    }
});

//here the model will get exported so that in the model we
// can perform CRUD operations
module.exports = mongoose.model('user', UserSchema);

```

## Key challenges

The challenges we faced initially were related to docker-compose. We were able to create the images properly and both the containers were running as well but the frontend and backend were unable to communicate with each other. The reason being we did not provide the container name in the docker-compose file. The fix was to put the container name in the docker-compose file and then use the same “container name” as

URI in the API instead of “localhost”.

We also faced the issue in kubernetes. The frontend was up and running but we were unable to communicate with the backend. We solved this by changing the end point of the APIs as the backend is also getting hosted on Kubernetes, so the end point of the APIs will also get changed. So we added the backend URI of the backend-service that was running on kubernetes.

Another challenge was with creating the log file in kubernetes. Finally we realised that we could simply download the log file from the minikube dashboard.

# Key learnings

## Technical

1. React.js
2. Node.js
3. Mongo DB
4. Jenkins
5. Docker
6. Kubernetes
7. Ansible
8. API creation
9. CI/CD pipeline
10. Docker Registry

# Experience

- Designing this application has taught us the power of collaborating and the importance

  of being a team player.Software Production Engineering (CS816)

- Creating the application from scratch and aligning it with the team’s vision and choosing

the correct people for the appropriate task have taught us to believe and support each

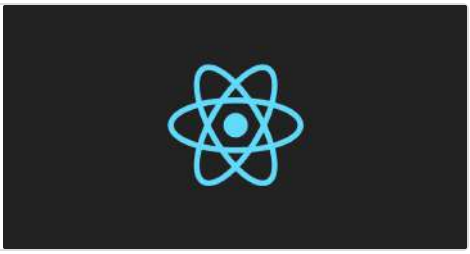
other.

# References


Getting Started – React

A JavaScript library for building user interfaces

https://legacy.reactjs.org/docs/getting-started.html



Documentation | Node.js

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

https://nodejs.org/en/docs

Welcome to the MongoDB Documentation

https://www.mongodb.com/docs/


Tutorials overview

Jenkins – an open source automation server which enables developers around the world to reliably build, test, and deploy their software

https://www.jenkins.io/doc/tutorials/#pipeline/

### Overview

Get started with the Docker basics in this comprehensive overview, You'll learn about containers, images, and how to containerize your first application.

https://docs.docker.com/get-started/


### Kubernetes Documentation

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native

https://kubernetes.io/docs/home/


### Ansible Documentation

https://docs.ansible.com/

### Welcome to Elastic Docs | Elastic

https://www.elastic.co/guide/index.html