COURSE NAME

SOFTWARE PRODUCTION ENGINEERING

Raghunadh(IMT2021042)

Kushal(IMT2021035)

Introduction

This is a book-store application is an innovative platform that bridges the gap between book enthusiasts and a seamless online shopping experience. Designed with dual-role functionality, it allows users to log in as customers to explore a diverse range of books, dynamically update their shopping cart, and proceed to checkout for payment. Users can view their orders in their dashboard.Simultaneously, administrators can manage the inventory by logging in to add, update, or remove books available in the store. With an intuitive interface and a user-friendly dashboard, customers can conveniently view and manage their past orders, making the application a comprehensive solution for modern online book retail. You can find the source code at book-store

DevOps tools:

Source Control Management: Git and GitHub

Continuous Integration Pipeline: Jenkins

Containerization: Docker

Container Orchestration: Docker composeFront End: React, Tailwind CSS

Logger: Winston

Monitoring: ELK Stack

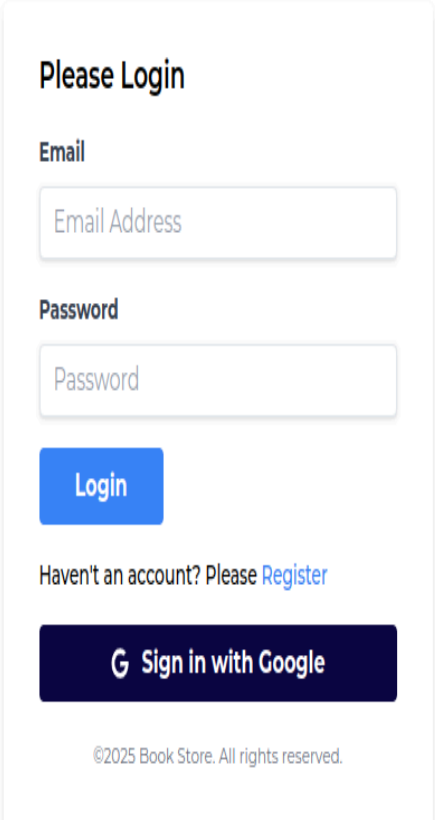
Database: MongoDB

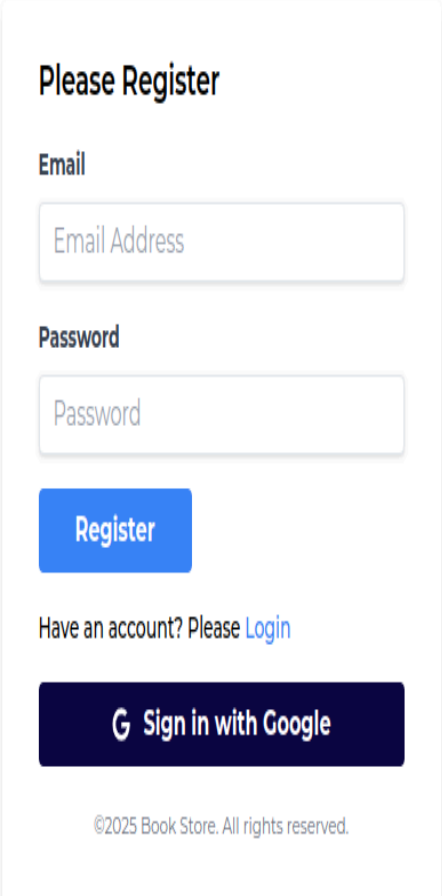
Other Dependencies:

- **express**: A web framework for building RESTful APIs and handling HTTP requests and responses.
- **cors**: Enables cross-origin requests, allowing your API to be accessed from different domains.
- **mongoose**: A library for MongoDB to define schemas and interact with the database in an object-oriented way.
- **jsonwebtoken**: Used for creating and verifying JSON Web Tokens (JWTs) for user authentication.
- **Firebase:** Firebase provides a comprehensive suite of backend services, including real-time databases, authentication, hosting, and cloud storage, enabling developers to easily build, manage, and scale web and mobile applications.

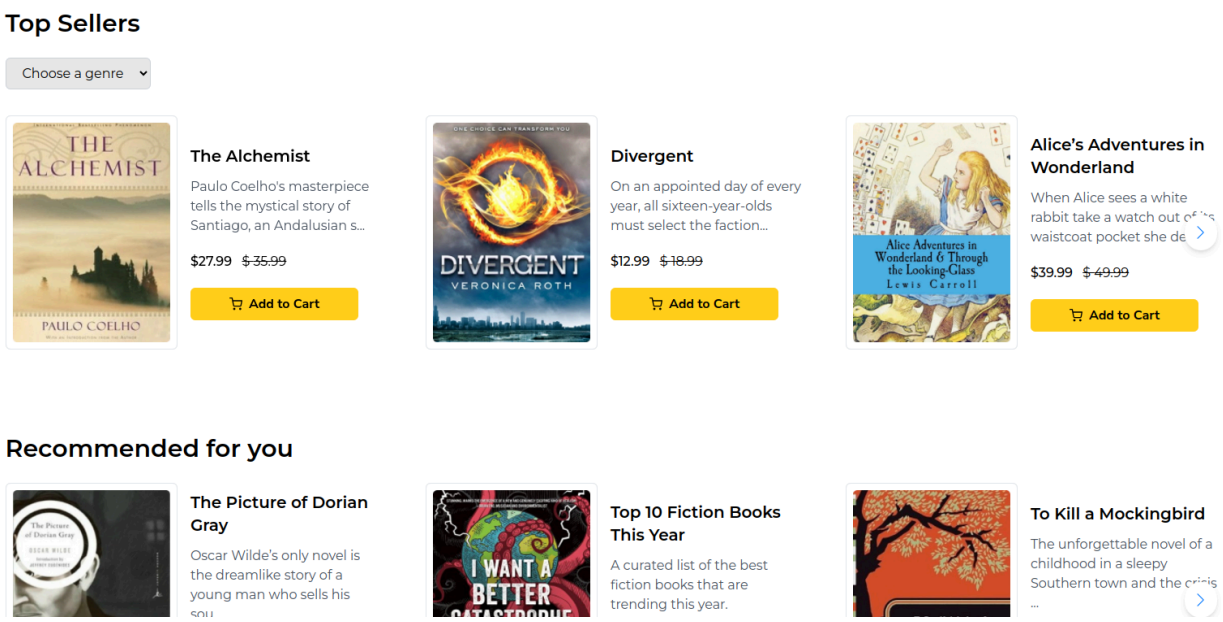
Features in the Application

- **Register**
    - New users have to first sign up by either registering through their email or signing in using their google account.

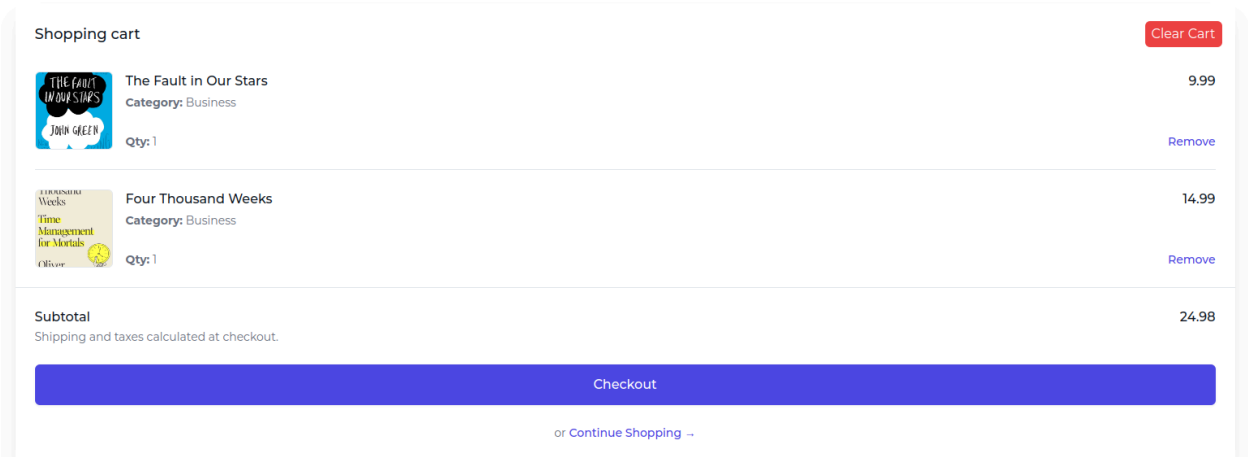




- **Login**
  - Users can login via email or signing in with google.
- **Home Page**
  - It has the top sellers and their books along with the books recommended for you.
  - You can filter the books based on your favourite genre.
  - You can add books to the cart here

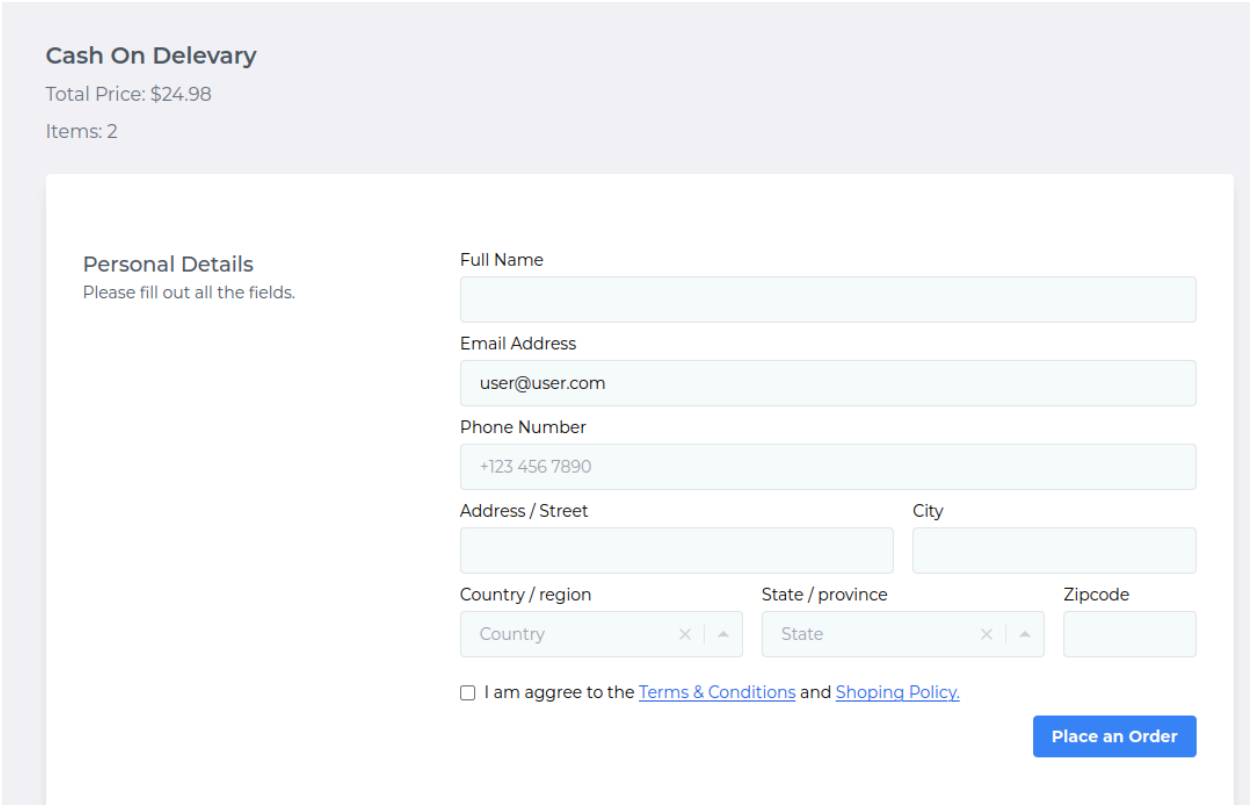


- **Cart**
    - It consists of books we have added to the cart.
    - We can dynamically remove books from the cart
    - We can also clear the cart if needed
    - When we feel that the cart contains books which we want to buy, we can checkout and place the order.

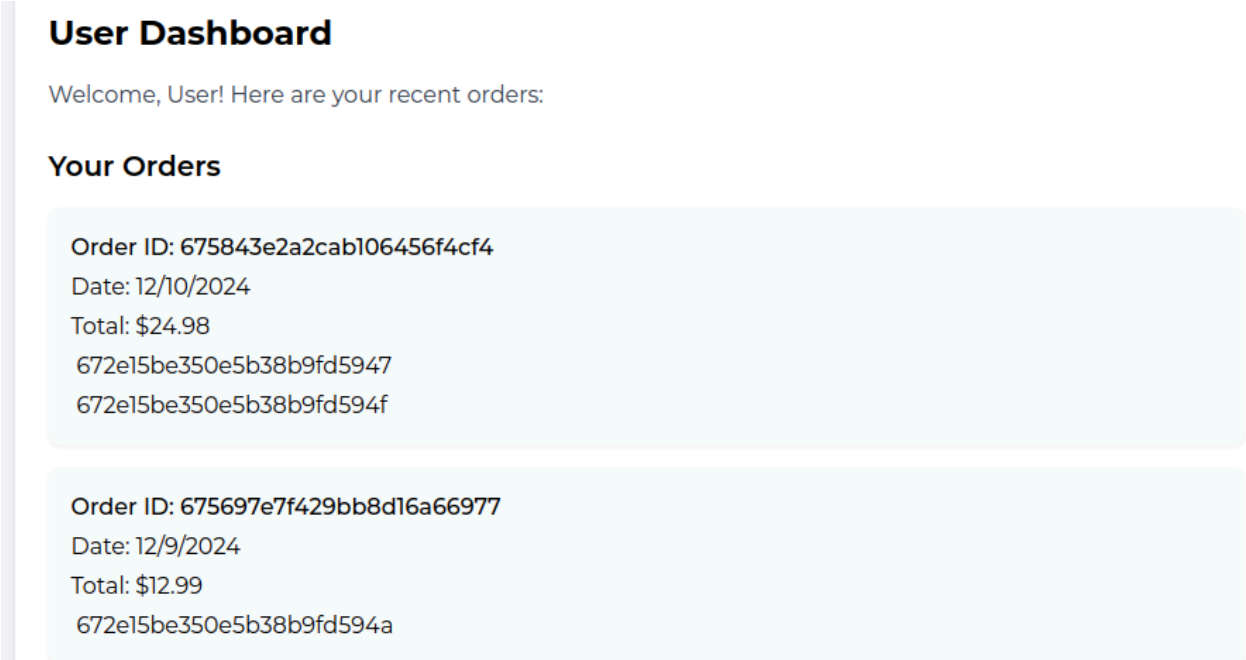


- **Checkout**

- We have to fill in user name, phone number, address, city, state, country, zip code.
- Finally, we have to place the order to get order successfully.



- **User Dashboard**
  - We can view the orders of present user in user dashboard

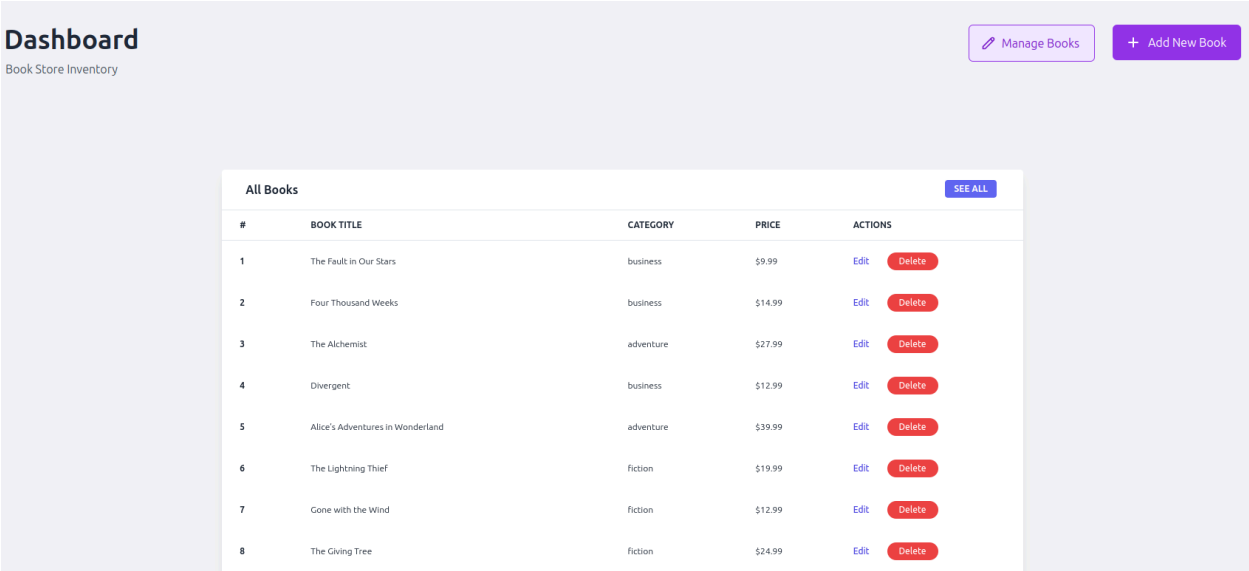


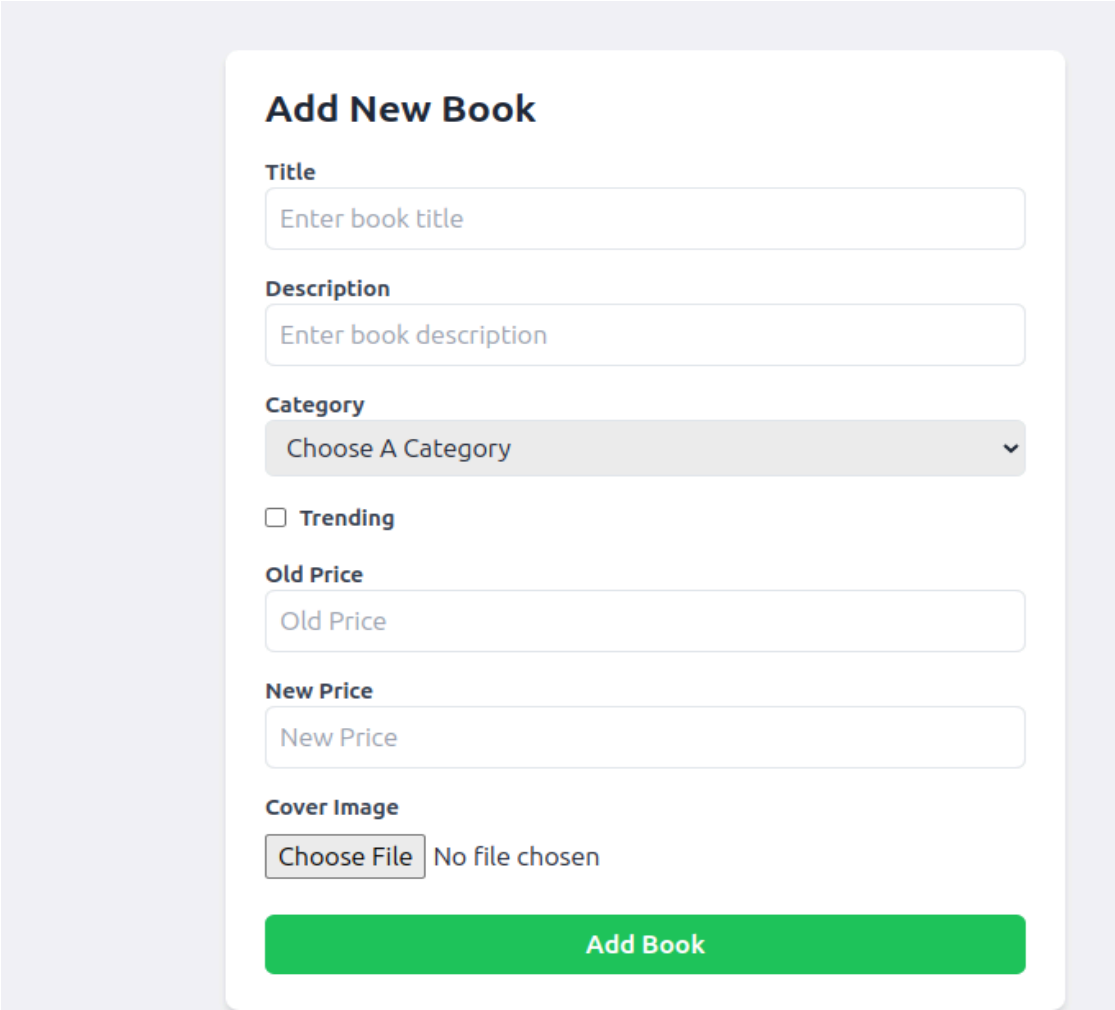
These are the features of user.

Admin features

When we login as admin, we can manage books by adding, updating and deleting books via admin's dashboard.

- By clicking on delete, we can delete the book.
- On going to add new book, we will be redirected to a new page where we have to fill in details like book title, description, price along with cover image and can add a new book.


Dashboard
Book Store Inventory
Manage Books
Add New Book
All Books
SEE ALL
BOOK TITLE CATEGORY PRICE ACTIONS
1 The Fault in Our Stars business $9.99 Edit Delete
2 Four Thousand Weeks business $14.99 Edit Delete
3 The Alchemist adventure $27.99 Edit Delete
4 Divergent business $12.99 Edit Delete
5 Alice's Adventures in Wonderland adventure $39.99 Edit Delete
6 The Lightning Thief fiction $19.99 Edit Delete
7 Gone with the Wind fiction $12.99 Edit Delete
8 The Giving Tree fiction $24.99 Edit Delete


Add New Book
Title
Enter book title
Description
Enter book description
Category
Choose A Category
Trending
Old Price
Old Price
New Price
New Price
Cover Image
Choose File
No file chosen
Add Book

Deploying using docker, docker compose and Kubernetes

Docker

We have both front end and back end files to run this application. So, we need to create 2 containers for both front end and back end.

```
ntend > Dockerfile
FROM node:21

# Create app directory
WORKDIR /app

# Install app dependencies
COPY ./package.json /app
COPY ./package-lock.json /app

RUN npm install
# If you are building your code for production
# RUN yarn install --production

# Bundle app source
COPY . /app

EXPOSE 5173
CMD [ "npm", "run", "dev" ]
```

Front end docker file

1. Uses `node:21` as the base image to leverage the latest Node.js version.
2. Sets `/app` as the working directory for all container operations.
3. Installs dependencies by copying `package.json` and running `npm install`.
4. Exposes port 5173 and runs the development server with `npm run dev`.

```
d > Dockerfile
FROM node:21

# Create app directory
WORKDIR /app

# Install nodemon globally
RUN npm install -g nodemon

# Install app dependencies
COPY ./package.json /app
COPY ./package-lock.json /app
RUN npm ci
# If you are building your code for production
# RUN npm ci --only=production

# Bundle app source
COPY . /app

# Expose the application port
EXPOSE 5000

# Run nodemon to start the app with index.js
CMD ["nodemon", "index.js"]
```

Back end docker file

- Uses `node:21` as the base image to run the latest Node.js version.
- Installs `nodemon` globally to enable automatic restarts during development.
- Copies `package.json` and `package-lock.json` before running `npm ci` to install dependencies with a clean slate.
- Exposes port `5000` and uses `nodemon` to run `index.js` for a development environment.

Jenkins

We used Jenkins pipeline scm from GitHub. We wrote the pipeline script which has multiple stages. The pipeline script was cloned from the GitHub repository and the code was also cloned from the same repository.

```
stages {
    stage('Step 1 :Clone Git') {
        steps {
            git branch: 'main',
            url: 'https://github.com/Raghunadh2004/book-store.git'
        }
    }

    stage('Step 2 : Frontend build') {
        steps {
            dir('frontend'){
            sh "npm install"
            sh 'docker build -t frontend-image .'
        }
        }
    }

    stage("Step 3 : Backend build") {
        steps {
            dir('backend'){
            sh "npm install"
            sh 'docker build -t backend-image .'
        }}
    }

    stage('Step 4 : Push to Docker Hub') {
        steps {
            script {
                withCredentials([usernamePassword(credentialsId: 'DockerHubCred', usernameVariable: 'DOCKER_USERNAME', passwordVariable: 'DOCKER_PASSWORD')]
                    sh """
                        docker login --username ${DOCKER_USERNAME} --password ${DOCKER_PASSWORD}
                        docker tag frontend-image ${DOCKER_USERNAME}/frontend-image:latest
                        docker push ${DOCKER_USERNAME}/frontend-image:latest
                        docker tag backend-image ${DOCKER_USERNAME}/backend-image:latest
                        docker push ${DOCKER_USERNAME}/backend-image:latest
                        docker rmi backend-image
                        docker rmi frontend-image
                    """
                }
            }
        }
    }
```

```
        }
    }
    stage('Step 5 : Clean Docker Images'){
        steps{
            script{
                sh 'docker container prune -f'
                sh 'docker image prune -f'
            }
        }
    }
     stage('Step 6: Ansible Deployment') {
        steps {
            ansiblePlaybook(
                colorized: true,
                credentialsId: 'localhost',
                disableHostKeyChecking: true,
                installation: 'Ansible',
                inventory: 'inventory-k8',
                playbook: 'playbook-k8.yml',
                sudoUser: null
            )
        }
     }
```

Stage 1

- Clones the book-store repository from GitHub using the specified URL.
- Specifies the main branch for checkout, ensuring the latest stable code is used.
- Integrates GitHub with Jenkins for automated build triggers based on repository changes.
- Simplifies source control management by automatically pulling the latest code during the build process.

Stage 2

- Navigates to the frontend directory to run build steps.
- Installs frontend dependencies using npm install.
- Builds a Docker image for the frontend and tags it as frontend-image.
- Ensures an isolated and consistent frontend build process.

Stage 3

- Changes to the `backend` directory to execute build steps.
- Installs backend dependencies with `npm install`.
- Builds a Docker image for the backend and tags it as `backend-image`.
- Ensures an isolated and consistent backend build process.

Stage 4

- Logs in to Docker Hub using credentials stored in `DockerHubCred`.
- Tags and pushes the `frontend-image` to Docker Hub as `latest`.
- Tags and pushes the `backend-image` to Docker Hub as `latest`.
- Removes local Docker images after pushing them to Docker Hub for cleanup.

Stage 5

- Removes unused Docker containers with `docker container prune -f`.
- Cleans up unused Docker images with `docker image prune -f`.
- Frees up disk space by removing unnecessary Docker resources.
- Images are built in this stage.

Stage 6

- Executes an Ansible playbook to deploy the application using `ansiblePlaybook`.
- Uses `localhost` credentials for authentication during deployment.
- Runs the playbook `playbook-k8.yml` with the specified inventory `inventory-k8`.

Stage View

	Declarative: Checkout SCM	Step 1 :Clone Git	Step 2 : Frontend build	Step 3 : Backend build	Step 4 : Push to Docker Hub	Step 5 : Clean Docker Images	Step 6: Ansible Deployment
Average stage times: (Average full run time: ~2min 5s)	2s	4s	19s	6s	1min 5s	539ms	4s
#85 Dec 10 17:40 1 commit	1s	898ms	10s	5s	1min 18s	660ms	6s

We are able to successfully build he complete project on jenkins and deploy the application.

Docker Compose

```yaml
cker-compose.yml
version: '3'
services:
  frontend:
    image: maturiraghu/frontend-image:latest
    restart: always
    ports:
      - '5173:5173'
    command: npm run dev --host
    environment:
      - VITE_APIKEY="AIzaSyD7ZwB8GVj5sn6NJ_HarqMUWYvfAmpBVFk"
      - VITE_AUTHDOMAIN="book-store-app-db422.firebaseapp.com"
      - VITE_PROJECTID="book-store-app-db422"
      - VITE_STORAGEBUCKET="book-store-app-db422.firebasestorage.app"
      - VITE_MESSAGINGSENDERID="295613074035"
      - VITE_APPID="1:295613074035:web:a96722a160f285bb69f806"
  backend:
    image: maturiraghu/backend-image:latest
    restart: always
    ports:
      - '5000:5000'
    environment:
      - DB_URL = "mongodb+srv://maturiraghunadh:PpCT2VGz3HfB7VKJ@cluster0.gxbg6.mongodb.net/book-store?retryWrites=true&w=majority&appName=Cluster0"
      - JWT_SECRET_KEY = "87245759fe82aacdacb9d9bd672170fee7266b0f24d3acc329e4a498288ae20f86a1f5dc12f9090fa73b488318f42df77ff94b50bf41ce0c773237269d814f03"
```

The file defines a Docker Compose configuration with two services: `frontend` and backend.

Frontend Service:

- Uses the image `maturiraghu/frontend-image:latest`.
- Exposes port 5173 to the host for frontend access.
- Runs the `npm run dev --host` command to start the React development server.
- Environment variables are set for Firebase configuration, allowing the frontend to interact with Firebase services (API key, project ID, messaging sender ID, etc.).

Backend Service:

- Uses the image `maturiraghu/backend-image:latest`.
- Exposes port `5000` for backend access.

- Environment variables are configured for MongoDB connection, including a connection string to a MongoDB Atlas database.
- Includes a `JWT_SECRET_KEY` for JWT token generation and authentication.

Running the docker compose file

We used an inventory file and playbook to run the docker compose file.

Inventory file

```
localhost ansible_connection=local ansible_user=raghunadh
ansible_python_interpreter=/usr/bin/python3
```

Playbook

```yaml
ybook.yml
---
- name: Deploy MERN Application
  hosts: all
  connection: local
  become: false
  vars:
    ansible_become_pass: "Raghu@2004"

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

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

- Deploys a MERN application using Ansible.
- Copies the docker-compose.yml file to the target machine.
- Runs docker-compose up -d to start the application in detached mode.

- Does not escalate privileges during execution.

When you open https://localhost:5173, you can see front end of the application now and back end runs at port 5000 in backend.

Deploying with Kubernetes

To deploy using Kubernetes, you need to create deployment and service files for both frontend and backend containers. Additionally, you should create a secrets file to store sensitive information, ensuring that the secrets are not visible in the main configuration files. We have created hpa files also for both frontend and backend to enable automated scaling of the application. We used ingress too.

Backend Deployment

- Deploys a single replica of the `backend` container using `maturiraghu/backend-image:latest`.
- Sets resource limits for memory (128Mi) and CPU (500m).
- Exposes port 5000 for HTTP traffic.
- Retrieves `MONGO` and `JWT` secrets from the `mern-backend-secret` Kubernetes secret.

- ```yaml
  backend-deployment.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: backend-deployment
    namespace: mern-app
  spec:
    selector:
      matchLabels:
        app: backend
    replicas: 1
    template:
      metadata:
        labels:
          app: backend
      spec:
        containers:
        - name: backend
          image: maturiraghu/backend-image:latest
          resources:
            limits:
              memory: "128Mi"
              cpu: "500m"
          ports:
          - name: http
            containerPort: 5000
          env:
          - name: MONGO
            valueFrom:
              secretKeyRef:
                name: mern-backend-secret
                key: MONGO
          - name: JWT
            valueFrom:
              secretKeyRef:
                name: mern-backend-secret
                key: JWT
  ```

Backend Service

- Creates a `Service` named `backend-service` in the `mern-app` namespace.
- Exposes the backend container using `NodePort` with port 5000.
- Maps `nodePort` to 30002 for external access to the service.
- Uses a selector to link the service to the backend deployment via the `app: backend` label.
  ```

```yaml
backend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: mern-app
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - name: http
      port: 5000
      targetPort: 5000
      nodePort: 30002
```

Frontend Deployment

- Deploys a single replica of the `frontend` container using the `maturiraghu/frontend-image:latest` image in the `mern-app` namespace.
- Exposes container port 5173 for the frontend service.
- Uses a selector to match the `frontend` label for targeting the correct pod.

```yaml
frontend-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  namespace: mern-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: maturiraghu/frontend-image:latest
          ports:
            - containerPort: 5173
```

Frontend Service

- Creates a `Service` named `frontend-service` in the `mern-app` namespace.
- Exposes the frontend container using `NodePort`, mapping internal port 5173 to external node port 30001.
- Uses a selector to link the service to the frontend deployment via the `app: frontend` label.

```
frontend-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: mern-app
spec:
  type: NodePort  # Ensure this is capitalized as
  selector:
    app: frontend
  ports:
    - name: http
      port: 5173        # Port exposed within the
      targetPort: 5173  # Port on the frontend con
      nodePort: 30001   # Port exposed on each Nod
```

Frontend and Backend HPA

- Defines a Horizontal Pod Autoscaler (HPA) for both the frontend and backend deployments in the `mern-app` namespace.
- The HPA targets the respective deployment (`frontend-deployment` and `backend-deployment`).
- Minimum replicas are set to 1, and maximum replicas are set to 3 for both frontend and backend.

- The scaling is based on CPU utilization, with the target average utilization set to 50%.

Ingress

- Creates an `Ingress` resource named `mern-ingress` in the `mern-app` namespace.
- Maps the domain `book-store.com` to the frontend and backend services.
- Routes traffic with the path `/` to the `frontend-service` on port 5173.
- Routes traffic with the path `/api` to the `backend-service` on port 5000.

mern-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: mern-ingress
  namespace: mern-app
  labels:
    name: fintrack-ingress
spec:
  rules:
    - host: book-store.com
      http:
        paths:
          - path: "/"
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 5173
          - path: "/api"
            pathType: Prefix
            backend:
              service:
                name: backend-service
                port:
                  number: 5000
```

Inventory and playbook files for deployment of kubernetes

entory-k8

```
[ansible_nodes]
localhost ansible_user=raghunadh ansible_python_interpreter=/usr/bin/python3

[ansible_nodes:vars]
ansible_connection=local
```

ybook-k8.yml

```
- name: Deploying with Kubernetes
  tasks:
    - name: Create Namespace

    - name: Apply Secrets
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/mern-backend-secret.yaml') | from_yaml }}"

    - name: Create Frontend Deployment
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/frontend-deployment.yaml') | from_yaml }}"

    - name: Create Frontend Service
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/frontend-service.yaml') | from_yaml }}"


    - name: Create Backend Deployment
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/backend-deployment.yaml') | from_yaml }}"

    - name: Create Backend Service
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/backend-service.yaml') | from_yaml }}"


    - name: Create Config Map
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/frontend-configmap.yaml') | from_yaml }}"

    - name: Create Ingress
      kubernetes.core.k8s:
        state: present
        definition: "{{ lookup('file', './k8/mern-ingress.yaml') | from_yaml }}"
```

- We have listed all the hosts in the inventory file and the files to be run in the playbook file.
- When we run the playbook file, we see pods getting created in the specified namespace.
- We can access the application with the minikube ip and port number which we have assigned.

Using Ngrok

- Used ngrok to expose the local server to the internet for GitHub webhook integration.
- Configured GitHub to send webhook events to the public ngrok URL.
- Enabled GitHub Hook Trigger with GIT SCM polling to trigger actions based on changes in the repository.
- This setup allowed for testing webhooks and automating actions on local development environments.

```
🔀 Route traffic by anything: https://ngrok.com/r/iep

Session Status                online
Account                       Raghunadh2004 (Plan: Free)
Version                       3.18.4
Region                        India (in)
Web Interface                 http://127.0.0.1:4040
Forwarding                    https://3f30-117-248-71-42.ngrok-free.app -> http:

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

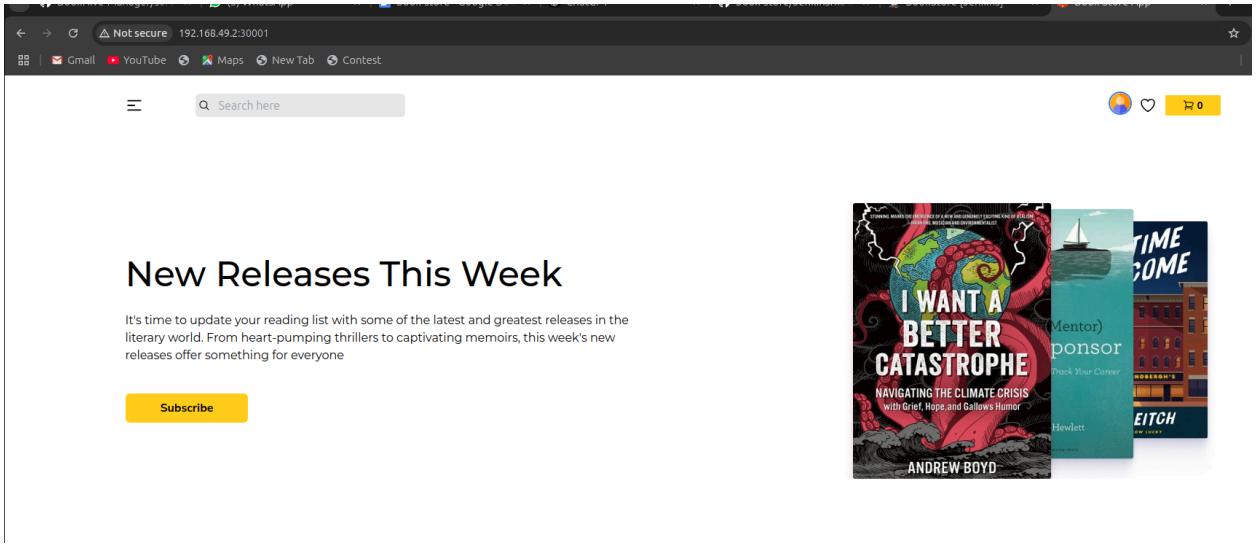
Accessing the application

```
raghunadh@raghunadh-83EM:~$ kubectl get pods -n mern-app
NAME                                  READY   STATUS    RESTARTS   AGE
backend-deployment-864d5db9c9-7ml6t   1/1     Running   0          4h11m
frontend-deployment-684d695f87-75wtc  1/1     Running   0          4h11m
raghunadh@raghunadh-83EM:~$
```

Pods are running in the mern-app namespace as shown above.

```
192.168.49.2
raghunadh@raghunadh-83EM:~$ minikube service frontend-service --url -n mern-app
http://192.168.49.2:30001
raghunadh@raghunadh-83EM:~$ minikube service backend-service --url -n mern-app
http://192.168.49.2:30002
raghunadh@raghunadh-83EM:~$
```

- We can access the application by using the above url.
- We can see the running application as shown below.
- By keeping the line **192.168.49.2 book-store.com,** we can access the application at http://book-store.com too.



Logging

- We have written custom logic to write logs into a file in the backend part.
- The format of log file is shown below.
- It consists of all backend requests and details about books and orders.


```
{"level":"info","message":"Request to get all books","timestamp":"2024-12-10 19:20:21"}
{"count":18,"level":"info","message":"Fetched all books successfully","timestamp":"2024-12-10 19:20:21"}
{"level":"info","message":"Order created successfully","orderDetails":{"__v":0,"_id":"67584755df130b96e67b5f91","address":{"city":"Bangalore ","country":"India"
{"email":"admin@admin.com","level":"info","message":"Orders retrieved for email","numberOfOrders":4,"timestamp":"2024-12-10 19:21:17"}
{"level":"info","message":"Request to get all books","timestamp":"2024-12-10 19:21:48"}
{"count":18,"level":"info","message":"Fetched all books successfully","timestamp":"2024-12-10 19:21:48"}
{"bookId":"672e15be350e5b38b9fd594f","level":"info","message":"Request to get a single book","timestamp":"2024-12-10 19:21:55"}
{"book":{"_id":"672e15be350e5b38b9fd594f","category":"business","coverImage":"book-20.png","createdAt":"2024-12-10T13:51:55.105Z","description":"Nobody needs to
{"bookId":"672e15be350e5b38b9fd594a","level":"info","message":"Request to get a single book","timestamp":"2024-12-10 19:21:59"}
{"book":{"_id":"672e15be350e5b38b9fd594a","category":"fiction","coverImage":"book-15.png","createdAt":"2024-12-10T13:51:59.988Z","description":"Scarlett O'Hara,
{"bookId":"672e15be350e5b38b9fd594b","level":"info","message":"Request to get a single book","timestamp":"2024-12-10 19:22:04"}
{"book":{"_id":"672e15be350e5b38b9fd594b","category":"fiction","coverImage":"book-16.png","createdAt":"2024-12-10T13:52:04.320Z","description":"Percy Jackson is
{"bookId":"672e15be350e5b38b9fd5947","level":"info","message":"Request to get a single book","timestamp":"2024-12-10 19:22:07"}
{"book":{"_id":"672e15be350e5b38b9fd5947","category":"business","coverImage":"book-12.png","createdAt":"2024-12-10T13:52:07.818Z","description":"Despite the tum
{"bookId":"672e15be350e5b38b9fd5946","level":"info","message":"Request to get a single book","timestamp":"2024-12-10 19:22:13"}
{"book":{"_id":"672e15be350e5b38b9fd5946","category":"fiction","coverImage":"book-11.png","createdAt":"2024-12-10T13:52:13.305Z","description":"The unforgettabl
{"level":"info","message":"Request to get all books","timestamp":"2024-12-10 19:22:23"}
```

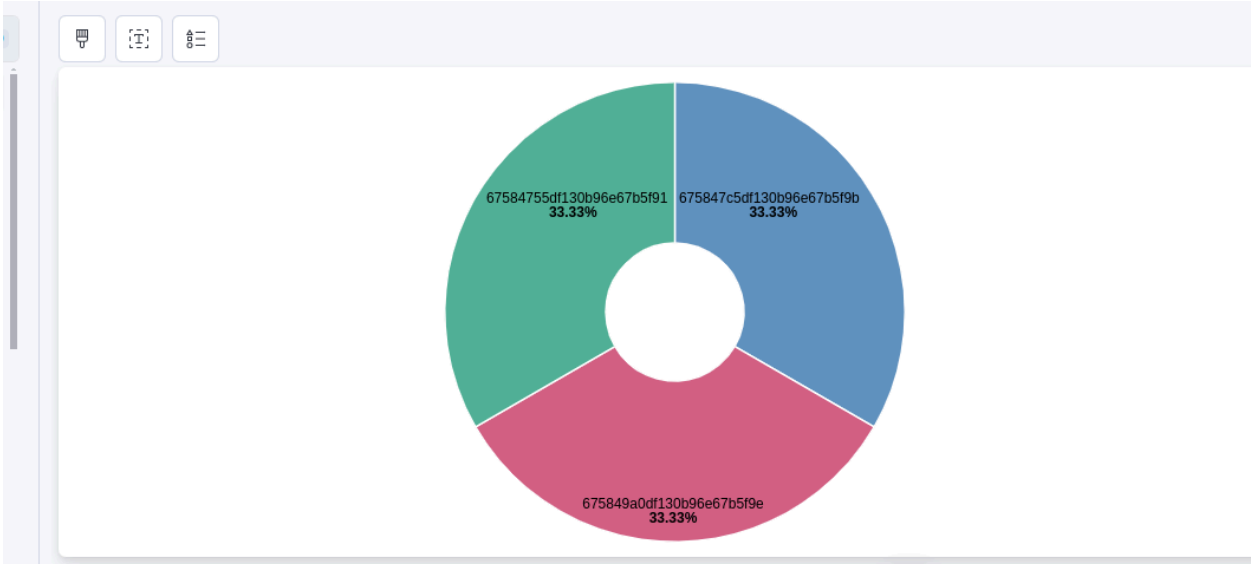

Monitoring with ELK Stack

- We uploaded the log file into elastic search and obtained visualisations of various fields in it.

- These are the users who have used the application in the given time.

- The top 5 books which have been seen by various users during their time of usage of the app.

- Product id’s of different books ordered by various users during the specified time.



In this way, we can visualise various fields by uploading the log files into elastic search and using kibana.

.