Cloud Computing - Homework 2

Emre Çam - 2518843

1 May 2025

1 Decisions

- The application I chose: Hipster Shop. It has educational value with different languages and many services and it includes a clear documentation of the deployment.
- The Ollama model I used: `smollm2`.

2 Deployment

2.1 Minikube Setup

I installed and started Minikube using the following commands:

```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force
Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri 'https://github.com/kubernetes
    /minikube/releases/latest/download/minikube-windows-amd64.exe' -UseBasicParsing

minikube start
```

The setup was smooth. I already had Docker installed, so I ran Docker and started Minikube without any problems.

2.2 Skaffold Installation

Installing Skaffold was easier. Chocolatey was already installed on my Windows. All I wrote was:

```
    choco install -y skaffold
```

2.3 Application Deployment

2.3.1 Google Cloud Deployment Mistake

At first, I deployed the project using a Google Cloud Project. To do this, I followed the instructions in the Quickstart (GKE) section of the project's README. Unfortunately, I realized that I needed to deploy it locally. I didn't capture any screenshots of this process, but I can at least share my browser history to show my work (Figure 1).

I had set up the Google Cloud SDK while doing that. Here is the output of the terminal showing the installed version:

```
Emre@DESKTOP-RNSQCS3 MINGW64 ~
$ gcloud version
Google Cloud SDK 520.0.0
beta 2025.04.25
bq 2.1.15
core 2025.04.25
gcloud-crc32c 1.0.0
gke-gcloud-auth-plugin 0.5.10
gsutil 5.34
```

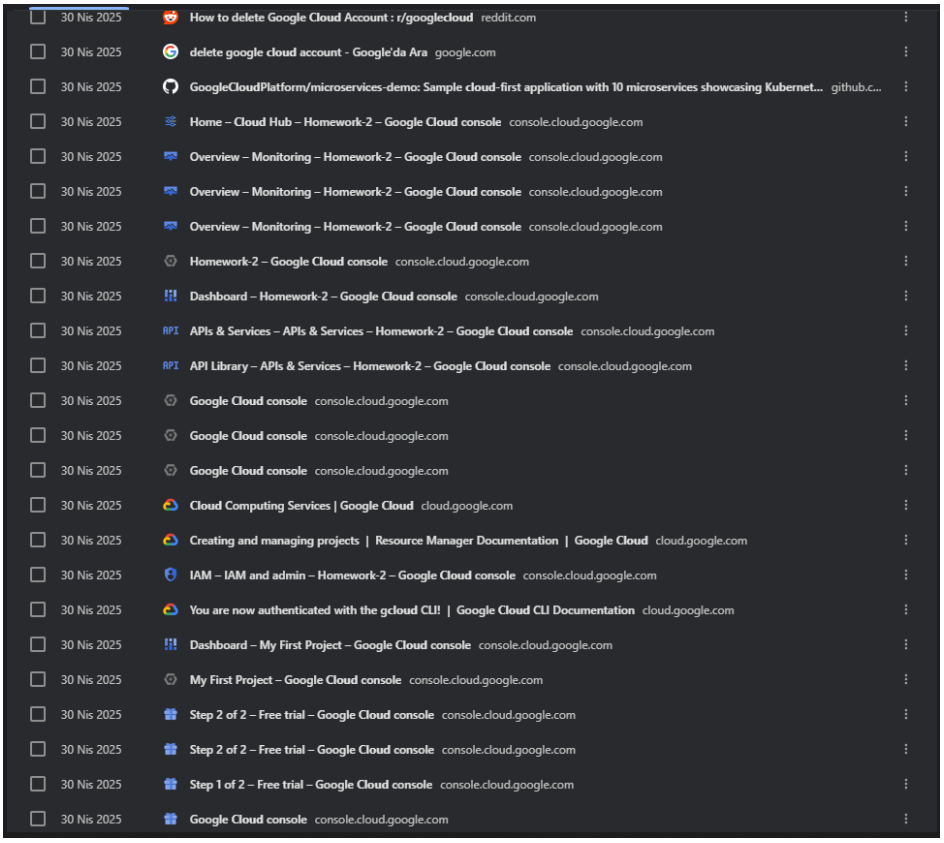

Figure 1: Google Cloud Platform Deployment Mistake

2.3.2 Local Deployment

To deploy the application locally, I followed this documentation. By doing that, the only problem I encountered was this:

```
    - deployment/shippingservice: 0/1 nodes are available: 1 Insufficient cpu.
      preemption: 0/1 nodes are available: 1 No preemption victims were found for the
      incoming pod.
```

I encountered the error even though I started Minikube as suggested in the documentation:

```
   minikube start --cpus=4 --memory 4096 --disk-size 32g
```

I changed the resources of the Minikube and it worked. The resources I used:

```
   minikube start --cpus=12 --memory 8192 --disk-size 32g
```

And that was all I needed to deploy this application locally. I also automatically forward the frontend application to localhost:8080 by adding the following lines to the skaffold.yaml file.

```
# Port forwarding configuration
portForward:
  - resourceType: service
    resourceName: frontend
    port: 80
    localPort: 8080
```

2.4 Ollama Deployment

First, I deployed ollama by using this tutorial :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ollama
  labels:
    app: ollama
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ollama
  template:
    metadata:
      labels:
        app: ollama
    spec:
      containers:
      - name: ollama
        image: ollama/ollama
        ports:
        - containerPort: 11434
        volumeMounts:
        - name: ollama-models
          mountPath: /root/.ollama
      volumes:
      - name: ollama-models
        emptyDir: {}
---
apiVersion: v1
kind: Service
metadata:
  name: ollama
spec:
  selector:
    app: ollama
  ports:
  - protocol: TCP
    port: 11434
    targetPort: 11434
  type: ClusterIP
```

Then I used the following commands to install smollm2 model and to check if it is working:

```
   kubectl exec -it <pod-name> -- ollama pull smollm2
   kubectl port-forward svc/ollama 11434:11434
```

I realized that Ollama's model was lost after restarting Minikube. To fix this, I updated the deployment to use a PersistentVolumeClaim, which ensures the model files are stored persistently. I created the pod and installed the model once. Now it is persistent.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ollama
  labels:
    app: ollama
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ollama
  template:
    metadata:
      labels:
        app: ollama
    spec:
      containers:
      - name: ollama
        image: ollama/ollama
        ports:
        - containerPort: 11434
        resources:
          requests:
            memory: "4Gi"
            cpu: "2"
          limits:
            memory: "8Gi"
            cpu: "4"
        volumeMounts:
        - name: ollama-models
          mountPath: /root/.ollama # To store models
      volumes:
      - name: ollama-models
        persistentVolumeClaim:
          claimName: ollama-persistent-volume-claim
---
apiVersion: v1
kind: PersistentVolumeClaim # To keep models persistent
metadata:
  name: ollama-persistent-volume-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: ollama
spec:
  selector:
    app: ollama
  ports:
  - protocol: TCP
    port: 11434
    targetPort: 11434
  type: ClusterIP
```

2.5 Exposing the Frontend

I exposed the frontend service to make it accessible at `localhost`:

```
# Port forwarding configuration
portForward:
  - resourceType: service
    resourceName: frontend
    port: 80
    localPort: 8080
```

This enabled browser access via `http://localhost:8080`.

2.6 Outputs from Kubectl Commands

Name	Ready	Up-to-date	Available	Age
adservice	1/1	1	1	25h
cartservice	1/1	1	1	25h
checkoutservice	1/1	1	1	25h
currencyservice	1/1	1	1	25h
emailservice	1/1	1	1	25h
frontend	1/1	1	1	25h
loadgenerator	1/1	1	1	25h
ollama	1/1	1	1	25h
paymentservice	1/1	1	1	25h
productcatalogservice	1/1	1	1	25h
recommendationservice	1/1	1	1	25h
redis-cart	1/1	1	1	25h

Table 1: Output of `kubectl get deployments`

Name	Ready	Status	Restarts	Age
adservice-745bd546d8-cq92j	1/1	Running	0	82m
cartservice-6b99df464b-dgkxn	1/1	Running	0	82m
checkoutservice-686f964f4c-fbmb8	1/1	Running	0	82m
currencyservice-6cd8c7c576-492nc	1/1	Running	0	82m
emailservice-5fb8df845f-5t6q2	1/1	Running	0	82m
frontend-548ccfccf8-nh6ch	1/1	Running	0	73m
loadgenerator-d9b97f9d5-zbmrb	1/1	Running	0	82m
ollama-5b758dcd49-bggds	1/1	Running	0	82m
paymentservice-5b549cd6d7-h46k9	1/1	Running	0	82m
productcatalogservice-76497b448d-mgtzr	1/1	Running	0	82m
recommendationservice-7f99c74d86-xlnbb	1/1	Running	0	82m
redis-cart-f88bd6fd8-gg2n2	1/1	Running	0	82m
shippingservice-588c488dbb-6jpd5	1/1	Running	0	82m

Table 2: Output of `kubectl get pods`

Name	Type	Cluster-IP	External-IP	Port(s)	Age
adservice	ClusterIP	10.102.242.26	<none>	9555/TCP	26h
cartservice	ClusterIP	10.103.100.15	<none>	7070/TCP	26h
checkoutservice	ClusterIP	10.110.225.12	<none>	5050/TCP	26h
currencyservice	ClusterIP	10.100.107.202	<none>	7000/TCP	26h
emailservice	ClusterIP	10.105.81.65	<none>	5000/TCP	26h
frontend	ClusterIP	10.101.146.73	<none>	80/TCP	26h
frontend-external	LoadBalancer	10.111.56.205	<pending>	80:32550/TCP	26h
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	27h
ollama	ClusterIP	10.101.229.189	<none>	11434/TCP	26h
paymentservice	ClusterIP	10.97.207.54	<none>	50051/TCP	26h
productcatalogservice	ClusterIP	10.97.243.182	<none>	3550/TCP	26h
recommendationservice	ClusterIP	10.98.157.8	<none>	8080/TCP	26h
redis-cart	ClusterIP	10.106.247.4	<none>	6379/TCP	26h
shippingservice	ClusterIP	10.107.58.215	<none>	50051/TCP	26h

Table 3: Output of `kubectl get services`

Name	Endpoints	Age
adservice	10.244.1.29:9555	26h
cartservice	10.244.1.30:7070	26h
checkoutservice	10.244.1.31:5050	26h
currencyservice	10.244.1.32:7000	26h
emailservice	10.244.1.33:8080	26h
frontend	10.244.1.42:8080	26h
frontend-external	10.244.1.42:8080	26h
kubernetes	192.168.49.2:8443	27h
ollama	10.244.1.28:11434	26h
paymentservice	10.244.1.35:50051	26h
productcatalogservice	10.244.1.36:3550	26h
recommendationservice	10.244.1.37:8080	26h
redis-cart	10.244.1.38:6379	26h
shippingservice	10.244.1.39:50051	26h

Table 4: Output of `kubectl get endpoints`

2.7 LLM Chatbot Integration

I added a smollm2 chatbot to the frontend application. The chatbot interacts with the smollm2 via HTTP requests to the Ollama API.

At first, I wrote the HTML and script directly in the homepage to test it. Once I saw it was working, I cleaned it up by moving the code into separate files: chatbot.html, chatbot.js, and added the styles to styles.css.

2.7.1 Chatbot UI elements

```
{{ define "chatbot" }}
<!-- Chatbot Button -->
<button id="chatbot-toggle" class="chatbot-button">
  <i class="fas fa-robot"></i>
</button>

<!-- Chatbot Popup Window -->
<div id="chatbot-window" class="chatbot-window">
  <div class="chatbot-header">
    <h5>AI Shopping Assistant</h5>
    <button id="chatbot-close" class="chatbot-close-btn">&times;</button>
  </div>
  <div id="chatbot-messages" class="chatbot-messages">
    <div class="message bot-message">Hello! How can I help you with your shopping today
        ?</div>
  </div>
  <div class="chatbot-input-area">
    <input type="text" id="chatbot-input" placeholder="Ask about products..." class="
        chatbot-input">
    <button id="chatbot-send" class="chatbot-send-btn">Send</button>
  </div>
</div>
{{ end }}
```

2.7.2 Chatbot Script

```
document.addEventListener('DOMContentLoaded', function() {
const chatbotToggle = document.getElementById('chatbot-toggle');
const chatbotWindow = document.getElementById('chatbot-window');
const chatbotClose = document.getElementById('chatbot-close');
const chatbotSend = document.getElementById('chatbot-send');
const chatbotInput = document.getElementById('chatbot-input');
const chatbotMessages = document.getElementById('chatbot-messages');

// Toggle chatbot window
chatbotToggle.addEventListener('click', function() {
  chatbotWindow.style.display = chatbotWindow.style.display === 'flex' ? 'none' : 'flex
      ';
});

// Close chatbot window
chatbotClose.addEventListener('click', function() {
  chatbotWindow.style.display = 'none';
});

// Send message
function sendMessage() {
  const message = chatbotInput.value.trim();
  if (message) {
    addMessage(message, 'user');
    chatbotInput.value = '';

    // Call Ollama API (running locally at port 8085)
    fetch('http://localhost:8085/api/generate', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        model: 'smollm2',
        prompt: message,
        stream: false,
      }),
    })
      .then(response => response.json())
      .then(data => {
        if (data.response) {
          addMessage(data.response, 'bot');
        } else {
          addMessage("Sorry, I couldn't process that.", 'bot');
        }
      })
      .catch(error => {
        console.error('Error:', error);
        addMessage("I'm having trouble connecting to the AI service.", 'bot');
      });
  }
}
```

```
  // Send message on button click
  chatbotSend.addEventListener('click', sendMessage);

  // Helper function to add messages to the chat
  function addMessage(text, sender) {
    const messageDiv = document.createElement('div');
    messageDiv.classList.add('message', sender + '-message');
    messageDiv.textContent = text;
    chatbotMessages.appendChild(messageDiv);
    chatbotMessages.scrollTop = chatbotMessages.scrollHeight;
  }
});
```

2.7.3 Added Styles for Chatbot

```
.chatbot-button {
  position: fixed;
  bottom: 20px;
  right: 20px;
  background: #4285f4;
  color: white;
  border: none;
  border-radius: 50%;
  width: 60px;
  height: 60px;
  font-size: 24px;
  cursor: pointer;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
  z-index: 1000;
}

.chatbot-window {
  display: none;
  position: fixed;
  bottom: 90px;
  right: 20px;
  width: 350px;
  height: 500px;
  background: white;
  border-radius: 10px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
  flex-direction: column;
  overflow: hidden;
  z-index: 1000;
}
```

```
.chatbot-header {
  background: #4285f4;
  color: white;
  padding: 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
}

.chatbot-messages {
  flex: 1;
  padding: 15px;
  overflow-y: auto;
}

.chatbot-input-area {
  display: flex;
  padding: 10px;
  border-top: 1px solid #eee;
}

.chatbot-input {
  flex: 1;
  padding: 10px;
  border: 1px solid #ddd;
  border-radius: 4px;
}

.chatbot-send-btn {
  margin-left: 10px;
  padding: 10px 15px;
  background: #4285f4;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.chatbot-close-btn {
  background: none;
  border: none;
  color: white;
  font-size: 20px;
  cursor: pointer;
}

.message {
  margin-bottom: 10px;
  padding: 8px 12px;
  border-radius: 18px;
  max-width: 80%;
}
```

```
.user-message {
  background: #e3f2fd;
  margin-left: auto;
  border-bottom-right-radius: 4px;
}

.bot-message {
  background: #f1f1f1;
  margin-right: auto;
  border-bottom-left-radius: 4px;
}
```

I included this chatbot template in the application's header so it shows up on every page. It can answer questions, but it doesn't know anything about the current page. I considered passing page information with the prompt, but didn't implement that.

Here are some screenshots from the final version:

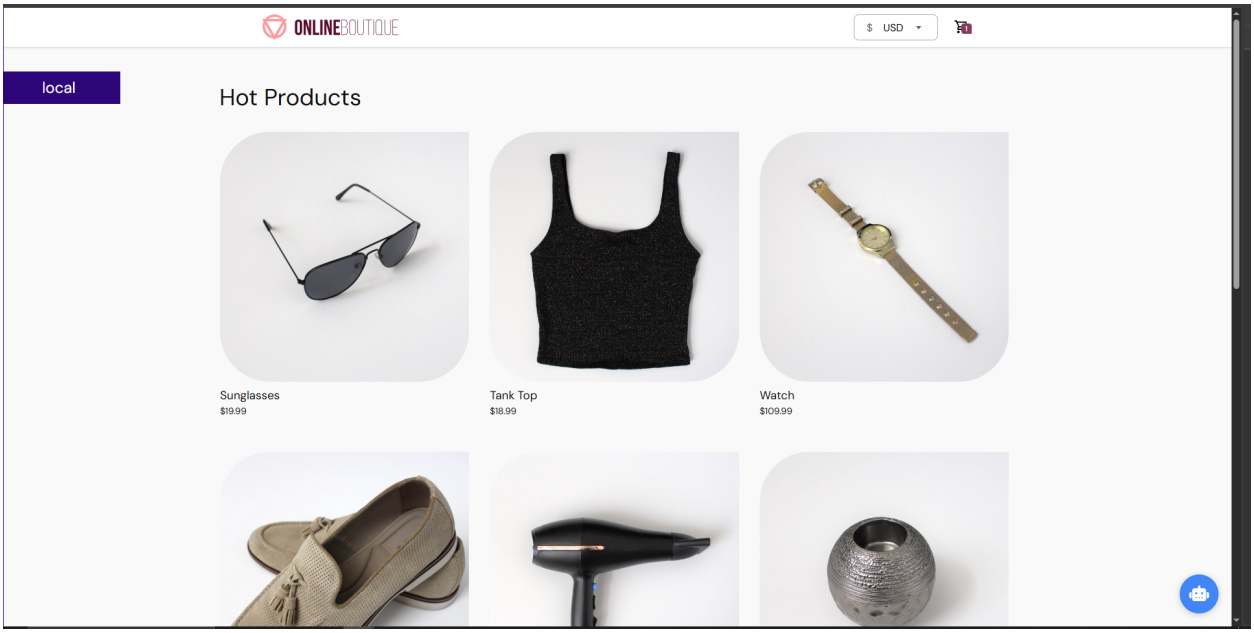

Figure 2: Home Page - Closed

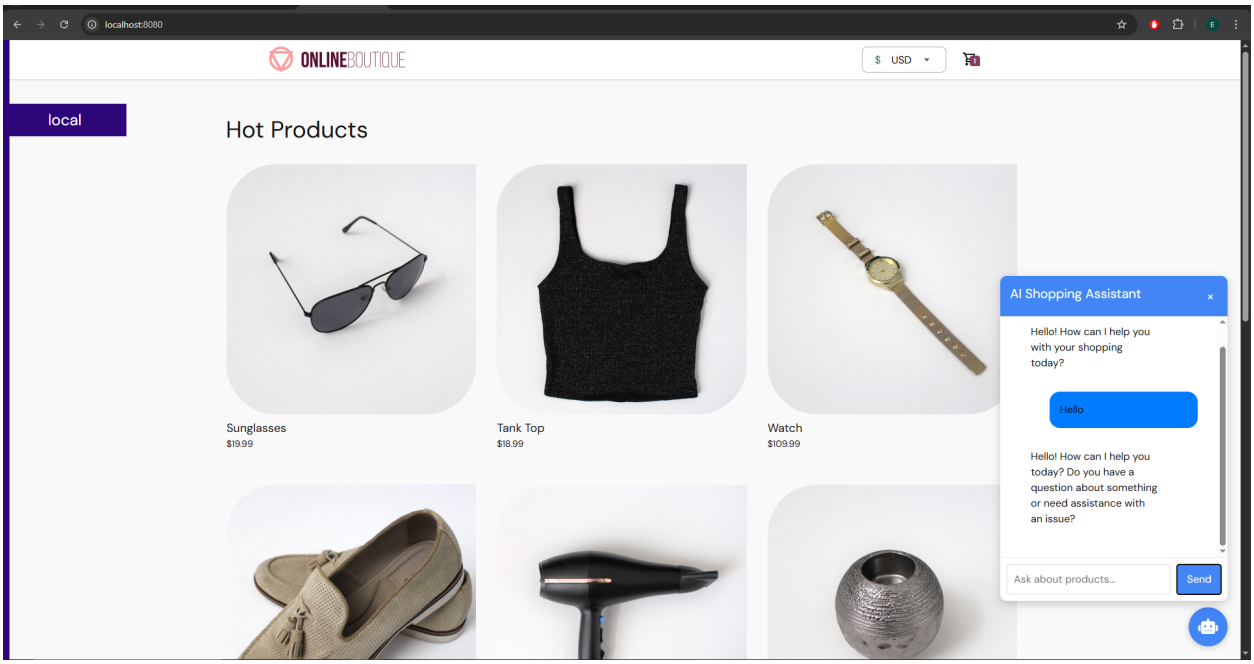

Figure 3: Home Page - Opened

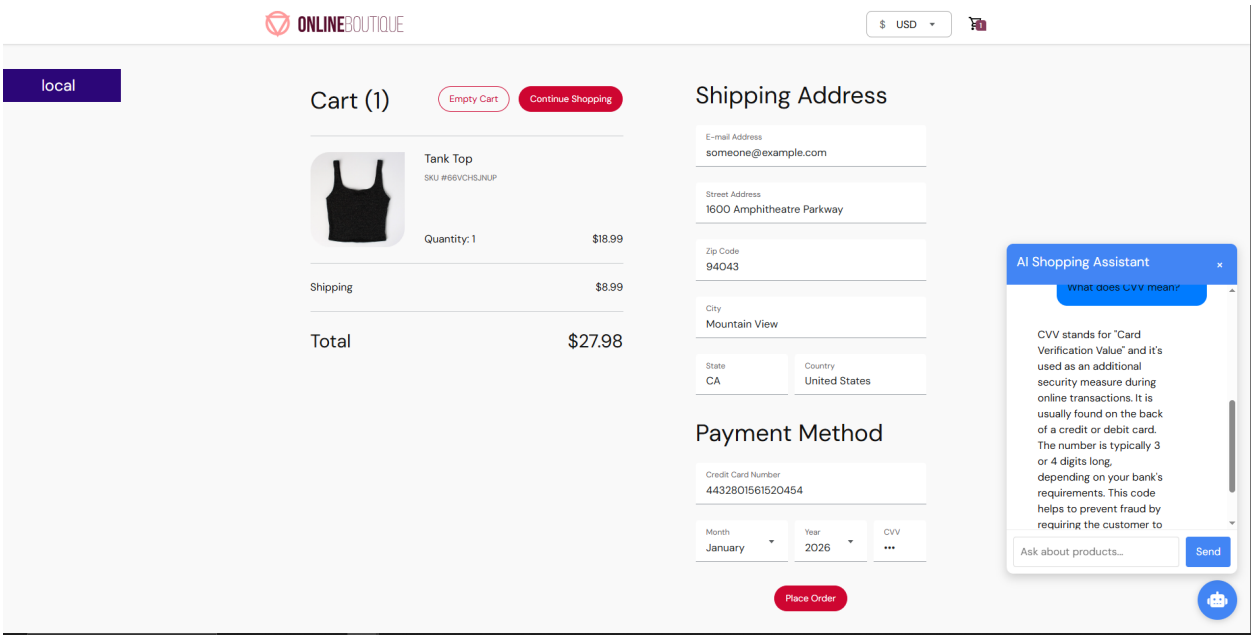

Figure 4: Shopping Cart

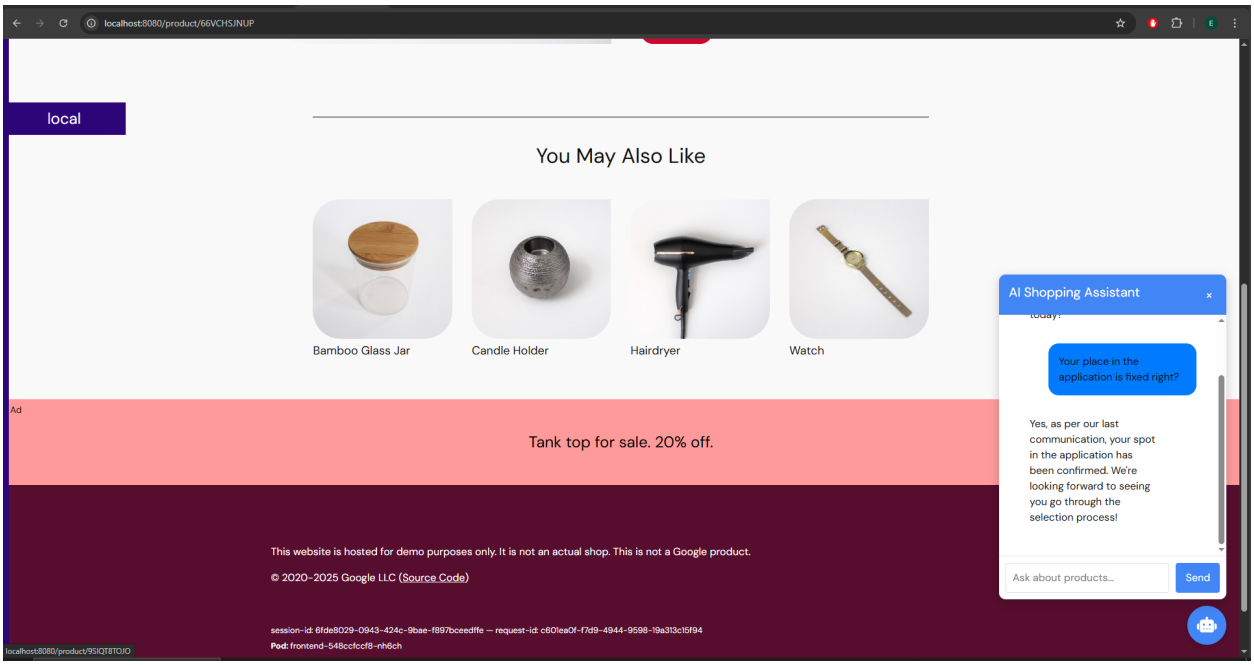

Figure 5: Has Fixed Position at Every Page