Middle East Technical University Department Of Computer Engineering

CENG 495

Cloud Computing

Spring 2024-2025

Second Assignment

Name Surname: Alkım Doğan
Student ID: 2521482

Documentation of Each Step

I have tried each step in MacOs ARM architecture M1 chip.

- **Click here** to see the github repo that I have chosen among the projects in the list given in the homework pdf.
- I actually tried a lot of repos and finally was able to find that one which worked fine.
- Then, I just cloned it to my local by using the command below

  ```
  git clone https://github.com/ewolff/microservice-kubernetes.git
  ```

- Now, I followed the quickstart part from the **skaffold** documentation.
- I generated two **.yaml** files for **Ollama**. One of them is for service where the other is for Ollama itself.
- `mvn clean package` is for building the project I have chosen. It is required for running as far as I see from the Readme file in the repository.

ollama.yaml is below. I increased the memory since I faced same issues at the first trials regarding the resource requirements related to ollama.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ollama
spec:
  selector:
    matchLabels:
      name: ollama
  template:
    metadata:
      labels:
        name: ollama
    spec:
      containers:
      - name: ollama
        image: ollama/ollama:latest
        ports:
        - name: http
          containerPort: 11434
          protocol: TCP
        resources:
          requests:
            memory: "4Gi"
            cpu: "2000m"
          limits:
            memory: "6Gi"
            cpu: "4000m"
```

ollama-service.yaml is below.

```
apiVersion: v1
kind: Service
metadata:
  name: ollama-service
  namespace: default
spec:
  type: LoadBalancer
  selector:
    name: ollama
  ports:
    - port: 11434
      targetPort: 11434
```

I also changed the content of index.html file so that there is a user friendly chatbot in the front-end. The content of the HTML is below. I added bootstrap libraries and very basic HTML and JavaScript.

```html
<!DOCTYPE html>
<html>
<head>
<title>Order Processing</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/css/bootstrap.↩
        min.css" rel="stylesheet" integrity="sha384-SgOJa3DmI69IUzQ2PVdRZhwQ+dy64/↩
        BUtbMJw1MZ8t5HZApcHrRKUc4W0kG879m7" crossorigin="anonymous">

</head>
<body>
<div>
  <h1>Order Processing</h1>
  <div class="container">
    <div class="row">
      <div class="col-md-4">
        <a href="/customer/list.html">Customer</a>
      </div>
      <div class="col-md-4">List / add / remove customers</div>
    </div>
    <div class="row">
      <div class="col-md-4">
        <a href="/catalog/list.html">Catalog</a>
      </div>
      <div class="col-md-4">List / add / remove items</div>
    </div>
    <div class="row">
      <div class="col-md-4">
        <a href="/catalog/searchForm.html">Catalog</a>
      </div>
      <div class="col-md-4">Search Items</div>
    </div>
    <div class="row">
      <div class="col-md-4">
        <a href="/order/">Order</a>
      </div>
      <div class="col-md-4">Create an order</div>
    </div>
    <div class="row">
    </div>
  </div>

</div>
<div class="d-flex justify-content-center" style="padding-top: 3rem; margin-top:3↩
    rem ;">
  <div id="chatbox">
    <h2>Ollama Chatbot</h2>
    <input type="text" id="userInput" placeholder="Type your message...">
    <button id="sendButton">Send</button>
  </div>
</div>

<script>
  const chatbox = document.getElementById('chatbox');
  const userInput = document.getElementById('userInput');
  const sendButton = document.getElementById('sendButton');

  const CHATBOT_URL = 'http://ollama-service:32761/api/generate';

  sendButton.addEventListener('click', sendMessage);
  userInput.addEventListener('keypress', function (e) {
    if (e.key === 'Enter') {
      sendMessage();
    }
  });

  function appendMessage(sender, text) {
    const messageDiv = document.createElement('div');
    messageDiv.classList.add('message', sender);
    messageDiv.innerText = `${sender === 'user' ? 'You' : 'Bot'}: ${text}`;
    chatbox.appendChild(messageDiv);
    chatbox.scrollTop = chatbox.scrollHeight;
  }

  async function sendMessage() {
    const message = userInput.value.trim();
    if (message === '') return;

    appendMessage('user', message);
    userInput.value = '';

    try {
      const response = await fetch(CHATBOT_URL, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          model: 'smollm2',
          prompt: message,
          stream: false
        })
      });

      const data = await response.json();
      if (data.response) {
        appendMessage('bot', data.response);
      } else {
        appendMessage('bot', 'Error: ' + JSON.stringify(data));
      }
    } catch (error) {
      console.error('Error:', error);
      appendMessage('bot', 'Failed to connect to the chatbot.');
    }
  }
</script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.8/dist/umd/popper.↩
    min.js" integrity="sha384-I7E8VVD/ismYTF4hNIPjVp/Zjvgyol6VFvRkX/vR+Vc4jQkC+↩
    hVqc2pM8ODewa9r" crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.5/dist/js/bootstrap.min.↩
```

```
    js" integrity="sha384-VQqxDN0EQCkWoxt/0↩
    vsQvZswzTHUVOImccYmSyhJTp7kGtPed0Qcx8rK9h9YEgx+" crossorigin="anonymous"></↩
    script>

</body>
</html>
```

One thing to mention would be the URL of the ollama in the index.html. `const CHATBOT_URL = 'http://ollama-service:32761/api/generate';` is used in the JavaScript code. This URL is used for the Ollama service in the kubernetes. (**NOTE**: `const CHATBOT_URL = 'http://localhost:11434/api/generate'` can also be used for some cases.)

- Now, I run the following commands.

  1. `skaffold init`. (For initializing the skaffold)
  2. `minikube start -memory=7g -cpus=4` (For starting the minikube with memory constrains to allow ollama)
  3. `skaffold config set -global local-cluster true`
  4. `eval $(minikube -p custom docker-env)` (These are directly from documentation of skaffold)
  5. `skaffold dev`. (This command is used for running to begin using Skaffold for continuous development)

Now, we receive the **skaffold.yaml** file automatically generated with the following content.

```yaml
apiVersion: skaffold/v4beta13
kind: Config
metadata:
  name: microservice-kubernetes-demo
build:
  artifacts:
    - image: docker.io/ewolff/microservice-kubernetes-demo-apache
      context: apache
      docker:
        dockerfile: Dockerfile
    - image: docker.io/ewolff/microservice-kubernetes-demo-catalog
      context: microservice-kubernetes-demo-catalog
      docker:
        dockerfile: Dockerfile
    - image: docker.io/ewolff/microservice-kubernetes-demo-customer
      context: microservice-kubernetes-demo-customer
      docker:
        dockerfile: Dockerfile
    - image: docker.io/ewolff/microservice-kubernetes-demo-order
      context: microservice-kubernetes-demo-order
      docker:
        dockerfile: Dockerfile
manifests:
  rawYaml:
    - microservices.yaml
    - ollama-service.yaml
    - ollama.yaml
```

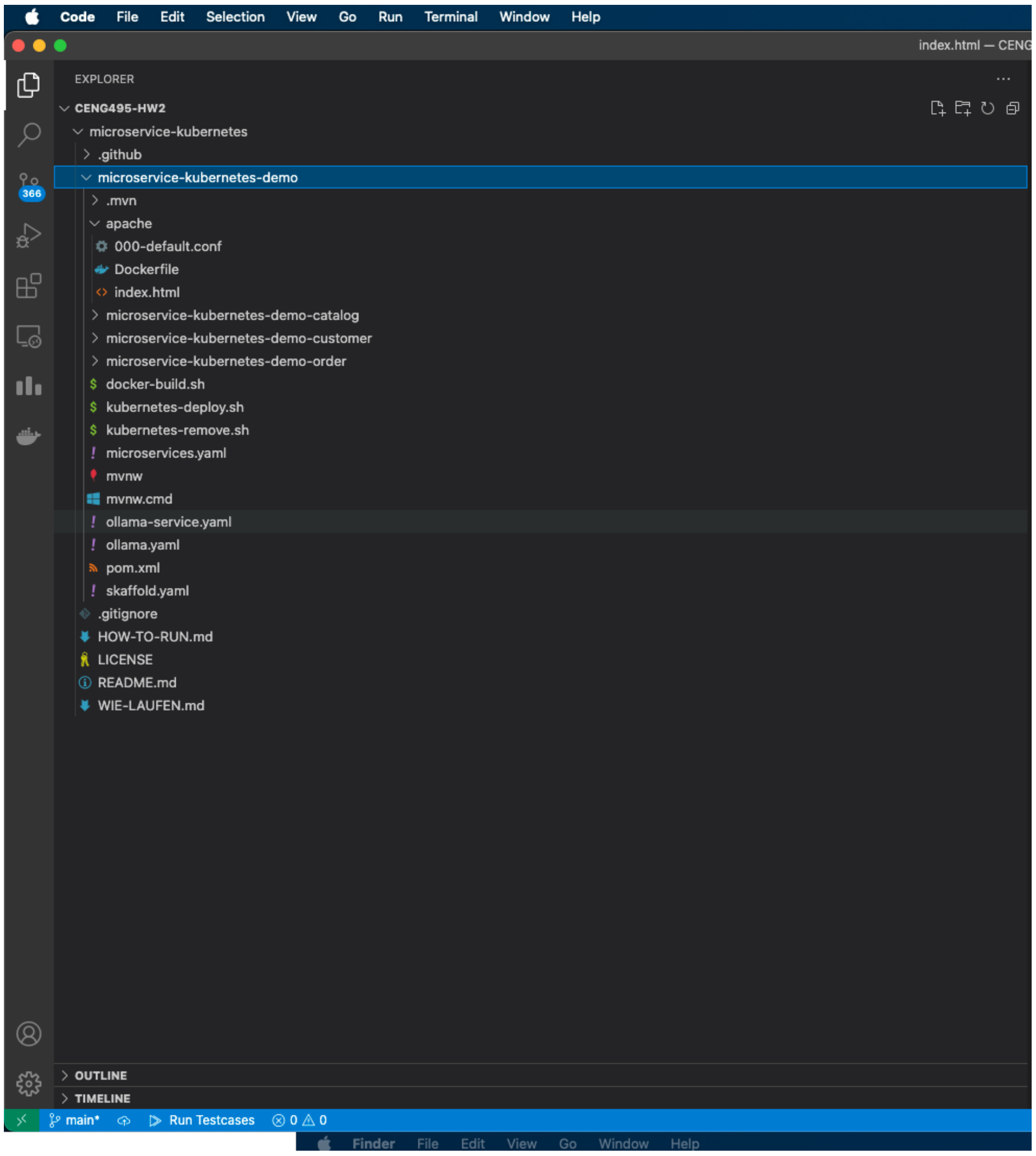
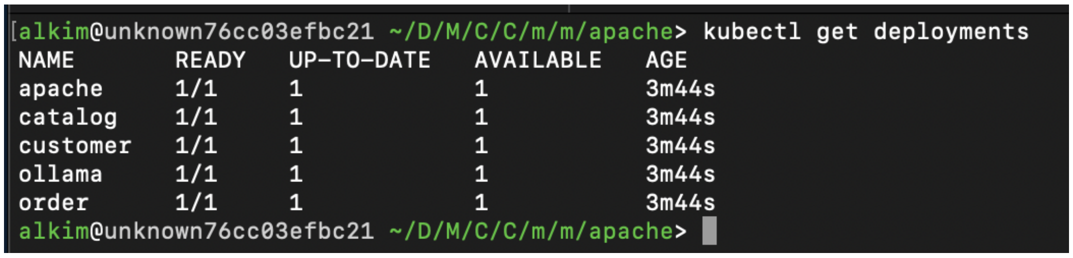


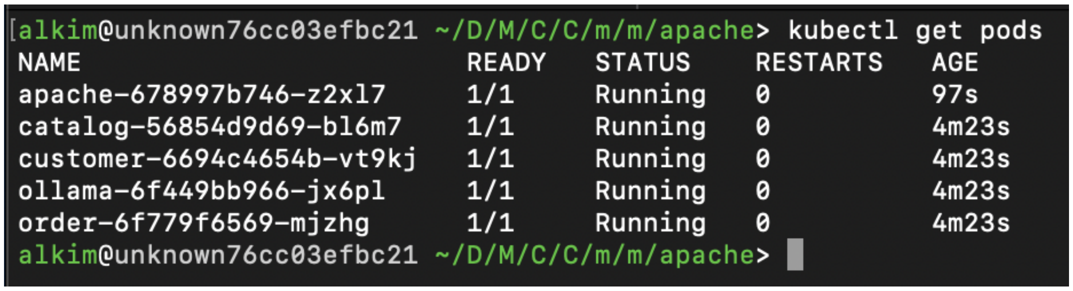
Figure 1: File Paths of The Project. We can see ollama yaml's as well as skaffold.yaml. Index.html can also be seen inside apache server.

After this step, we can also see the some details about the deployment. They actually show that we are on the right path. Below are the screenshots.



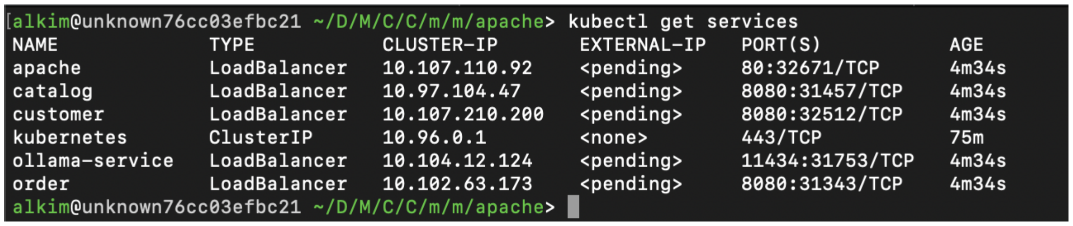
```
[alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache> kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
apache      1/1     1            1           3m44s
catalog     1/1     1            1           3m44s
customer    1/1     1            1           3m44s
ollama      1/1     1            1           3m44s
order       1/1     1            1           3m44s
alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache>
```

Figure 2: Output of kubectl get deployments



```
[alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache> kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
apache-678997b746-z2xl7        1/1     Running   0          97s
catalog-56854d9d69-bl6m7       1/1     Running   0          4m23s
customer-6694c4654b-vt9kj      1/1     Running   0          4m23s
ollama-6f449bb966-jx6pl        1/1     Running   0          4m23s
order-6f779f6569-mjzhg         1/1     Running   0          4m23s
alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache>
```

Figure 3: Output of kubectl get pods



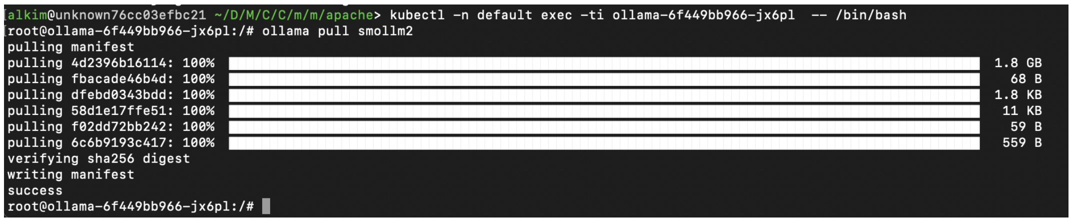
```
[alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache> kubectl get services
NAME             TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)           AGE
apache           LoadBalancer   10.107.110.92    <pending>     80:32671/TCP      4m34s
catalog          LoadBalancer   10.97.104.47     <pending>     8080:31457/TCP    4m34s
customer         LoadBalancer   10.107.210.200   <pending>     8080:32512/TCP    4m34s
kubernetes       ClusterIP      10.96.0.1        <none>        443/TCP           75m
ollama-service   LoadBalancer   10.104.12.124    <pending>     11434:31753/TCP   4m34s
order            LoadBalancer   10.102.63.173    <pending>     8080:31343/TCP    4m34s
alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache>
```

Figure 4: Output of kubectl get services

Then, It is time to pull ollama to the pod.

- `kubectl -n default exec -ti ollama-6f449bb966-jx6pl - /bin/bash` (For SSH to Ollama service pod).
- `ollama pull smollm2` (For pulling the lightweight LLM we chose)

The command above is for ssh into pod. The importent issue is that I have made use of Pod Name to ssh into it.



```
[alkim@unknown76cc03efbc21 ~/D/M/C/C/m/m/apache> kubectl -n default exec -ti ollama-6f449bb966-jx6pl  -- /bin/bash
[root@ollama-6f449bb966-jx6pl:/# ollama pull smollm2
pulling manifest
pulling 4d2396b16114: 100%                                                     1.8 GB
pulling fbacade46b4d: 100%                                                       68 B
pulling dfebd0343bdd: 100%                                                     1.8 KB
pulling 58d1e17ffe51: 100%                                                      11 KB
pulling f02dd72bb242: 100%                                                       59 B
pulling 6c6b9193c417: 100%                                                      559 B
verifying sha256 digest
writing manifest
success
root@ollama-6f449bb966-jx6pl:/#
```

Figure 5: SSH into Pod to download (pull) ollama **smollm2**

Now, I type the command `minikube service apache -url` to see the url. I also visit the url from my browser. The output of the command is **http://127.0.0.1:54077**

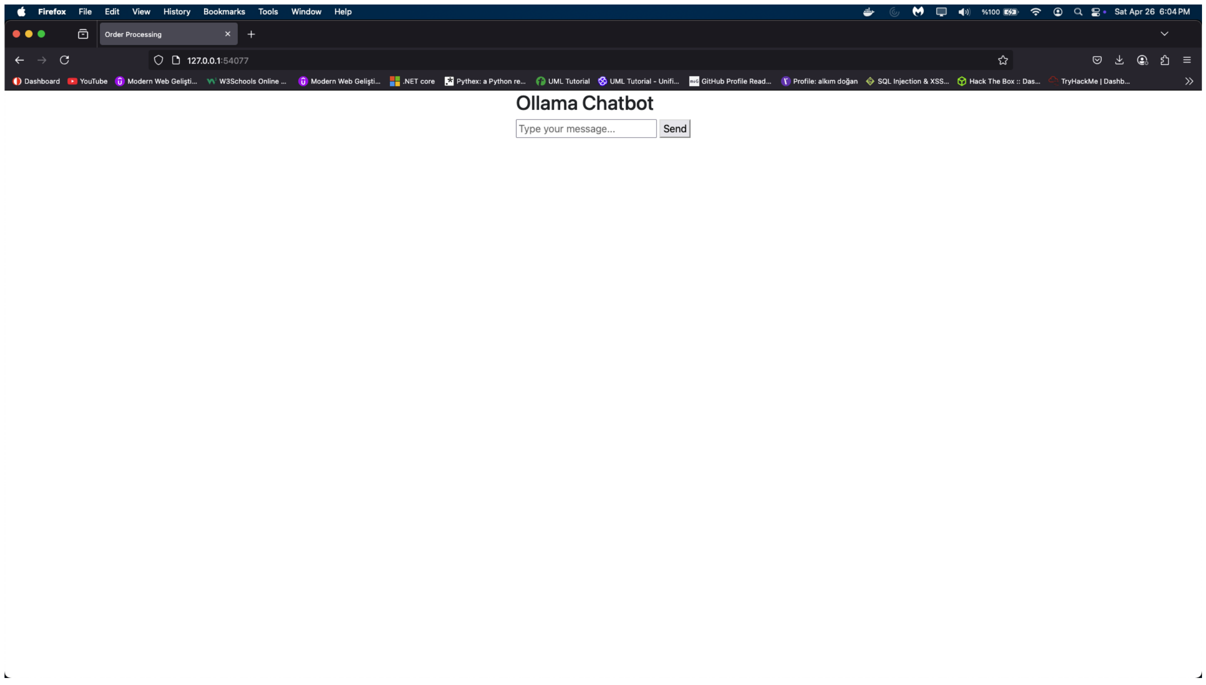
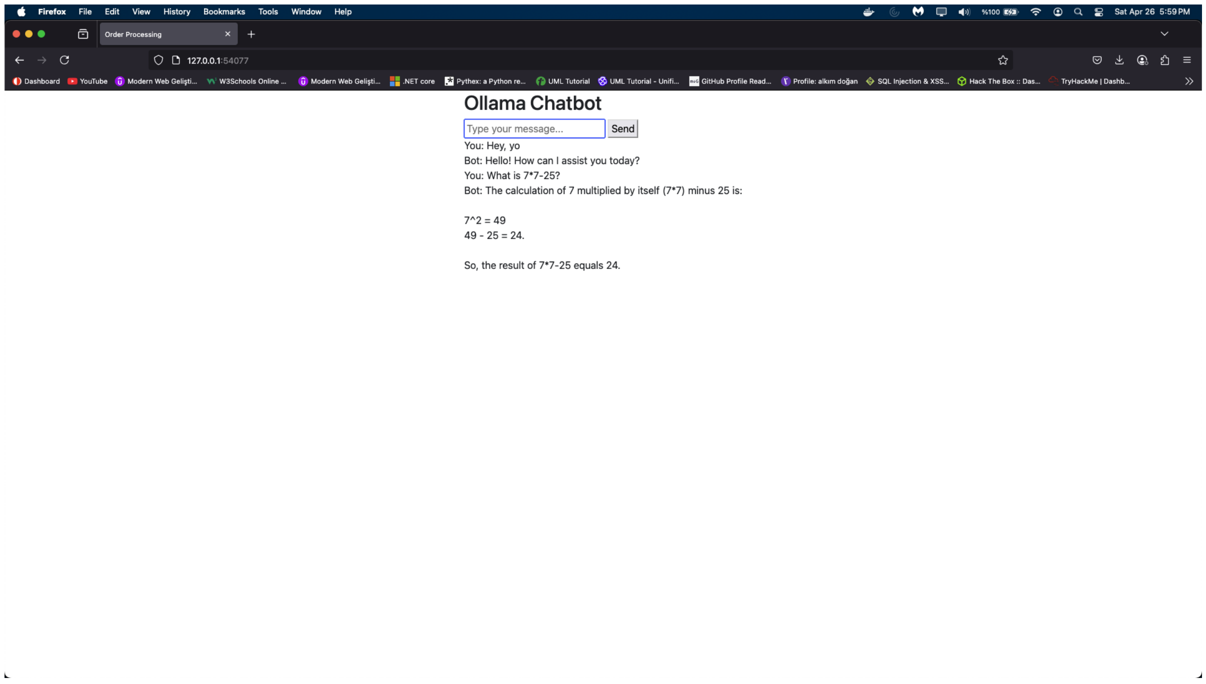


Figure 6: UI of the deployment

After the completion of the downloading (pulling) process of ollama, we can directly talk to chatbot as follows.

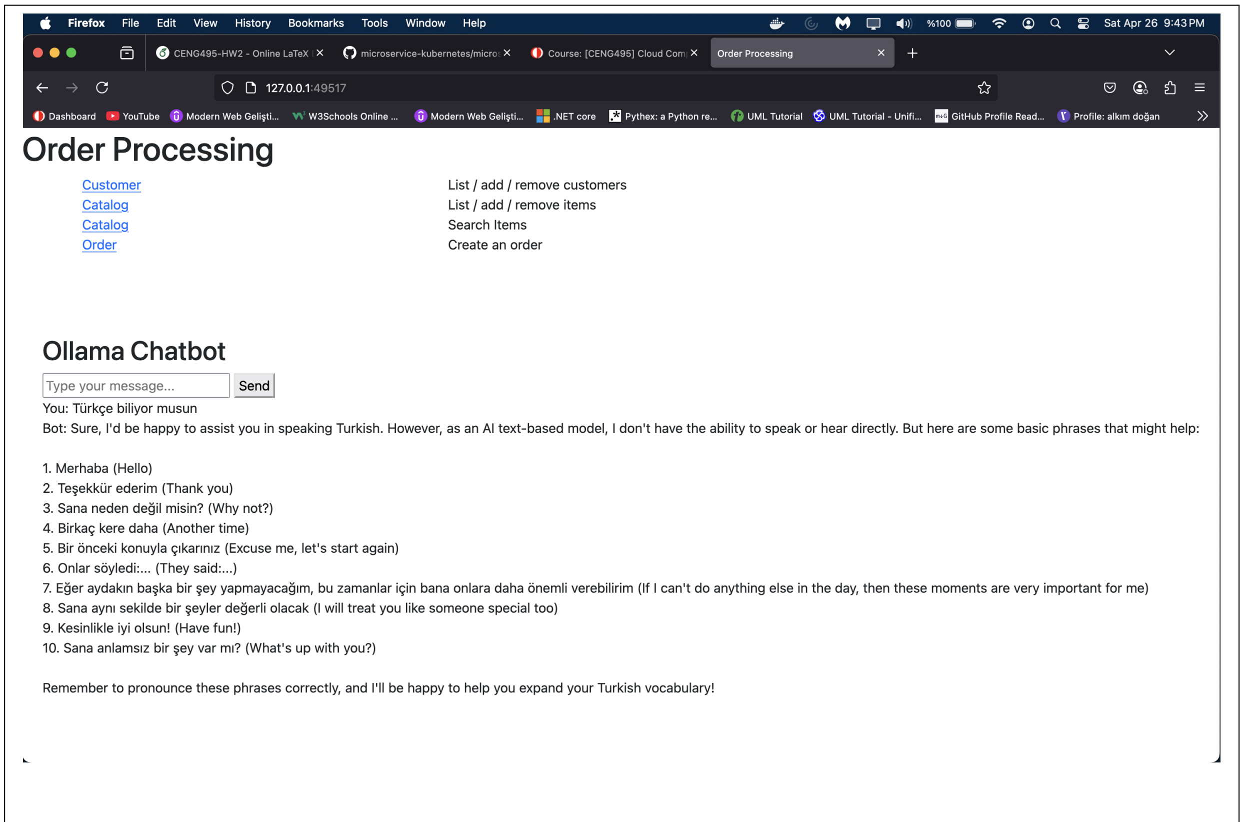


```
You: Hey, yo
Bot: Hello! How can I assist you today?
You: What is 7*7-25?
Bot: The calculation of 7 multiplied by itself (7*7) minus 25 is:

7^2 = 49
49 - 25 = 24.

So, the result of 7*7-25 equals 24.
```

Figure 7: Chatbot is ready!



Order Processing

Customer	List / add / remove customers
Catalog	List / add / remove items
Catalog	Search Items
Order	Create an order

Ollama Chatbot

You: Türkçe biliyor musun

Bot: Sure, I'd be happy to assist you in speaking Turkish. However, as an AI text-based model, I don't have the ability to speak or hear directly. But here are some basic phrases that might help:

1. Merhaba (Hello)
2. Teşekkür ederim (Thank you)
3. Sana neden değil misin? (Why not?)
4. Birkaç kere daha (Another time)
5. Bir önceki konuyla çıkarınız (Excuse me, let's start again)
6. Onlar söyledi:... (They said:...)
7. Eğer aydakın başka bir şey yapmayacağım, bu zamanlar için bana onlara daha önemli verebilirim (If I can't do anything else in the day, then these moments are very important for me)
8. Sana aynı sekilde bir şeyler değerli olacak (I will treat you like someone special too)
9. Kesinlikle iyi olsun! (Have fun!)
10. Sana anlamsız bir şey var mı? (What's up with you?)

Remember to pronounce these phrases correctly, and I'll be happy to help you expand your Turkish vocabulary!

Figure 8: Our application has chatbot!

In the previous image, I had changed the index.html so we can only see chatbot. I re-run the project and changed index.html so that both the application I cloned and ollama chat are available in the front end html file.

Additional Commands

- `kubectl port-forward pod/<pod_name> 11434:11434` (This is used for port forwarding of given pod)
- `kubectl apply -f <file_name>.yaml` (For creating the service given the file name of the yaml)