<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>credit-card-deals-chat</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.3.9</version>

<relativePath/>

</parent>

<properties>

<java.version>17</java.version>

<spring-cloud.version>2023.0.4</spring-cloud.version>

<aws.sdk.version>2.20.0</aws.sdk.version>

<langchain4j.version>0.10.0</langchain4j.version>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>bom</artifactId>

<version>${aws.sdk.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Boot & Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring AI (for agent orchestration) -->

<dependency>

<groupId>org.springframework.ai</groupId>

<artifactId>spring-ai-core</artifactId>

<version>0.2.0</version>

</dependency>

<!-- LangChain4j in-memory vector store -->

<dependency>

<groupId>com.langchain4j</groupId>

<artifactId>langchain4j</artifactId>

<version>${langchain4j.version}</version>

</dependency>

<!-- AWS SDK v2 (Bedrock, DynamoDb, core) -->

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>bedrock</artifactId>

</dependency>

<dependency>

<groupId>software.amazon.awssdk</groupId>

<artifactId>dynamodb</artifactId>

</dependency>

<!-- Validation & Configuration -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- JSON support -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- Lombok for boilerplate -->

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<!-- Testing -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.example.CreditCardDealsChatApplication</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

package com.example;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class CreditCardDealsChatApplication {

public static void main(String[] args) {

SpringApplication.run(CreditCardDealsChatApplication.class, args);

}

}

package com.example.config;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import software.amazon.awssdk.regions.Region;

import software.amazon.awssdk.services.bedrock.BedrockClient;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;

@Configuration

public class AwsConfig {

@Value("${aws.region}")

private String awsRegion;

/\*\*

\* BedrockClient for Chat and Embeddings calls.

\* Uses the default credentials provider chain.

\*/

@Bean

public BedrockClient bedrockClient() {

return BedrockClient.builder()

.region(Region.of(awsRegion))

.credentialsProvider(DefaultCredentialsProvider.create())

.build();

}

}

package com.example.controller;

import com.example.agent.ChatCoordinator;

import com.example.domain.ChatResponse;

import jakarta.validation.Valid;

import jakarta.validation.constraints.NotBlank;

import org.springframework.http.MediaType;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping(path = "/api/chat", produces = MediaType.APPLICATION\_JSON\_VALUE)

public class ChatController {

private final AgentExecutor executor;

public ChatController(AgentExecutor executor) {

this.executor = executor;

}

@PostMapping

public ChatResponse chat(@RequestBody ChatRequest request) {

// “chatCoordinator” matches the @Component name above

return executor.execute("chatCoordinator", request, ChatResponse.class);

}

public static class ChatRequest {

@NotBlank

private String conversationId;

@NotBlank

private String text;

public ChatRequest() {

}

public ChatRequest(String conversationId, String text) {

this.conversationId = conversationId;

this.text = text;

}

public String getConversationId() {

return conversationId;

}

public void setConversationId(String conversationId) {

this.conversationId = conversationId;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

}

}

package com.example.agent;

import com.example.domain.DealOffer;

import com.example.domain.ProductRecord;

import com.example.client.CardOfferClient;

import com.example.client.DynamicYieldClient;

import com.example.client.GoogleShoppingClient;

import com.example.client.KevelClient;

import com.langchain4j.data.record.Record;

import com.langchain4j.memory.vectorstore.MemoryVectorStore;

import org.springframework.ai.agent.Agent;

import org.springframework.stereotype.Component;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

@Component("commerceAgent")

public class CommerceAgent implements Agent<String, List<DealOffer>> {

private static final int TOP\_K = 5;

private final MemoryVectorStore<ProductRecord> vectorStore;

private final GoogleShoppingClient googleShoppingClient;

private final CardOfferClient cardOfferClient;

private final DynamicYieldClient dynamicYieldClient;

private final KevelClient kevelClient;

public CommerceAgent(MemoryVectorStore<ProductRecord> vectorStore,

GoogleShoppingClient googleShoppingClient,

CardOfferClient cardOfferClient,

DynamicYieldClient dynamicYieldClient,

KevelClient kevelClient) {

this.vectorStore = vectorStore;

this.googleShoppingClient = googleShoppingClient;

this.cardOfferClient = cardOfferClient;

this.dynamicYieldClient = dynamicYieldClient;

this.kevelClient = kevelClient;

}

/\*\*

\* Input: free-form user query

\* Output: a list of enriched DealOffer objects

\*/

@Override

public List<DealOffer> execute(String userQuery) {

// 1) embed & vector-search

float[] embedding = vectorStore.embed(userQuery);

List<Record<ProductRecord>> hits = vectorStore.findNearest(embedding, TOP\_K);

// 2) enrich each result

return hits.stream()

.map(hit -> {

ProductRecord p = hit.record();

String title = googleShoppingClient.fetchTitle(p.getMerchantName());

List<String> benefits = cardOfferClient.fetchCardBenefits(p.getMerchantName());

Map<String, String> personalization = dynamicYieldClient.fetchPersonalization();

Map<String, String> banner = kevelClient.fetchBannerData();

return DealOffer.builder()

.merchantName(p.getMerchantName())

.tier(p.getTier())

.title(title)

.benefits(benefits)

.personalization(personalization)

.banner(banner)

.build();

})

.collect(Collectors.toList());

}

}

package com.example.agent;

import com.example.service.ChatServiceClient;

import org.springframework.ai.agent.Agent;

import org.springframework.stereotype.Component;

@Component("responseAgent")

public class ResponseAgent implements Agent<String, String> {

private final ChatServiceClient chatServiceClient;

public ResponseAgent(ChatServiceClient chatServiceClient) {

this.chatServiceClient = chatServiceClient;

}

/\*\*

\* Input: full prompt (history + offers)

\* Output: LLM’s textual reply

\*/

@Override

public String execute(String prompt) {

return chatServiceClient.chatCompletion(prompt);

}

}

package com.example.agent;

import com.example.controller.ChatController.ChatRequest;

import com.example.domain.ChatResponse;

import com.example.domain.DealOffer;

import com.example.memory.ConversationMemoryService;

import com.example.memory.MemoryRecord;

import org.springframework.ai.agent.Agent;

import org.springframework.stereotype.Component;

import java.util.List;

@Component("chatCoordinator")

public class ChatCoordinatorAgent implements Agent<ChatRequest, ChatResponse> {

private final CommerceAgent commerceAgent;

private final ResponseAgent responseAgent;

private final ConversationMemoryService memoryService;

public ChatCoordinatorAgent(CommerceAgent commerceAgent,

ResponseAgent responseAgent,

ConversationMemoryService memoryService) {

this.commerceAgent = commerceAgent;

this.responseAgent = responseAgent;

this.memoryService = memoryService;

}

/\*\*

\* Input: ChatRequest(conversationId, text)

\* Output: ChatResponse(text, offers)

\*/

@Override

public ChatResponse execute(ChatRequest request) {

String cid = request.getConversationId();

String userMessage = request.getText();

// 1) load & persist user turn

List<MemoryRecord> history = memoryService.getHistory(cid);

memoryService.addMemory(cid, "user", userMessage);

// 2) fetch offers via commerceAgent

List<DealOffer> offers = commerceAgent.execute(userMessage);

// 3) build prompt

String prompt = buildPrompt(history, userMessage, offers);

// 4) get LLM reply & persist

String reply = responseAgent.execute(prompt);

memoryService.addMemory(cid, "assistant", reply);

// 5) return composite response

return ChatResponse.builder()

.text(reply)

.offers(offers)

.build();

}

private String buildPrompt(List<MemoryRecord> history,

String userMessage,

List<DealOffer> offers) {

var sb = new StringBuilder();

for (MemoryRecord turn : history) {

sb.append(turn.getRole())

.append(": ")

.append(turn.getContent())

.append("\n");

}

sb.append("user: ").append(userMessage).append("\n")

.append("Here are some curated deals:\n");

offers.forEach(o -> sb.append("- ")

.append(o.getMerchantName())

.append(" [").append(o.getTier()).append("]: ")

.append(o.getTitle())

.append("\n"));

sb.append("assistant:");

return sb.toString();

}

}

package com.example.memory;

import java.time.Instant;

/\*\*

\* A single turn in a conversation: either user or assistant.

\*/

public class MemoryRecord {

private final Instant timestamp;

private final String role; // "user" or "assistant"

private final String content;

public MemoryRecord(Instant timestamp, String role, String content) {

this.timestamp = timestamp;

this.role = role;

this.content = content;

}

public Instant getTimestamp() {

return timestamp;

}

public String getRole() {

return role;

}

public String getContent() {

return content;

}

}

package com.example.memory;

import org.springframework.stereotype.Service;

import java.time.Instant;

import java.util.\*;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.ConcurrentMap;

/\*\*

\* In-JVM conversation memory.

\* Keeps up to MAX\_TURNS per conversation in a Deque.

\*/

@Service

public class ConversationMemoryService {

// conversationId → chronological deque of MemoryRecord

private final ConcurrentMap<String, Deque<MemoryRecord>> memoryMap = new ConcurrentHashMap<>();

private static final int MAX\_TURNS = 20;

/\*\*

\* Retrieve the conversation history (up to MAX\_TURNS).

\*/

public List<MemoryRecord> getHistory(String conversationId) {

Deque<MemoryRecord> deque = memoryMap.get(conversationId);

if (deque == null) {

return Collections.emptyList();

}

synchronized (deque) {

return new ArrayList<>(deque);

}

}

/\*\*

\* Add a new turn to memory, evicting oldest if over capacity.

\*/

public void addMemory(String conversationId, String role, String content) {

Deque<MemoryRecord> deque = memoryMap.computeIfAbsent(

conversationId, id -> new LinkedList<>()

);

synchronized (deque) {

if (deque.size() >= MAX\_TURNS) {

deque.removeFirst();

}

deque.addLast(new MemoryRecord(Instant.now(), role, content));

}

}

/\*\*

\* Optional: clear all memory for the conversation.

\*/

public void clearConversation(String conversationId) {

memoryMap.remove(conversationId);

}

}

package com.example.client;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import org.springframework.web.reactive.function.client.WebClient;

/\*\*

\* Fetches a user-friendly product title from Google Shopping API.

\*/

@Service

public class GoogleShoppingClient {

private final WebClient webClient;

public GoogleShoppingClient(

@Value("${clients.google-shopping.base-url}") String baseUrl,

WebClient.Builder webClientBuilder) {

this.webClient = webClientBuilder

.baseUrl(baseUrl)

.build();

}

/\*\*

\* Call Google Shopping to retrieve a clean title for a merchant.

\*/

public String fetchTitle(String merchantName) {

return webClient.get()

.uri(uriBuilder ->

uriBuilder.path("/v1/products/search")

.queryParam("q", merchantName)

.build()

)

.retrieve()

.bodyToMono(GoogleShoppingResponse.class)

.map(resp -> resp.getResults().get(0).getTitle())

.block();

}

// DTO for deserialization

private static class GoogleShoppingResponse {

private List<Result> results;

// getters & setters

public List<Result> getResults() { return results; }

public void setResults(List<Result> results) { this.results = results; }

static class Result {

private String title;

// getters & setters

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

}

}

}

package com.example.client;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import org.springframework.web.reactive.function.client.WebClient;

import java.util.List;

/\*\*

\* Retrieves credit-card benefits for a given merchant.

\*/

@Service

public class CardOfferClient {

private final WebClient webClient;

public CardOfferClient(

@Value("${clients.card-offer.base-url}") String baseUrl,

WebClient.Builder builder) {

this.webClient = builder.baseUrl(baseUrl).build();

}

/\*\*

\* Returns a list of benefit descriptions (e.g., "5% cash back").

\*/

public List<String> fetchCardBenefits(String merchantName) {

return webClient.get()

.uri(uriBuilder ->

uriBuilder.path("/offers")

.queryParam("merchant", merchantName)

.build()

)

.retrieve()

.bodyToFlux(String.class)

.collectList()

.block();

}

}

package com.example.client;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import org.springframework.web.reactive.function.client.WebClient;

import java.util.Map;

/\*\*

\* Retrieves personalization attributes from Dynamic Yield.

\*/

@Service

public class DynamicYieldClient {

private final WebClient webClient;

public DynamicYieldClient(

@Value("${clients.dynamic-yield.base-url}") String baseUrl,

WebClient.Builder builder) {

this.webClient = builder.baseUrl(baseUrl).build();

}

/\*\*

\* Returns a map of personalization keys → values

\* (e.g., { "segment": "new\_user", "promo": "spring21" }).

\*/

public Map<String, String> fetchPersonalization() {

return webClient.get()

.uri("/personalization")

.retrieve()

.bodyToMono(new ParameterizedTypeReference<Map<String, String>>() {})

.block();

}

}

package com.example.client;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.core.ParameterizedTypeReference;

import org.springframework.stereotype.Service;

import org.springframework.web.reactive.function.client.WebClient;

import java.util.Map;

/\*\*

\* Retrieves targeted banner/ad data from Kevel.

\*/

@Service

public class KevelClient {

private final WebClient webClient;

public KevelClient(

@Value("${clients.kevel.base-url}") String baseUrl,

WebClient.Builder builder) {

this.webClient = builder.baseUrl(baseUrl).build();

}

/\*\*

\* Returns key → value pairs for banner personalization

\* (e.g., { "bannerId": "123", "creative": "xyz" }).

\*/

public Map<String, String> fetchBannerData() {

return webClient.get()

.uri("/banners/targeted")

.retrieve()

.bodyToMono(new ParameterizedTypeReference<Map<String, String>>() {})

.block();

}

}

package com.example.domain;

import com.langchain4j.data.record.Embeddable;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

/\*\*

\* A product or merchant entry stored in the in-memory vector store.

\*/

@Data

@NoArgsConstructor

@AllArgsConstructor

public class ProductRecord implements Embeddable {

/\*\*

\* Unique identifier for this record.

\*/

private String id;

/\*\*

\* Name of the merchant (e.g. "BestBuy").

\*/

private String merchantName;

/\*\*

\* Reward tier or level (e.g. "Platinum", "Gold").

\*/

private String tier;

/\*\*

\* Text used by the vector store for embeddings.

\*/

private String description;

@Override

public String text() {

return description;

}

}

package com.example.domain;

import lombok.Builder;

import lombok.Data;

import java.util.List;

import java.util.Map;

/\*\*

\* Enriched deal information to show in chat UI.

\*/

@Data

@Builder

public class DealOffer {

/\*\*

\* Merchant name, as stored in ProductRecord.

\*/

private String merchantName;

/\*\*

\* Card tier/category (e.g. "Platinum").

\*/

private String tier;

/\*\*

\* Human-friendly title, fetched from Google Shopping.

\*/

private String title;

/\*\*

\* Cardholder benefits (e.g. "5% cash back").

\*/

private List<String> benefits;

/\*\*

\* Personalization key/value pairs from Dynamic Yield.

\*/

private Map<String, String> personalization;

/\*\*

\* Banner metadata from Kevel (e.g. creative IDs).

\*/

private Map<String, String> banner;

}

package com.example.domain;

import lombok.Builder;

import lombok.Data;

import java.util.List;

/\*\*

\* Payload returned by /api/chat:

\* - text: the assistant’s reply

\* - offers: list of DealOffer objects

\*/

@Data

@Builder

public class ChatResponse {

private String text;

private List<DealOffer> offers;

}

package com.example.service;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Service;

import software.amazon.awssdk.core.SdkBytes;

import software.amazon.awssdk.services.bedrock.BedrockClient;

import software.amazon.awssdk.services.bedrock.model.InvokeModelRequest;

import software.amazon.awssdk.services.bedrock.model.InvokeModelResponse;

import java.nio.charset.StandardCharsets;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Wrapper around AWS Bedrock for embeddings & chat completions.

\*/

@Service

public class ChatServiceClient {

private final BedrockClient bedrockClient;

private final ObjectMapper mapper = new ObjectMapper();

@Value("${models.bedrock.embedding-model-id}")

private String embeddingModelId;

@Value("${models.bedrock.chat-model-id}")

private String chatModelId;

public ChatServiceClient(BedrockClient bedrockClient) {

this.bedrockClient = bedrockClient;

}

/\*\*

\* Returns a float[] embedding for the provided text.

\*/

public float[] getEmbeddings(String text) {

try {

// Build request payload: { "input": ["text"] }

String payload = mapper.writeValueAsString(

mapper.createObjectNode()

.putArray("input").add(text)

);

InvokeModelRequest req = InvokeModelRequest.builder()

.modelId(embeddingModelId)

.body(SdkBytes.fromString(payload, StandardCharsets.UTF\_8))

.build();

InvokeModelResponse resp = bedrockClient.invokeModel(req);

// Parse JSON response: { "embeddings": [[...]] }

JsonNode root = mapper.readTree(resp.body().asUtf8String());

JsonNode vectorNode = root.path("embeddings").get(0);

float[] embedding = new float[vectorNode.size()];

for (int i = 0; i < vectorNode.size(); i++) {

embedding[i] = (float) vectorNode.get(i).asDouble();

}

return embedding;

} catch (Exception e) {

throw new RuntimeException("Failed to get embeddings", e);

}

}

/\*\*

\* Sends a prompt and returns the assistant’s reply.

\*/

public String chatCompletion(String prompt) {

try {

// Build request payload: { "messages": [{ "role": "user", "content": prompt }] }

JsonNode message = mapper.createObjectNode()

.put("role", "user")

.put("content", prompt);

String payload = mapper.writeValueAsString(

mapper.createObjectNode()

.put("modelId", chatModelId)

.set("messages", mapper.createArrayNode().add(message))

);

InvokeModelRequest req = InvokeModelRequest.builder()

.modelId(chatModelId)

.body(SdkBytes.fromString(payload, StandardCharsets.UTF\_8))

.build();

InvokeModelResponse resp = bedrockClient.invokeModel(req);

// Parse JSON response: { "choices": [{ "message": { "content": "..." } }] }

JsonNode root = mapper.readTree(resp.body().asUtf8String());

return root.path("choices")

.get(0)

.path("message")

.path("content")

.asText();

} catch (Exception e) {

throw new RuntimeException("Failed to complete chat", e);

}

}

}

# src/main/resources/application.yml

aws:

region: us-east-1

models:

bedrock:

embedding-model-id: amazon.titan-embedding-001

chat-model-id: amazon.titan-tg1-large

clients:

google-shopping:

base-url: https://api.google.shopping.com

card-offer:

base-url: https://api.cardoffers.com

dynamic-yield:

base-url: https://api.dynamicyield.com

kevel:

base-url: https://api.kevel.com

spring:

main:

allow-bean-definition-overriding: true

 Tune MAX\_TURNS in ConversationMemoryService for memory–latency trade-offs.

* Secure sensitive endpoints and credentials with AWS IAM roles and environment-injected variables.
* Add metrics around Bedrock call latencies and vector-search hit rates.
* Extend domain objects with more fields (e.g. product images, prices) as needed.

{

"name": "credit-card-deals-frontend",

"version": "1.0.0",

"private": true,

"dependencies": {

"react": "^18.2.0",

"react-dom": "^18.2.0",

"react-scripts": "5.0.1",

"axios": "^1.4.0",

"bootstrap": "^5.3.0",

"react-bootstrap": "^2.7.4"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build"

}

}

public/index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0"/>

<title>Credit Card Deals Chat</title>

</head>

<body>

<div id="root"></div>

</body>

</html>

src/index.js

import React from 'react';

import { createRoot } from 'react-dom/client';

import 'bootstrap/dist/css/bootstrap.min.css';

import App from './App';

const root = createRoot(document.getElementById('root'));

root.render(<App />);

src/App.jsx

import React from 'react';

import { Container, Row, Col } from 'react-bootstrap';

import ChatWidget from './components/ChatWidget';

export default function App() {

return (

<Container fluid className="vh-100 d-flex align-items-center justify-content-center bg-light">

<Row className="w-100">

<Col xs={12} md={8} lg={6} xl={4} className="mx-auto">

<ChatWidget />

</Col>

</Row>

</Container>

);

}

src/components/ChatWidget.jsx

import React, { useState, useEffect, useRef } from 'react';

import axios from 'axios';

import {

Card,

ListGroup,

InputGroup,

FormControl,

Button,

Spinner

} from 'react-bootstrap';

import OfferCard from './OfferCard';

export default function ChatWidget() {

const [messages, setMessages] = useState([]);

const [offers, setOffers] = useState([]);

const [input, setInput] = useState('');

const [loading, setLoading] = useState(false);

const bottomRef = useRef(null);

// Auto‐scroll on new messages

useEffect(() => {

bottomRef.current?.scrollIntoView({ behavior: 'smooth' });

}, [messages]);

const send = async () => {

const text = input.trim();

if (!text) return;

// 1) Add user message

setMessages(m => [...m, { text, isUser: true }]);

setInput('');

setLoading(true);

try {

// 2) Call backend

const resp = await axios.post(

'/api/chat',

text,

{ headers: { 'Content-Type': 'text/plain' } }

);

// 3) Update offers

if (resp.data.offers) {

setOffers(resp.data.offers);

}

// 4) Add agent message

setMessages(m => [

...m,

{ text: resp.data.text, isUser: false }

]);

} catch (err) {

console.error(err);

setMessages(m => [

...m,

{ text: 'Sorry, something went wrong.', isUser: false }

]);

} finally {

setLoading(false);

}

};

return (

<Card className="shadow h-100">

<Card.Header className="bg-primary text-white text-center">

Credit Card Deals Chat

</Card.Header>

<Card.Body className="d-flex flex-column p-0">

<ListGroup

variant="flush"

className="flex-grow-1 overflow-auto p-3"

style={{ maxHeight: '55vh' }}

>

{messages.map((m, idx) => (

<ListGroup.Item

key={idx}

className={m.isUser ? 'text-end border-0 py-1' : 'text-start border-0 py-1'}

>

<span

className={

'py-2 px-3 rounded-3 d-inline-block ' +

(m.isUser ? 'bg-primary text-white' : 'bg-light text-dark')

}

>

{m.text}

</span>

</ListGroup.Item>

))}

<div ref={bottomRef} />

</ListGroup>

<div className="p-3 border-top bg-white">

{offers.map((o, idx) => <OfferCard key={idx} offer={o} />)}

</div>

</Card.Body>

<Card.Footer className="p-2 bg-light">

<InputGroup>

<FormControl

placeholder="Ask for deals…"

value={input}

disabled={loading}

onChange={e => setInput(e.target.value)}

onKeyDown={e => e.key === 'Enter' && send()}

/>

<Button

variant="primary"

onClick={send}

disabled={loading || !input.trim()}

>

{loading ? <Spinner size="sm" animation="border" /> : 'Send'}

</Button>

</InputGroup>

</Card.Footer>

</Card>

);

}

src/components/OfferCard.js

import React from 'react';

import { Card } from 'react-bootstrap';

export default function OfferCard({ offer }) {

const {

merchantName,

tier,

productDesc,

cardOffer,

banner

} = offer;

return (

<Card className="mb-2">

<Card.Body>

<Card.Title>{merchantName}</Card.Title>

<Card.Subtitle className="mb-2 text-muted">{tier}</Card.Subtitle>

<Card.Text>{productDesc}</Card.Text>

<Card.Text><strong>{cardOffer}</strong></Card.Text>

</Card.Body>

{banner?.imageUrl && (

<Card.Img

variant="bottom"

src={banner.imageUrl}

style={{ cursor: 'pointer' }}

onClick={() => window.open(banner.link, '\_blank')}

/>

)}

</Card>

);

}

7. Optional: src/components/MessageList.jsx & Message.jsx & MessageInput.jsx

If you prefer to split messages into separate components, here they are:

MessageList.jsx

import React from 'react';

import { ListGroup } from 'react-bootstrap';

import Message from './Message';

export default function MessageList({ messages }) {

return (

<ListGroup variant="flush">

{messages.map((m, i) => (

<Message key={i} message={m} />

))}

</ListGroup>

);

}

Message.jsx

import React from 'react';

export default function Message({ message }) {

const align = message.isUser ? 'text-end' : 'text-start';

const bg = message.isUser ? 'bg-primary text-white' : 'bg-light text-dark';

return (

<div className={`${align} my-1`}>

<span className={`py-2 px-3 rounded-3 d-inline-block ${bg}`}>

{message.text}

</span>

</div>

);

}

MessageInput.jsx

import React, { useState } from 'react';

import { InputGroup, FormControl, Button } from 'react-bootstrap';

export default function MessageInput({ onSend, disabled }) {

const [text, setText] = useState('');

const handleSend = () => {

const t = text.trim();

if (!t) return;

onSend(t);

setText('');

};

return (

<InputGroup className="mt-3">

<FormControl

placeholder="Type your message…"

value={text}

disabled={disabled}

onChange={e => setText(e.target.value)}

onKeyDown={e => e.key === 'Enter' && handleSend()}

/>

<Button onClick={handleSend} disabled={disabled || !text.trim()}>

Send

</Button>

</InputGroup>

);

}

How to Run

1. cd frontend
2. npm install
3. npm start

Point your browser at http://localhost:3000 (or proxy to your Spring Boot backend at http://localhost:8080). You now have a rich, responsive chat UI showing messages and personalized offer cards

Directory Layout

/my-app

/backend ← Spring Boot project (Maven or Gradle)

pom.xml

src/main/java/…

src/main/resources/

application.yml

/static ← React’s production build goes here

/frontend ← React app (created with create-react-app)

package.json

public/

src/

Manual Build & Copy (Simplest)

 Build React

cd frontend npm ci npm run build

1. Copy into Spring Boot

rm -rf ../backend/src/main/resources/static/\* cp -r build/\* ../backend/src/main/resources/static/

1. Package & Run

cd ../backend mvn clean package java -jar target/backend-0.0.1-SNAPSHOT.jar

Spring Boot will now serve:

* / → index.html
* /static/js/... → your JS bundles
* /api/\*\* → your REST controllers

<build>

<plugins>

<!-- 1. Frontend Plugin -->

<plugin>

<groupId>com.github.eirslett</groupId>

<artifactId>frontend-maven-plugin</artifactId>

<version>1.12.1</version>

<executions>

<!-- Install Node & npm -->

<execution>

<id>install node and npm</id>

<goals><goal>install-node-and-npm</goal></goals>

<configuration>

<nodeVersion>v18.16.0</nodeVersion>

<npmVersion>9.8.0</npmVersion>

</configuration>

</execution>

<!-- npm install -->

<execution>

<id>npm install</id>

<goals><goal>npm</goal></goals>

<configuration>

<arguments>ci</arguments>

<workingDirectory>../frontend</workingDirectory>

</configuration>

</execution>

<!-- npm build -->

<execution>

<id>npm build</id>

<goals><goal>npm</goal></goals>

<configuration>

<arguments>run build</arguments>

<workingDirectory>../frontend</workingDirectory>

</configuration>

</execution>

</executions>

</plugin>

<!-- 2. Copy React build into Spring static -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.3.1</version>

<executions>

<execution>

<id>copy-react-build</id>

<phase>generate-resources</phase>

<goals><goal>copy-resources</goal></goals>

<configuration>

<outputDirectory>${project.build.outputDirectory}/static</outputDirectory>

<resources>

<resource>

<directory>${basedir}/../frontend/build</directory>

<filtering>false</filtering>

</resource>

</resources>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

cd backend

mvn clean package

java -jar target/backend-0.0.1-SNAPSHOT.jar

// src/main/java/com/example/WebConfig.java

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.\*;

@Configuration

public class WebConfig implements WebMvcConfigurer {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/\*\*/\*")

.addResourceLocations("classpath:/static/")

.resourceChain(true);

}

@Override

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/{spring:\\w+}")

.setViewName("forward:/index.html");

registry.addViewController("/\*\*/{spring:\\w+}")

.setViewName("forward:/index.html");

registry.addViewController("/{spring:\\w+}/\*\*{spring:?!(\\.js|\\.css)$}")

.setViewName("forward:/index.html");

}

}

credit-card-deals-chat/

│

├─ pom.xml

│

├─ src/

│ └─ main/

│ ├─ java/com/example/

│ │ ├─ CreditCardDealsChatApplication.java

│ │ │

│ │ ├─ config/

│ │ │ └─ AwsConfig.java

│ │ │

│ │ ├─ controller/

│ │ │ └─ ChatController.java

│ │ │

│ │ ├─ agent/

│ │ │ ├─ CommerceAgent.java

│ │ │ ├─ ResponseAgent.java

│ │ │ └─ ChatCoordinatorAgent.java

│ │ │

│ │ ├─ memory/

│ │ │ ├─ MemoryRecord.java

│ │ │ └─ ConversationMemoryService.java

│ │ │

│ │ ├─ client/

│ │ │ ├─ GoogleShoppingClient.java

│ │ │ ├─ CardOfferClient.java

│ │ │ ├─ DynamicYieldClient.java

│ │ │ └─ KevelClient.java

│ │ │

│ │ ├─ domain/

│ │ │ ├─ ProductRecord.java

│ │ │ ├─ DealOffer.java

│ │ │ └─ ChatResponse.java

│ │ │

│ │ └─ service/

│ │ └─ ChatServiceClient.java

│ │

│ └─ resources/

│ └─ application.yml

│

└─ frontend/

├─ package.json

│

├─ public/

│ └─ index.html

│

└─ src/

├─ index.js

├─ App.jsx

└─ components/

├─ ChatWidget.jsx

├─ OfferCard.jsx

├─ MessageList.jsx

├─ Message.jsx

└─ MessageInput.jsx

Credit-Card Deals Chat (In-Memory Vector Store + AWS Bedrock)

A pre-login chat widget that recommends partner-merchant offers (premium/non-premium), enriched with card-offer details, personalization, and ad banners—powered by:

* AWS Bedrock for embeddings & chat
* In-memory vector store (no external DB)
* Google Shopping, Dynamic Yield & KEVEL for domain data
* Internal Card-Offer API
* Spring AI Agents orchestrated in Spring Boot
* Responsive React-Bootstrap front end

** ChatController** receives the user prompt.

1. **ChatCoordinator** logs the turn, calls **CommerceAgent**, then **ResponseAgent**.
2. **CommerceAgent**
   * Fetches product candidates from Google Shopping.
   * Embeds products & query via Bedrock embed model.
   * Scores via cosine similarity in an in-memory vector store.
   * Calls CardOffer, Dynamic Yield & KEVEL APIs for enrichment.
3. **ResponseAgent** builds a RAG prompt with top-K offers and calls Bedrock chat.
4. **React UI** renders messages and responsive offer cards

// src/main/java/com/example/app/dto/IntentDto.java

package com.example.app.dto;

import java.util.Map;

public class IntentDto {

private UserIntent intent;

private Map<String, String> entities;

public IntentDto() {}

public IntentDto(UserIntent intent, Map<String, String> entities) {

this.intent = intent;

this.entities = entities;

}

public UserIntent getIntent() {

return intent;

}

public void setIntent(UserIntent intent) {

this.intent = intent;

}

public Map<String, String> getEntities() {

return entities;

}

public void setEntities(Map<String, String> entities) {

this.entities = entities;

}

}

// src/main/java/com/example/app/dto/UserIntent.java

package com.example.app.dto;

public enum UserIntent {

GET\_TRAVEL\_DEALS,

GET\_CASHBACK\_DEALS,

GET\_CREDIT\_SCORE\_INFO,

UNKNOWN

}

// src/main/java/com/example/app/agent/IntentRecognitionAgent.java

package com.example.app.agent;

import com.example.app.dto.IntentDto;

import com.example.app.dto.UserIntent;

import com.example.app.external.LLMClient;

import com.fasterxml.jackson.core.type.TypeReference;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Service;

import java.util.Collections;

import java.util.Map;

/\*\*

\* Spring AI “agent” that calls an LLM to classify user input into an intent

\* and extract any entities. Returns a strongly-typed IntentDto.

\*/

@Service

public class IntentRecognitionAgent {

private static final Logger logger = LoggerFactory.getLogger(IntentRecognitionAgent.class);

private final LLMClient llmClient;

private final ObjectMapper objectMapper;

public IntentRecognitionAgent(LLMClient llmClient,

ObjectMapper objectMapper) {

this.llmClient = llmClient;

this.objectMapper = objectMapper;

}

/\*\*

\* Recognize the user’s intent by prompting the LLM for a JSON response.

\*

\* @param userMessage the raw text from the chat widget

\* @return an IntentDto containing the intent enum and any entities

\*/

public IntentDto recognize(String userMessage) {

String prompt = buildPrompt(userMessage);

try {

// 1) Call the LLM

String jsonResponse = llmClient.chatCompletion(prompt);

// 2) Parse the JSON into a Map

Map<String, Object> result = objectMapper.readValue(

jsonResponse,

new TypeReference<Map<String, Object>>() {}

);

// 3) Extract intent field

String intentStr = (String) result.getOrDefault("intent", "UNKNOWN");

UserIntent intent;

try {

intent = UserIntent.valueOf(intentStr);

} catch (IllegalArgumentException ex) {

intent = UserIntent.UNKNOWN;

}

// 4) Extract entities field (if any)

@SuppressWarnings("unchecked")

Map<String, String> entities = (Map<String, String>) result.get("entities");

if (entities == null) {

entities = Collections.emptyMap();

}

return new IntentDto(intent, entities);

} catch (Exception ex) {

logger.error("Failed to recognize intent for message [{}]", userMessage, ex);

// Fallback to UNKNOWN intent

return new IntentDto(UserIntent.UNKNOWN, Collections.emptyMap());

}

}

/\*\*

\* Build a prompt that instructs the LLM to return JSON with

\* keys "intent" and "entities".

\*/

private String buildPrompt(String userMessage) {

return """

You are an intent classification agent for a credit-card deals chat.

Read the user’s message, classify it into one of:

GET\_TRAVEL\_DEALS, GET\_CASHBACK\_DEALS, GET\_CREDIT\_SCORE\_INFO, UNKNOWN

Also extract any entities (e.g., "travel", "cashback", "student", etc.).

Respond in JSON ONLY, with fields:

{

"intent": "<INTENT\_ENUM>",

"entities": { "key1": "value1", ... }

}

User Message:

\"%s\"

"""

.formatted(userMessage);

}

}

==============