FULL STACK PROJECT : RECIPE-MEDIA

By:
Aakash Anand (MT2022009)
Sameeksha Gupta(MT2022098)

Git URL: https://github.com/akashanand842/RecipeMedia-SPEMajor
Branch name: final

TECHNOLOGY USED:

- MongoDB
- Express
- React
- NodeJs

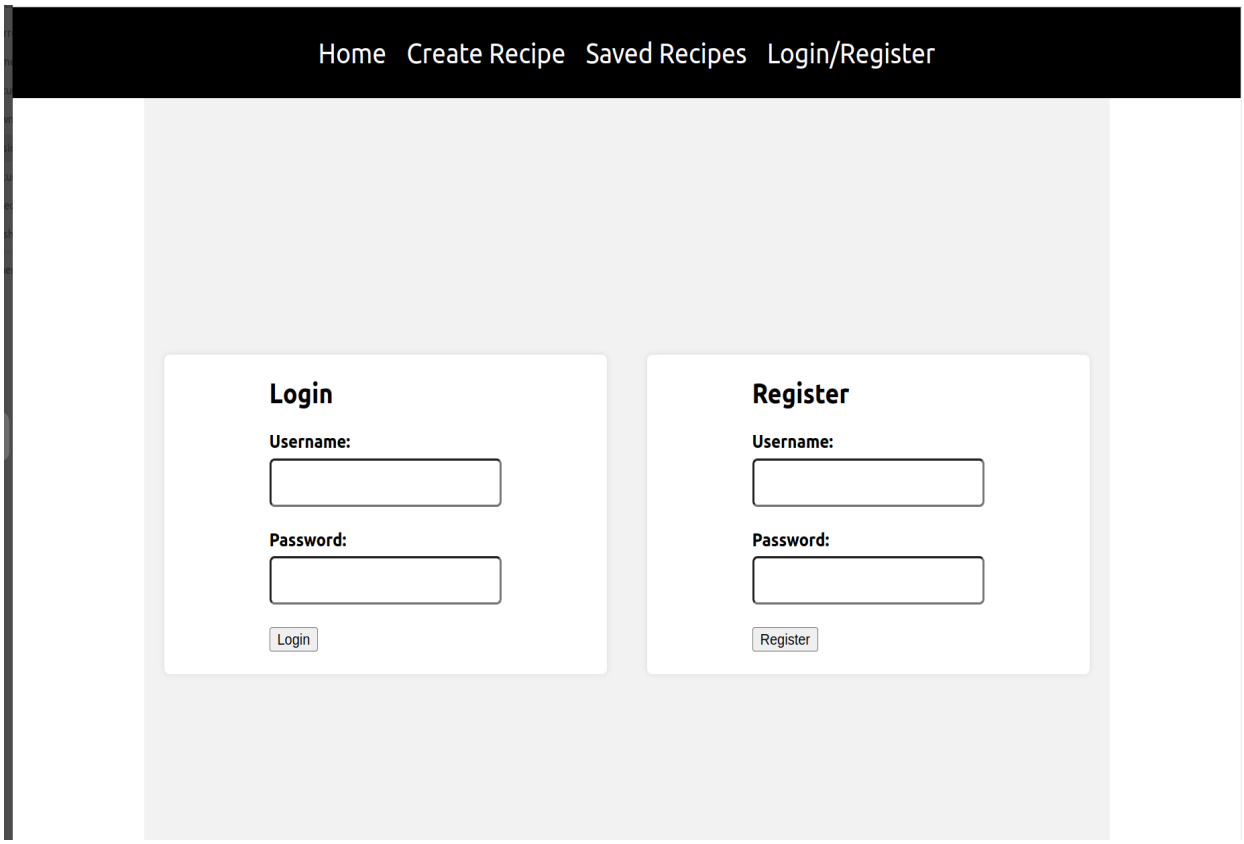
FUNCTIONALITIES IMPLEMENTED:

1. Login
2. Register
3. Create Recipes
4. View Recipes
5. Save Recipes
6. Search Recipes
7. Add Rating
8. Add Comments

Login And Register

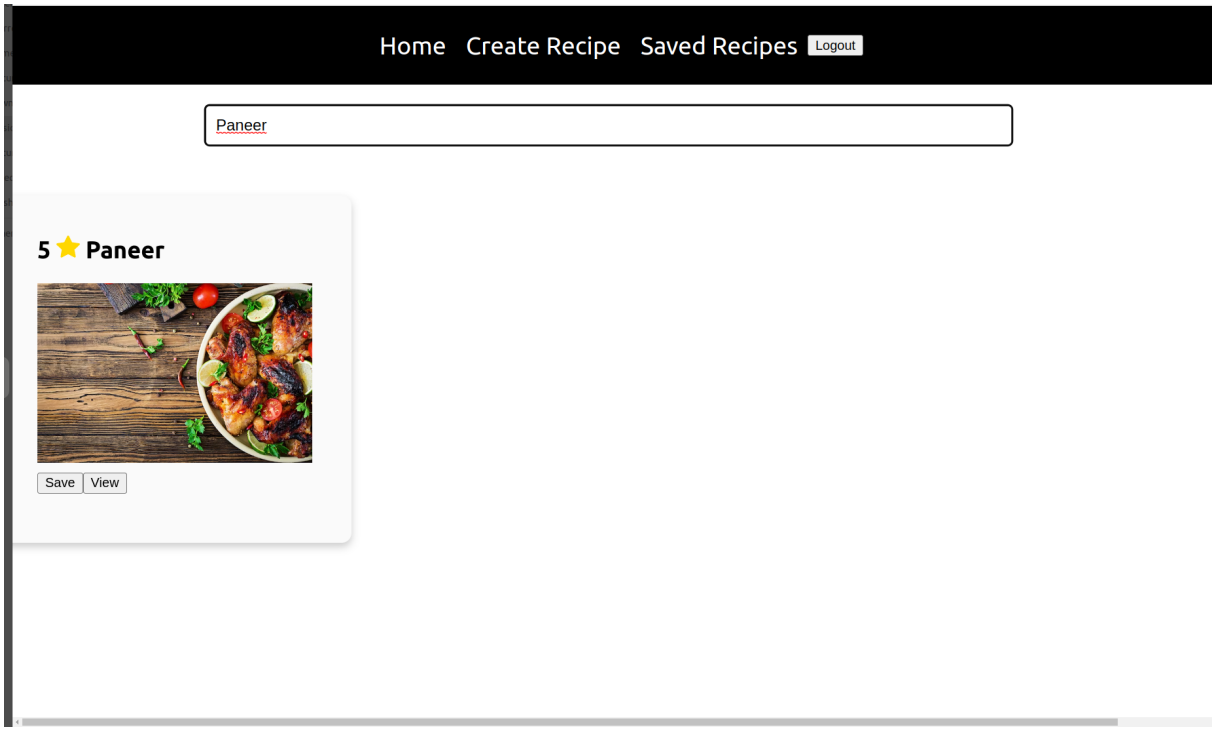
When frontend is started the user will have to choose Login/Register option in the navbar.

Then there will be two options:
Login : Already registered user can login by entering their Username and Password.
Register:New user will first register them by entering Username and Password . On successful registration an alert message will appear . After registration user can use their credentials to login.



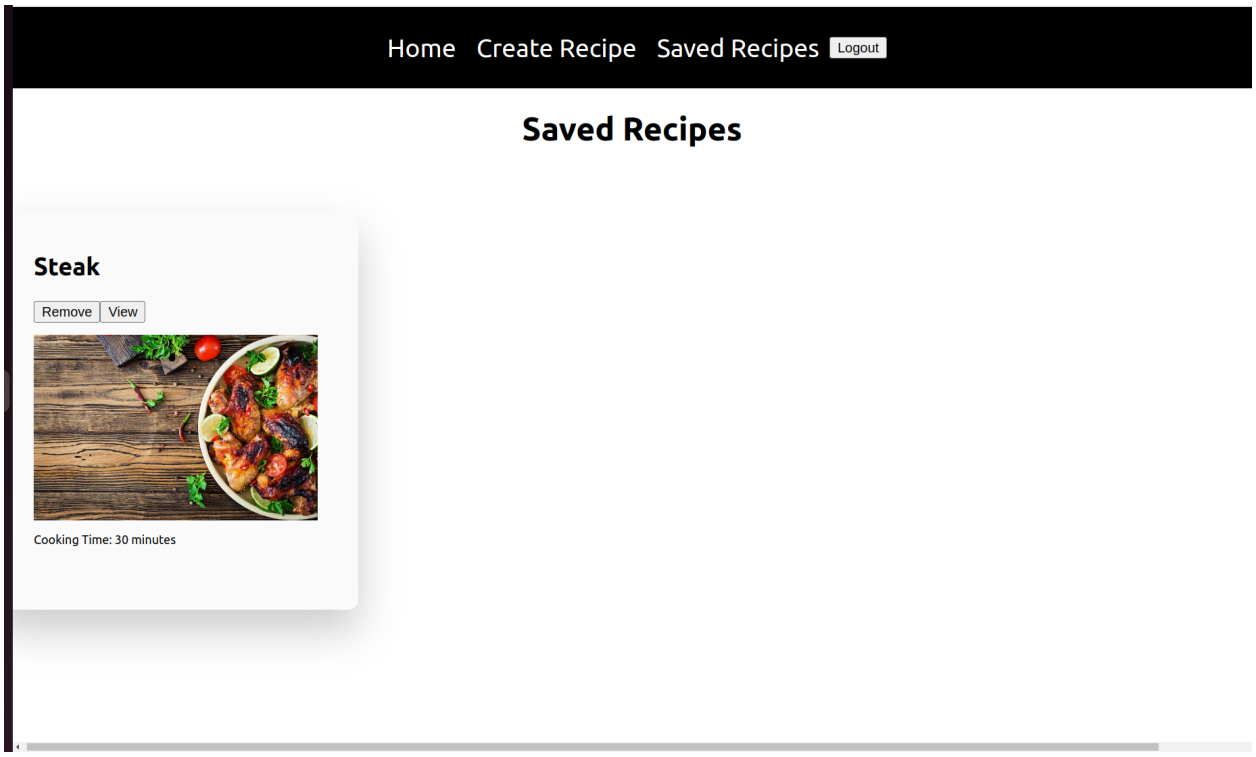
Search Recipe

Once the user is logged in , user can search recipe by giving their search query in search box.



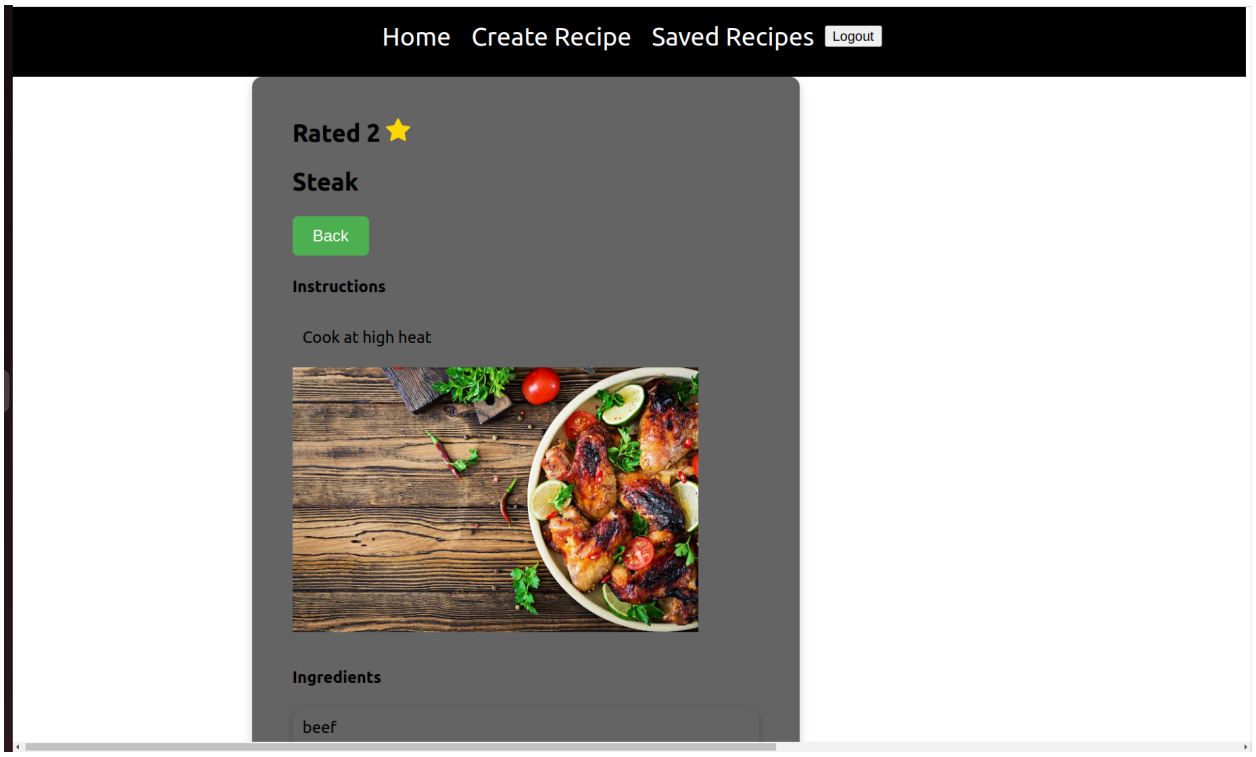
Save Recipe

There will be an option to save recipe by clicking on save button. The saved recipes can be viewed by clicking on Saved Recipe option in the navbar.



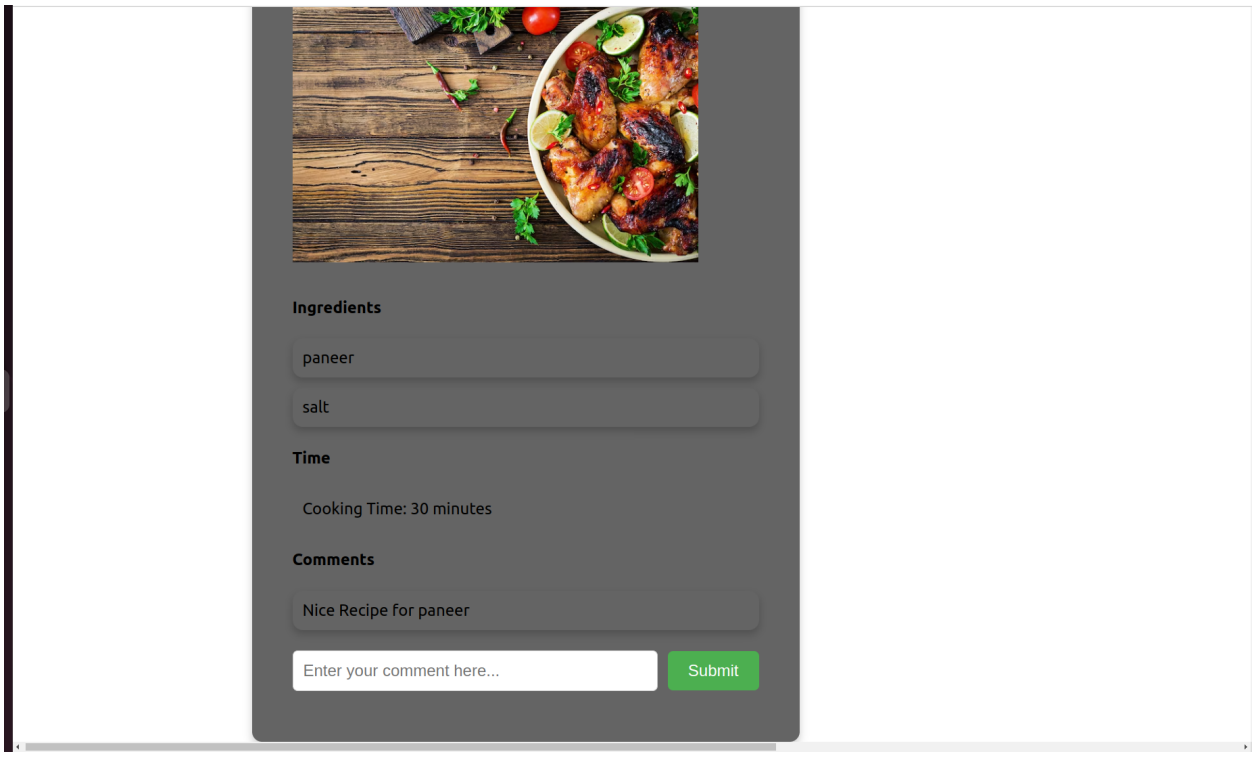
View Recipe

There will be an option to view recipe by clicking on view button. On clicking view button all the the description about recipe will be displayed.User will we able to give rating and add comments.



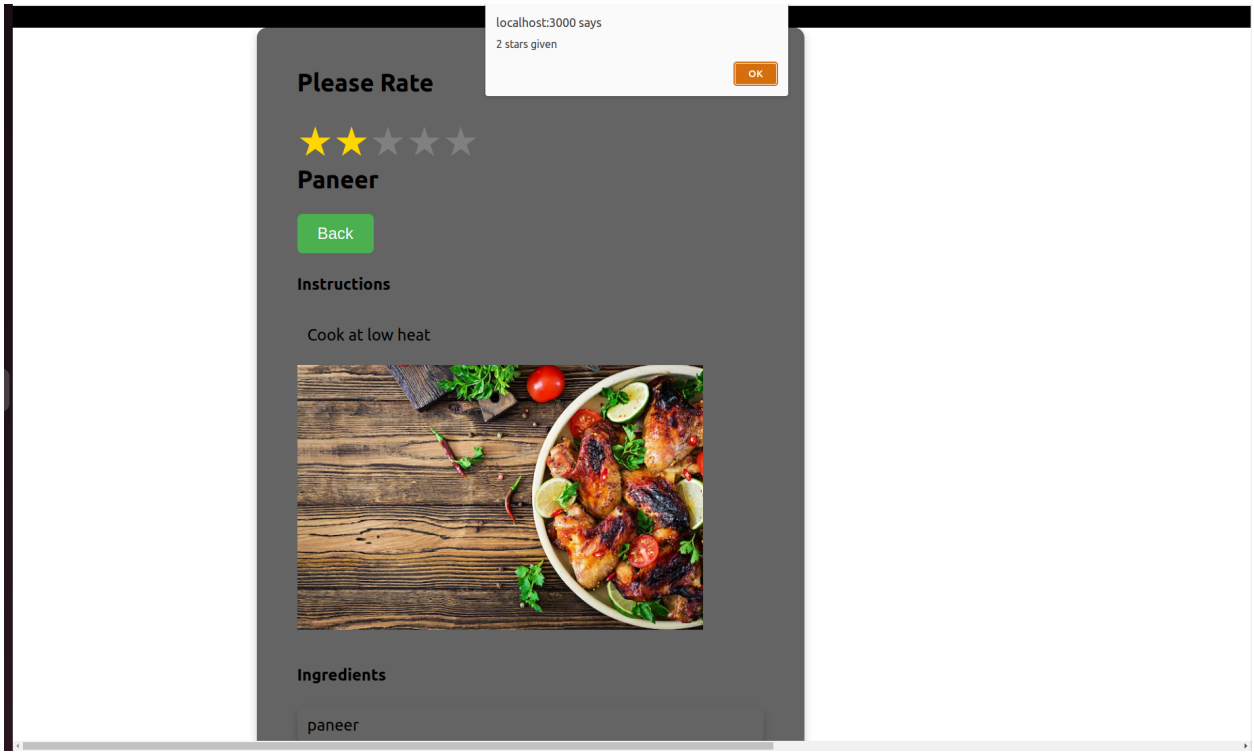
Add Comments

User can add comments for the recipe.



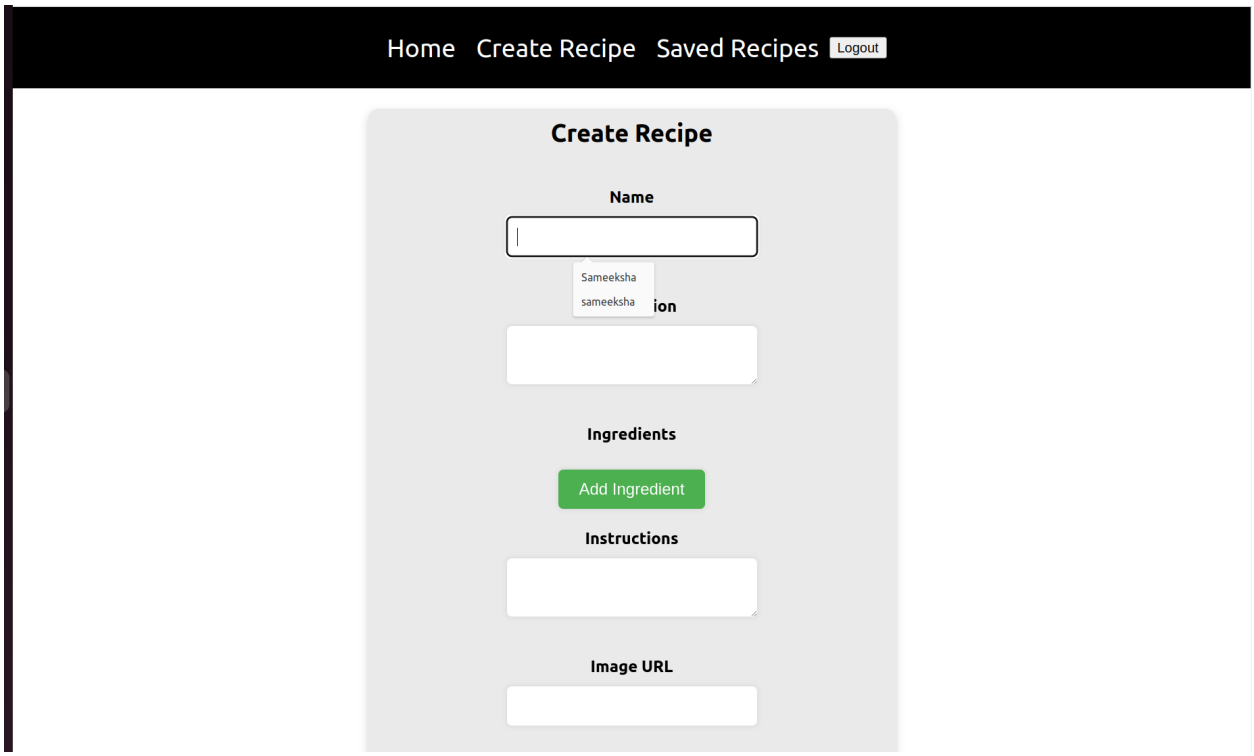
Add Rating

Once the user has given rating an alert message will pop up.



Create Recipe

User can create new recipe and give its description.



DEVOPS:

DockerFile (Client):

```
FROM node:14-alpine

# Set the working directory to /frontend
WORKDIR /frontend

# Copy the package.json and package-lock.json files to the container
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the frontend code to the container
COPY . .

# Build the production-ready code
RUN npm run build

# Expose the port that the application will run on
EXPOSE 2004

# Start the frontend server
CMD ["npm", "start"]
```

- This is a Dockerfile that can be used to build a container image for a frontend web application that uses Node.js as the runtime environment. The Dockerfile starts with a base image of Node.js version 14 running on Alpine Linux, a lightweight distribution.
- The Dockerfile sets the working directory to `/frontend` and copies the `package.json` and `package-lock.json` files into the container. Then it installs the dependencies using `npm install`.
- After that, it copies the rest of the frontend code into the container and runs the `npm run build` command to build the production-ready code.
- The Dockerfile then exposes port `2004` on the container and starts the frontend server by running the `npm start` command.

To build a container image using this Dockerfile, navigate to the directory containing the Dockerfile and run the following command:

docker build -t frontend .

This will create a Docker image named **`frontend`** based on the Dockerfile in the current directory.To run a container using this image, use the following command:

docker run -p 2004:2004 frontend

DockerFile(Server):

```
FROM node:14-alpine

# Set the working directory to /backend
WORKDIR /backend

# Copy the package.json and package-lock.json files to the container
COPY package*.json ./

# Install the dependencies
RUN npm install

# Copy the rest of the backend code to the container
COPY . .

# Expose the port that the application will run on
EXPOSE 8082

# Start the backend server
CMD ["npm", "start"]
```

- This is a Dockerfile that can be used to build a container image for a backend web application that uses Node.js as the runtime environment. The Dockerfile starts with a base image of Node.js version 14 running on Alpine Linux, a lightweight distribution.
- The Dockerfile sets the working directory to **`/backend`** and copies the **`package.json`** and **`package-lock.json`** files into the container. Then it installs the dependencies using **`npm install`**.
- After that, it copies the rest of the backend code into the container.
- The Dockerfile then exposes port **`8082`** on the container and starts the backend server by running the **`npm start`** command.

To build a container image using this Dockerfile, navigate to the directory containing the Dockerfile and run the following command:

docker build -t backend .

This will create a Docker image named **backend** based on the Dockerfile in the current directory.

To run a container using this image, use the following command:

docker run -p 8082:8082 backend

This will start a container running the **backend** image and forward traffic from port 8082 on the host machine to port 8082 on the container.

Kubernetes:

Recipe-backend-secret.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: recipe-backend-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: recipe-backend
 template:
   metadata:
     labels:
       app: recipe-backend
   spec:
     containers:
       - name: recipe-backend-container
         image: akashanand842/recipe-backend
         ports:
           - name: http
             containerPort: 8082
         env:
           - name: MONGODB_URI
             valueFrom:
               secretKeyRef:
                 name: recipe-backend-secret
                 key: MONGODB_URI
           - name: PORT
             valueFrom:
               secretKeyRef:
                 name: recipe-backend-secret
```

```
                key: PORT
---
apiVersion: v1
kind: Service
metadata:
 name: recipe-backend-service
spec:
 # type: LoadBalancer
 selector:
   app: recipe-backend
 ports:
   - name: http
     protocol: TCP
     port: 8082
     targetPort: 8082
     # nodePort: 30001
```

- This is a Kubernetes YAML configuration file that defines a secret named **`recipe-backend-secret`**.
- The **`metadata`** section specifies the name of the secret, and the **`type`** section specifies that it is an **`Opaque`** secret, meaning that its data is not encoded in any specific format.
- The **`data`** section contains two key-value pairs. The first key-value pair is **`PORT: ODA4Mg==`**. Here, the **`PORT`** key represents the port number that the backend service will listen on, and **`ODA4Mg==`** is the base64-encoded value of the port number **`8082`**.
- The second key-value pair is **`MONGODB_URI: bW9uZ29kYitzcnY6Ly9ha2FzaDpBa0FzaDAyQHJlY2lwZS1kYi4zNWp2a2RvLm1vbmdvZGIubmV0L3JlY2lwZS1kYj9yZXRyeVdyaXRlcz10cnVlJnc9bWFqb3JpdHk=`**. Here, the **`MONGODB_URI`** key represents the connection string for the MongoDB database, and **`bW9uZ29kYitzcnY6Ly9ha2FzaDpBa0FzaDAyQHJlY2lwZS1kYi4zNWp2a2RvLm1vbmdvZGIubmV0L3JlY2lwZS1kYj9yZXRyeVdyaXRlcz10cnVlJnc9bWFqb3JpdHk=`** is the base64-encoded value of the actual connection string **`mongodb+srv://<username>:<password>@recipe-db.35jkdro.mongodb.net/recipe-db?retryWrites=true&w=majority`**.

This secret will be used by the backend service deployment to configure the database connection and other backend settings.

Recipe-backend.yaml

```
apiVersion: apps/v1

kind: Deployment
```

```
metadata:
 name: recipe-backend-deployment
spec:
 replicas: 1
 selector:
   matchLabels:
     app: recipe-backend
 template:
   metadata:
     labels:
       app: recipe-backend
   spec:
     containers:
       - name: recipe-backend-container
         image: akashanand842/recipe-backend
         ports:
           - name: http
             containerPort: 8082
         env:
           - name: MONGODB_URI
             valueFrom:
               secretKeyRef:
                 name: recipe-backend-secret
                 key: MONGODB_URI
           - name: PORT
             valueFrom:
```

```
              secretKeyRef:
                name: recipe-backend-secret
                key: PORT
---
apiVersion: v1
kind: Service
metadata:
 name: recipe-backend-service
spec:
 # type: LoadBalancer
 selector:
   app: recipe-backend
 ports:
   - name: http
     protocol: TCP
     port: 8082
     targetPort: 8082
     # nodePort: 30001
```

- The manifest defines the Pod template that is used to create the replicas, including the container specification.
- The container specification specifies the container name **`recipe-backend-container`** and the image to use **`akashanand842/recipe-backend`.** It also exposes the port **`8082`** on the container, sets environment variables for the MongoDB connection URI and the port number, which are retrieved from the **`recipe-backend-secret`** Secret.

- The Service is named `recipe-backend-service` and specifies that it should target the Pods that match the `app: recipe-backend` label. It exposes port `8082` and forwards traffic to the container port `8082`.

To apply this manifest file, save it as a YAML file (e.g. `recipe-backend.yaml`) and run the following command:

kubectl apply -f recipe-backend.yaml

This will create the Deployment and Service resources on the Kubernetes cluster.

Recipe-frontend.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: recipe-frontend-deployment
 labels:
   app: recipe-frontend
spec:
 selector:
   matchLabels:
     app: recipe-frontend
 template:
   metadata:
     labels:
       app: recipe-frontend
   spec:
     containers:
       - name: recipe-frontend
         image: akashanand842/recipe-frontend
         ports:
```

```
          - containerPort: 2004
---


apiVersion: v1
kind: Service
metadata:
 name: recipe-frontend-service


spec:
 type: LoadBalancer
 selector:
   app: recipe-frontend
 ports:
   - name: http
     port: 2004
     targetPort: 2004
     # nodePort: 30000
```

This is a Kubernetes YAML configuration file that defines a frontend deployment and a frontend service.

- The deployment is defined using the `apps/v1` API version and the `Deployment` kind. The `metadata` section specifies the name of the deployment and the labels used to identify the deployment. The `spec` section specifies the details of the deployment, including the selector to match the pods controlled by this deployment, the container image to use, and the port to expose. In this case, the deployment will use the `akashanand842/recipe-frontend` Docker image, and will expose port `2004`.

- The service is defined using the `v1` API version and the `Service` kind. The `metadata` section specifies the name of the service. The `spec` section specifies the details of the service, including the service type, the selector to match the pods controlled by this service, and the ports to expose. In this case, the service type is `LoadBalancer`, which exposes the service externally using a cloud provider's load balancer. The service will match the pods controlled by the frontend deployment, and will expose port `2004`.
- The `nodePort` field is commented out, indicating that the node port will be assigned dynamically by the Kubernetes cluster. If you uncomment this field and specify a specific port number, the service will also be accessible on that port number on each node of the Kubernetes cluster.

Ansible:
Inventory file

```
[localhost]
akash ansible_connection=local
```

This is an Ansible inventory file that specifies a single host named `akash`, with `ansible_connection=local` indicating that the connection will be made to the local machine running the Ansible playbook.

In this case, `localhost` is not defined as a group, so the playbook will only be applied to the `akash` host. This is useful when you want to apply a playbook to only a specific machine or a group of machines with similar characteristics, such as similar hardware configurations or shared roles.

Playbook.yml file

```
---
- name: Deploy recipes app to minikube
 hosts: all
 tasks:
   - name: delete previous frontend deployment
     shell: kubectl delete -f ../kubernetes-config/recipe-frontend.yaml

   - name: delete previous backend deployment
```

```
    shell: kubectl delete -f ../kubernetes-config/recipe-backend.yaml

  - name: delete previous secret
    shell: kubectl delete -f ../kubernetes-config/recipe-backend-secret.yaml

  - name: apply secret
    shell: kubectl apply -f ../kubernetes-config/recipe-backend-secret.yaml

  - name: apply backend
    shell: kubectl apply -f ../kubernetes-config/recipe-backend.yaml

  - name: apply frontend
    shell: kubectl apply -f ../kubernetes-config/recipe-frontend.yaml
```

This is a YAML script that deploys a recipe application to minikube.

The script has a single playbook with several tasks, each with a name and a shell command.

The first three tasks are responsible for deleting any previous deployments and secrets from the minikube environment. The fourth task applies a new secret to the minikube environment. The fifth task applies a backend deployment, and the final task applies a frontend deployment.

Overall, this script is used to ensure that the latest version of the recipe application is deployed to minikube, by first deleting any old versions and then applying the new version with the updated configurations.

JenkinsFile:

```
pipeline {

   environment{
            DOCKERHUB_CREDENTIALS = credentials('dockerhub')
       }
 agent any

 stages {
   stage('Git Pull') {
       steps {
           git branch: 'sameeksha', url:
'https://github.com/akashanand842/RecipeMedia-SPEMajor.git'
       }
   }

   stage('Build and Push Frontend Image') {
     environment {
       IMAGE_NAME = ''
     }
     steps {
       dir('client') {
         script {
           // sh 'docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .'
           // sh 'docker login -u ${DOCKER_HUB_USERNAME} -p
${DOCKER_HUB_PASSWORD}'
           // sh 'docker push ${IMAGE_NAME}:${IMAGE_TAG}'
           IMAGE_NAME=docker.build "akashanand842/recipe-frontend"
           docker.withRegistry('','docker-key'){
               IMAGE_NAME.push()
           }
         }
       }
     }
   }

   stage('Build and Push Backend Image') {
     environment {
       IMAGE_NAME = ''
     }
     steps {
       dir('server') {
         script {
           // sh 'docker build -t ${IMAGE_NAME}:${IMAGE_TAG} .'
           // sh 'docker login -u ${DOCKER_HUB_USERNAME} -p
${DOCKER_HUB_PASSWORD}'
           // sh 'docker push ${IMAGE_NAME}:${IMAGE_TAG}'
           IMAGE_NAME=docker.build "akashanand842/recipe-backend"
```

```
            docker.withRegistry('','docker-key'){
                IMAGE_NAME.push()
            }
          }
        }
      }
    }

    stage('Deploy with Ansible') {
      steps {
        // withCredentials([usernamePassword(credentialsId: 'user-id',
usernameVariable: 'SSH_USER', passwordVariable: 'SSH_PASS')]) {
        //    sh 'echo "AKASH_USER=$SSH_USER"'
        //    sh 'echo "AKASH_PASS=$SSH_PASS"'
        //    script {
        //      ansiblePlaybook becomeUser: null, colorized: true,
disableHostKeyChecking: true, installation: 'Ansible', inventory:
'ansible-deploy/inventory',
        //      playbook: 'ansible-deploy/ansible-book.yml', sudoUser: null
        //    }
        // }
        script{
          sh ' ansiblePlaybook becomeUser: null, colorized: true,
disableHostKeyChecking: true, installation: 'Ansible', inventory:
'ansible-deploy/inventory',
            playbook: 'ansible-deploy/ansible-book.yml', sudoUser: null'
        }
      }
    }
  }
```

Jenkins Pipeline

This is a Jenkins Pipeline script that has four stages:

1. **`Git Pull`**: pulls the code repository from the specified URL and branch.
2. **`Build and Push Frontend Image`:** builds and pushes the frontend Docker image to a Docker registry. The **`docker.build`** command builds the image and assigns it to the **`IMAGE_NAME`** variable, and the **`docker.withRegistry`** command pushes the image to the registry. The registry is authenticated using a Docker Hub username and password, which are stored in Jenkins environment variables.
3. **`Build, Test and Push Backend Image`**: builds and pushes the backend Docker image to a Docker registry. This stage is similar to the previous one but builds and pushes the backend image instead.
4. **`Deploy with Ansible`:** deploys the application to the target environment using Ansible. The **`ansiblePlaybook`** command runs an Ansible playbook on the target

hosts. The playbook and inventory files are specified in the command, and the `sudoUser` and `becomeUser` options are set to null, which means that no user switching is performed during the playbook execution.

To use this script, copy it to a Jenkins Pipeline job, replace the Git repository URL and Docker registry credentials with your own, and adjust the Ansible inventory and playbook files to match your deployment environment.

Logging

- Creating logger using Winston Library.
- The logger is configured to output log messages to both the console and a file named "server.log".
- The logger has two transports: the Console transport outputs log messages to the console, and the File transport outputs log messages to a file named **"server.log"**.


```
{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T06:31:15.421Z"}

{"level":"info","message":"Recipes returned saved for the
user","timestamp":"2023-05-09T06:31:15.777Z"}

{"level":"info","message":"All Recipes
returned","timestamp":"2023-05-09T06:31:16.092Z"}

{"level":"info","message":"Recipes returned saved for the
user","timestamp":"2023-05-09T06:31:17.153Z"}

{"level":"info","message":"All Recipes
returned","timestamp":"2023-05-09T06:31:17.433Z"}

{"level":"info","message":"Recipe returned:
","timestamp":"2023-05-09T06:31:45.233Z"}

{"level":"info","message":"Recipe returned:
","timestamp":"2023-05-09T06:31:45.243Z"}

{"level":"info","message":"Recipes returned saved for the
user","timestamp":"2023-05-09T06:31:49.488Z"}

{"level":"info","message":"Recipes returned saved for the
user","timestamp":"2023-05-09T06:31:49.511Z"}

{"level":"info","message":"All Recipes
returned","timestamp":"2023-05-09T06:31:49.773Z"}
```

```
{"level":"info","message":"All Recipes
returned","timestamp":"2023-05-09T06:31:49.795Z"}

{"level":"info","message":"User added:
{\"username\":\"audd\",\"password\":\"$2b$10$ngizuLXDedFoRiy8aFuM8.IY3Hxg8OOoXd
EaDBsH2kjPaa9lAcUyW\",\"savedRecipes\":[],\"_id\":\"645a2f75f44a9180fcc9d901\"}
","timestamp":"2023-05-09T11:33:09.090Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$DVNaTMZ45ZWC4SwuxokTWuSu0uQit61zmC/
26E7GJrGRWTSS9p7He\",\"savedRecipes\":[],\"_id\":\"645a2fe8e8248b11a07c3617\"}"
,"timestamp":"2023-05-09T11:35:04.680Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$2bicbbxf/mSDwwb3r9GlO.niapvCr7wb1MY
txHkVG/E/cAH1tX2TG\",\"savedRecipes\":[],\"_id\":\"645a3a1ad3a49e7a0e87d9f5\"}"
,"timestamp":"2023-05-09T12:18:34.480Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T12:18:35.106Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$rBkIpIIG52bw6tw72JEG3u1csRqPjdqFQG6
8xrPCVcHXpODIydTZ2\",\"savedRecipes\":[],\"_id\":\"645a40fe6d46d4d94951ec1b\"}"
,"timestamp":"2023-05-09T12:47:58.320Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T12:47:59.043Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T12:56:51.821Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$BErFic/Hh9dRFrKdTZ0Y8uOzMDVm25V9Y8s
CqVNUlrz8FvQfWjSyO\",\"savedRecipes\":[],\"_id\":\"645a43145b4f5680fbc1c055\"}"
,"timestamp":"2023-05-09T12:56:52.217Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T12:56:52.903Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T13:04:43.030Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$763vI5wcXu7tMkxx2dyRIe0kZ8zxovNAxjr
Xdsg.MgCR5/ckd76ty\",\"savedRecipes\":[],\"_id\":\"645a44ebd42156251dc5b627\"}"
,"timestamp":"2023-05-09T13:04:43.455Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T13:04:44.183Z"}
```

```
{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T13:12:30.018Z"}

{"level":"info","message":"User added:
{\"username\":\"udd\",\"password\":\"$2b$10$V.C.I6fvPlmM9IS1WI.GIOwXyFB4ihAvKjp
QLDn86EsPNMLrdh3gy\",\"savedRecipes\":[],\"_id\":\"645a46beee6881235eb40c8d\"}"
,"timestamp":"2023-05-09T13:12:30.403Z"}

{"level":"info","message":"User logged in
:","timestamp":"2023-05-09T13:12:30.996Z"}
```

Testing

Backend

Recipe API

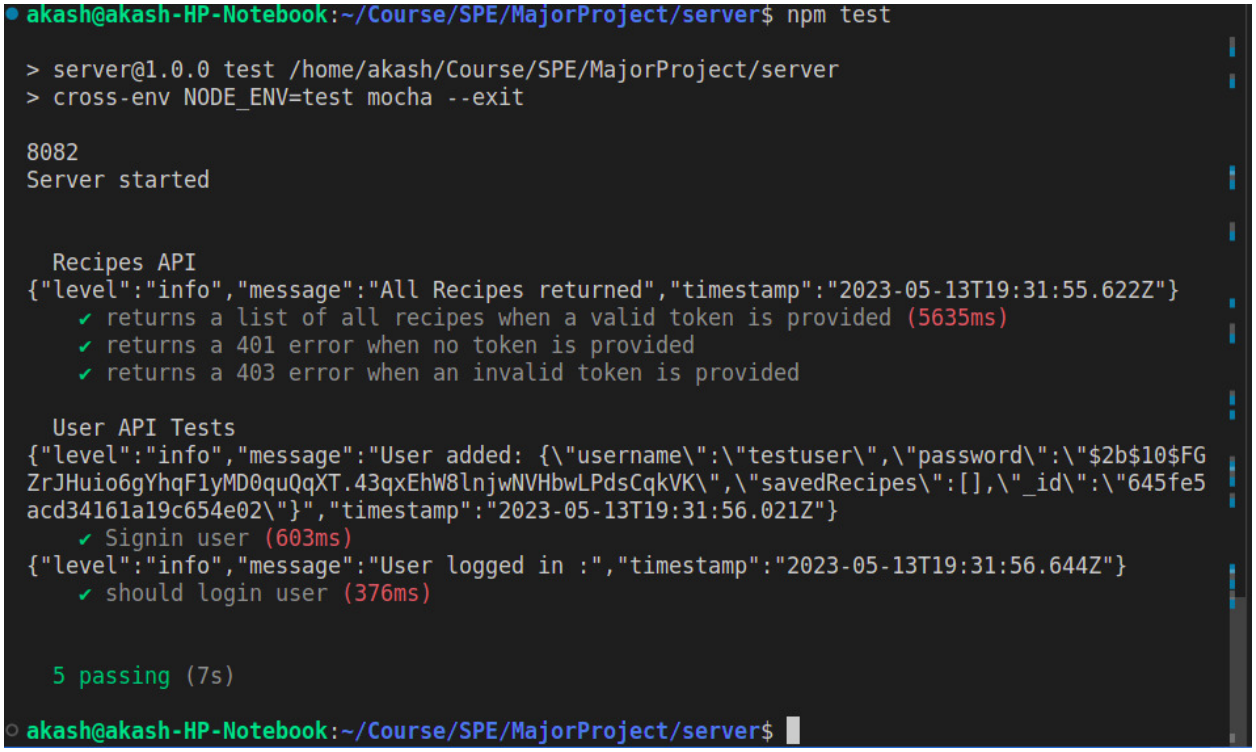
- This code is a Mocha test suite that tests the Recipes API. It includes two tests that check if the API returns the correct error codes when an authentication token is missing or invalid.
- The `before` and `after` hooks are used to execute code before and after the tests, respectively. However, in this code, these hooks are empty and do not perform any actions.
- The first test sends a GET request to the `/api/recipes/` endpoint without providing an authentication token. The test checks if the server responds with a 401 Unauthorized error, which is the expected behavior in this case.
- The second test sends a GET request to the same endpoint but with an invalid authentication token in the `Authorization` header. The test checks if the server responds with a 403 Forbidden error, which indicates that the token is invalid.
- Overall, this code tests the authentication functionality of the Recipes API and ensures that it behaves correctly when receiving requests with missing or invalid authentication tokens.

User API

It uses the Mocha testing framework and the Chai assertion library to perform unit tests on the endpoints of a User API.

- The first line sets the `NODE_ENV` environment variable to `'test'`.
- Next, it imports the necessary dependencies such as `mongoose`, `chai`, `chai-http`, and the `UserModel` from the User API.
- It then sets up the Mocha `before` and `after` hooks to execute any code that needs to be run before and after the tests are executed.
- The `describe` block defines a test suite for the User API.
- Inside the `describe` block, there are two `it` blocks that define individual tests.
- The first test is for registering a new user, and the second test is for logging in an existing user.
- Both tests use the `chai-http` library to make HTTP requests to the User API endpoints.
- The tests use the `should` and `expect` assertion methods from Chai to validate the responses from the API.

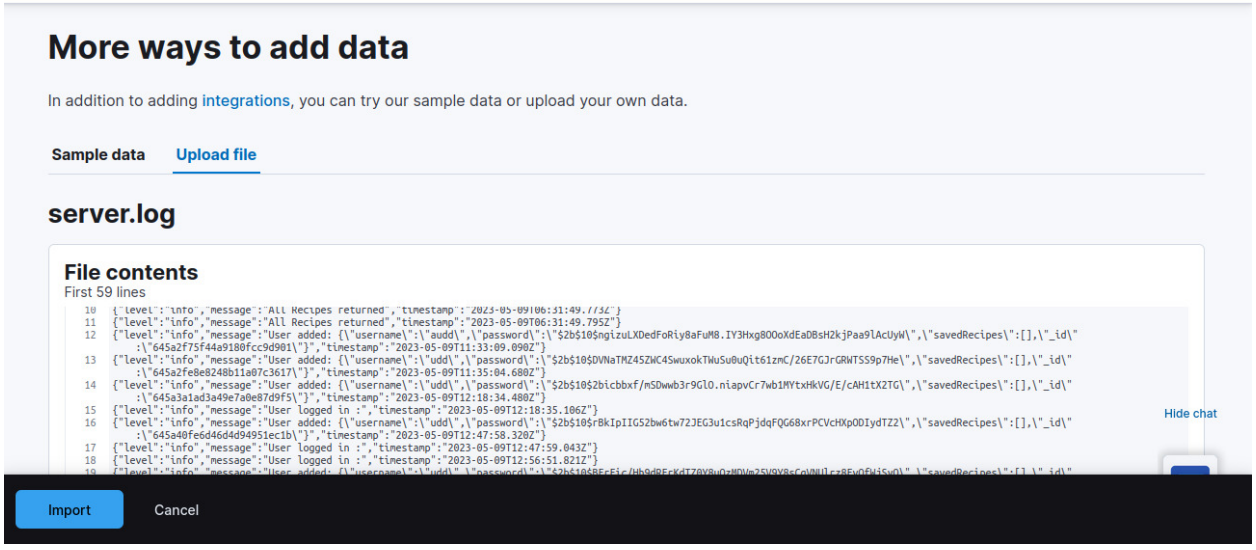
Overall, this code sets up a basic test suite for a User API and tests its endpoints using the Mocha and Chai libraries.

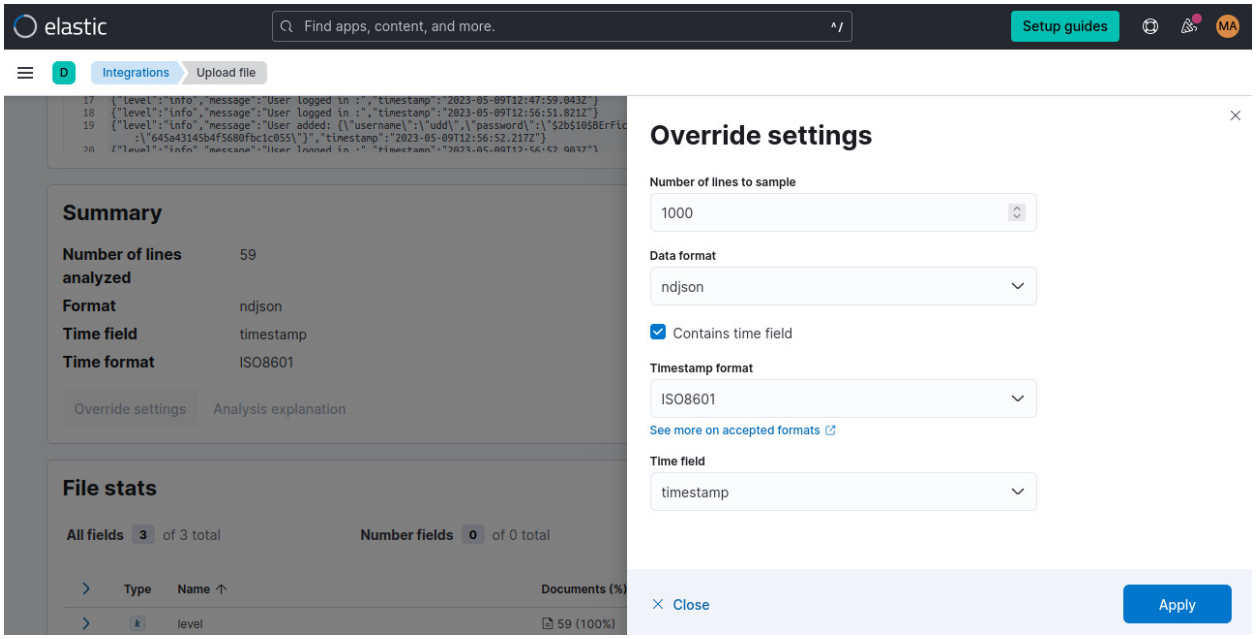

akash@akash-HP-Notebook:~/Course/SPE/MajorProject/server$ npm test
> server@1.0.0 test /home/akash/Course/SPE/MajorProject/server
> cross-env NODE_ENV=test mocha --exit
8082
Server started
Recipes API
{"level":"info","message":"All Recipes returned","timestamp":"2023-05-13T19:31:55.622Z"}
✔ returns a list of all recipes when a valid token is provided (5635ms)
✔ returns a 401 error when no token is provided
✔ returns a 403 error when an invalid token is provided
User API Tests
{"level":"info","message":"User added: {\"username\":\"testuser\",\"password\":\"$2b$10$FG
ZrJHuio6gYhqF1yMD0quQqXT.43qxEhW8lnjwNVHbwLPdsCqkVK\",\"savedRecipes\":[],\"_id\":\"645fe5
acd34161a19c654e02\"}","timestamp":"2023-05-13T19:31:56.021Z"}
✔ Signin user (603ms)
{"level":"info","message":"User logged in :","timestamp":"2023-05-13T19:31:56.644Z"}
✔ should login user (376ms)
5 passing (7s)
akash@akash-HP-Notebook:~/Course/SPE/MajorProject/server$

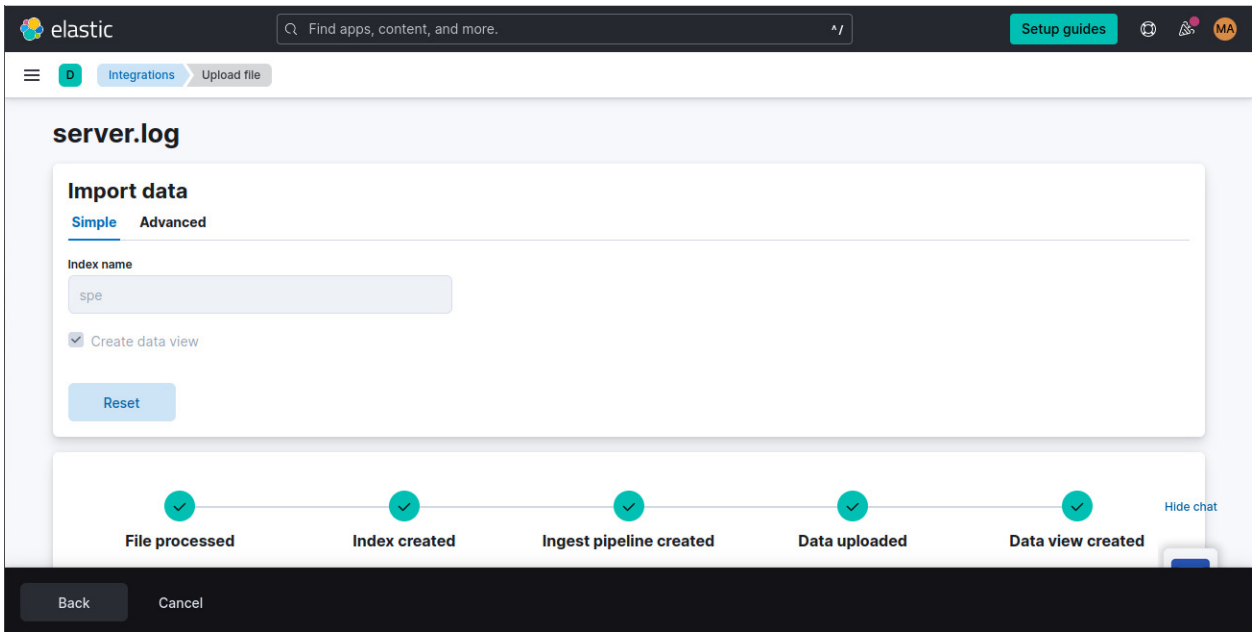
ELK(Elastic Logstash Kibana)

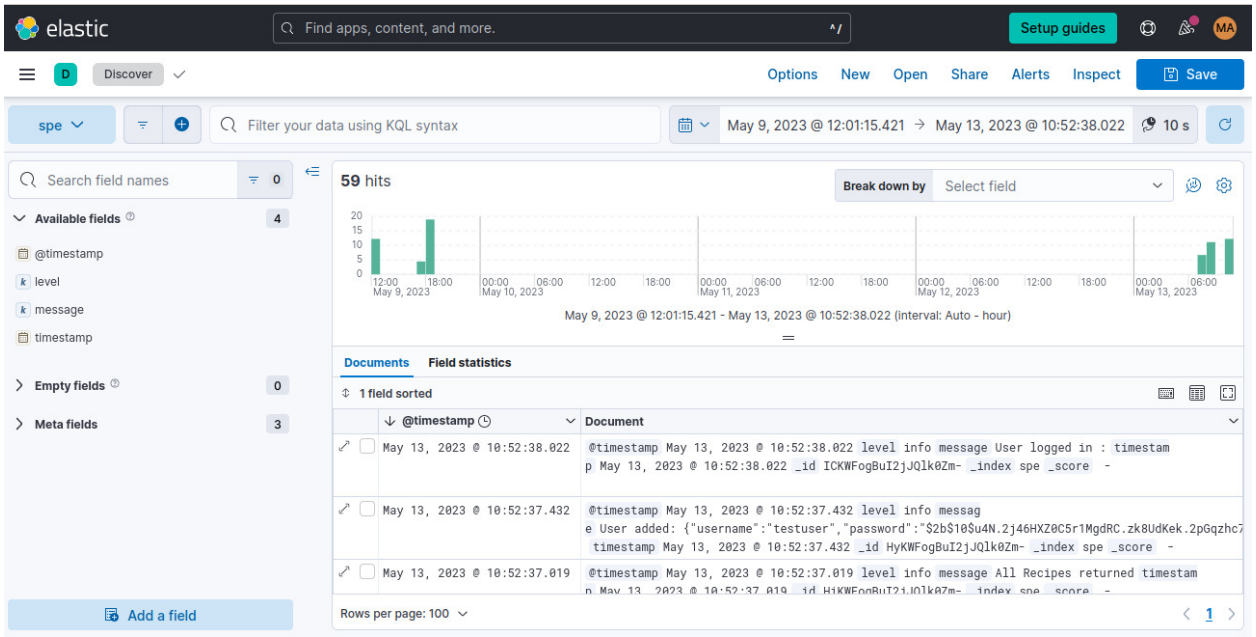
We will get server.log file from kubernetes backend pod using these commands.

- **kubectl cp recipe-backend-deployment-849b4d7cf9-n5q6x:/backend/logs /home/iiitb**
- Using this command log file will be present inside **/home/iiitb**






elastic
Find apps, content, and more.
Setup guides
MA
D
Integrations
Upload file
server.log
Import data
Simple
Advanced
Index name
spe
Create data view
Reset
File processed
Index created
Ingest pipeline created
Data uploaded
Data view created
Hide chat
Back
Cancel


elastic
Find apps, content, and more.
Setup guides
MA
D
Discover
Options
New
Open
Share
Alerts
Inspect
Save
spe
Filter your data using KQL syntax
May 9, 2023 @ 12:01:15.421
May 13, 2023 @ 10:52:38.022
10 s
Search field names
Available fields
4
@timestamp
level
message
timestamp
Empty fields
0
Meta fields
3
Add a field
59 hits
Break down by
Select field
May 9, 2023 @ 12:01:15.421 - May 13, 2023 @ 10:52:38.022 (interval: Auto - hour)
Documents
Field statistics
1 field sorted
@timestamp
Document
May 13, 2023 @ 10:52:38.022
May 13, 2023 @ 10:52:37.432
May 13, 2023 @ 10:52:37.019
Rows per page: 100