King Fahd University of Petroleum and Minerals

COE 452

Section 01

Bank Queue Management Project

Intro. To Cloud Engineering – Term 241

To: Dr. Abdulaziz Al-Tabbakh

Name	ID
Basil Alashqar	202045700
Abdulrahman Sharqawi	202182610
Mohab Hussein	202182810

Date: 12/09/2024

Introduction

The Riyadh Bank Queue Management System is a modern solution aimed at optimizing customer service operations in one of the busiest financial institutions in Saudi Arabia. This system addresses the challenges of long waiting times, inefficient ticket allocation, and inadequate tracking of customer service statuses. By leveraging serverless cloud architecture and integrating modern services such as Amazon DynamoDB, AWS Lambda, S3 storage, and API Gateway, the system ensures seamless ticket generation, service allocation, and counter authentication processes. The project's objective is to deliver a scalable, efficient, and user-friendly system that enhances customer satisfaction and operational productivity.

Problem Statement

The current queue management system at Riyadh Bank faces challenges related to efficiency, scalability, and redundancy. Long waiting times, lack of real-time tracking, and system failures during high-demand periods affect customer satisfaction and operational effectiveness. Additionally, the system's lack of redundancy increases the risk of downtime, impacting the bank's ability to provide uninterrupted services.

Goal

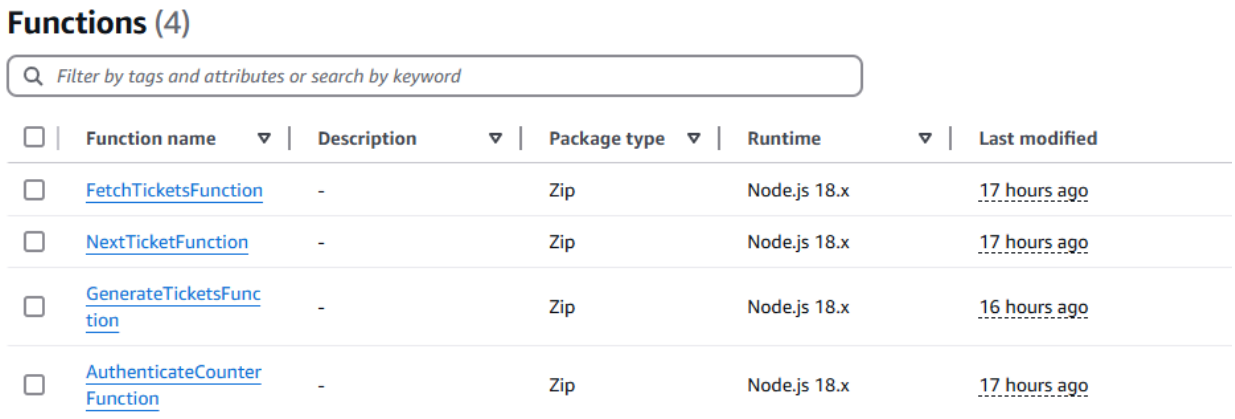
The primary goal of this project is to optimize Riyadh Bank's queue management system by developing a scalable, redundant, and efficient cloud-based solution. The system aims to streamline ticket generation, counter service allocation, and real-time monitoring, ensuring high availability and fault tolerance. By leveraging AWS cloud services, the project ensures seamless performance during peak hours, reduces latency, and provides a reliable backup for critical operations, ultimately enhancing the customer experience and operational reliability.

Services and API implementation

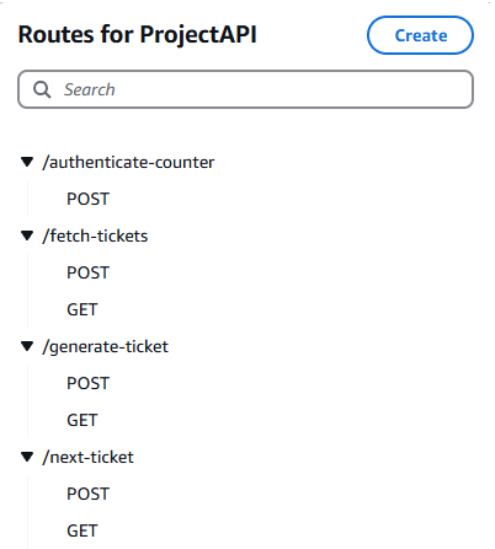
Service Name	Description	HTTP method	Route
GenerateTickets **(Public)**	**Generate a new ticket for a specified service type.**	**Post**	**/generate-ticket**
GenerateTickets	**Reads info from all generated tickets.**	**Get**	**/generate-ticket**
NextTicket **(Private)**	**Assigns the tickets in the queue for the specific counters**	**Post**	**/next-ticket**

NextTicket	**Reads the next ticket in the queue**	**Get**	**/next-ticket**
FetchTickets **Private**	**Retrieve all tickets currently in the system (Dynamo DB)** **Response with (Ticket ID, service type, status)**	**Get**	**/fetch-tickets**
FetchTickets	**Filters tickets with a json file with {“serviceType”: “Tellers”,** **“status”: “waiting”}**	**Post**	**/fetch-tickets**
AuthenticateCounter **Private**	**Authenticate who is operating the counter**	**Post**	**/authenticate-counter**

Lambda Functions:



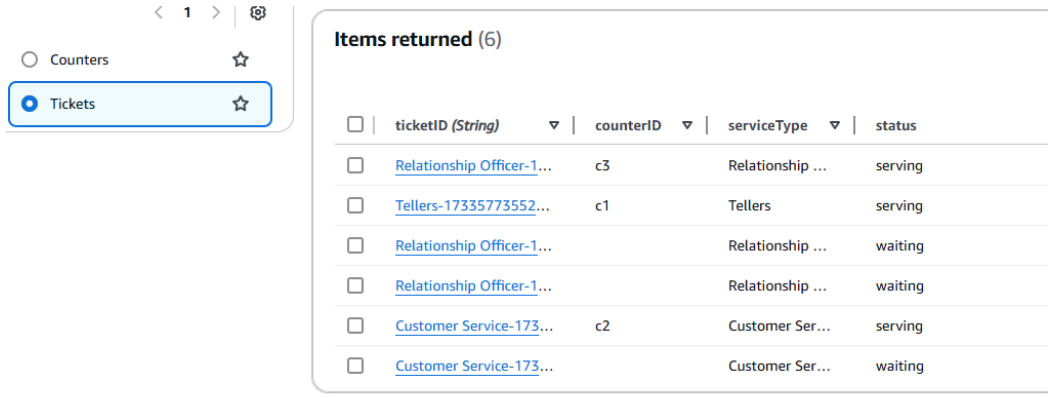
API gateway routes:



Dynamo DB and S3:

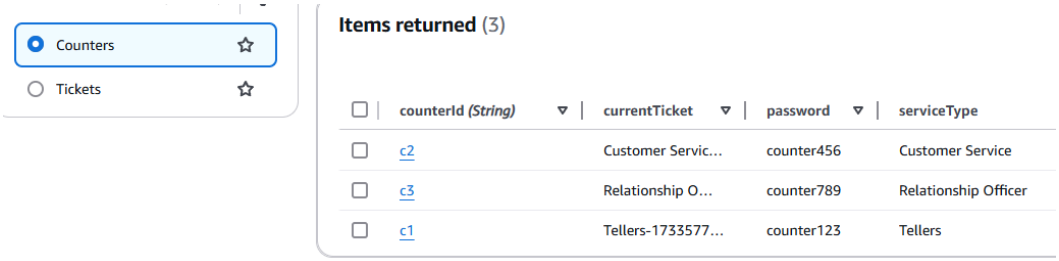
1. DynamoDB

- **Purpose**:
  DynamoDB is used as the primary database for storing and managing ticket-related information efficiently. It ensures high availability, scalability, and low latency for real-time ticket operations.
- **Tables Used**:
  - **Tickets Table**:
    - **Schema**:
      - **Partition Key**: `ticketID` (String) - A unique identifier for each ticket, formatted as `<serviceType>-<timestamp>-<randomNumber>`.
      - **Attributes**:
        - `serviceType` (String): The type of service requested (e.g., "Tellers", "Customer Service").
        - `status` (String): The current status of the ticket (e.g., "waiting", "serving", "completed").
    - **Use Case**:
      - Tracks all tickets generated by the system.
      - Used by services like `GenerateTicketsFunction`, `FetchTicketsFunction`, and `NextTicketFunction`.



This shows the tickets that are being serviced or waiting at the moment.

- **Counters Table**:
  - **Schema**:
    - **Partition Key**: `counterID` (String) - A unique identifier for each service counter.
    - **Attributes**:
      - `password` (String): Authentication credentials for the counter.
      - `serviceType` (String): The service type the counter is responsible for.
  - **Use Case**:
    - Validates counter authentication and assign tickets to the respective counter.
    - Used by the `AuthenticateCounterFunction` and `NextTicketFunction`.



This shows the counters registered with their passwords and what can they serve.

C1 → Telles → password counter123

C2 → Customer Service → password counter456

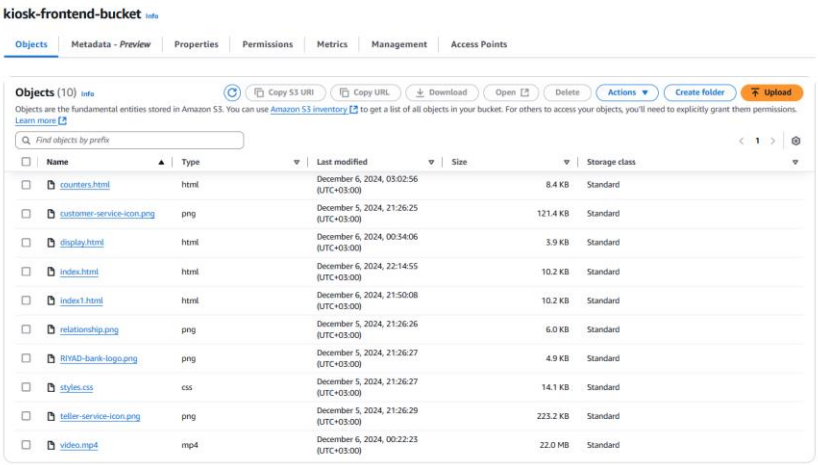
C3 → Relationship Officer → password counter789

- **Advantages of DynamoDB**:
    - Serverless: Fully managed with no need for server provisioning.
    - Scalability: Handles variable workloads seamlessly.
    - Cost-Effective: Pay-per-use model minimizes costs.
    - High Availability: Ensures the system is operational even during peak usage.

---

2. S3 (Amazon Simple Storage Service)

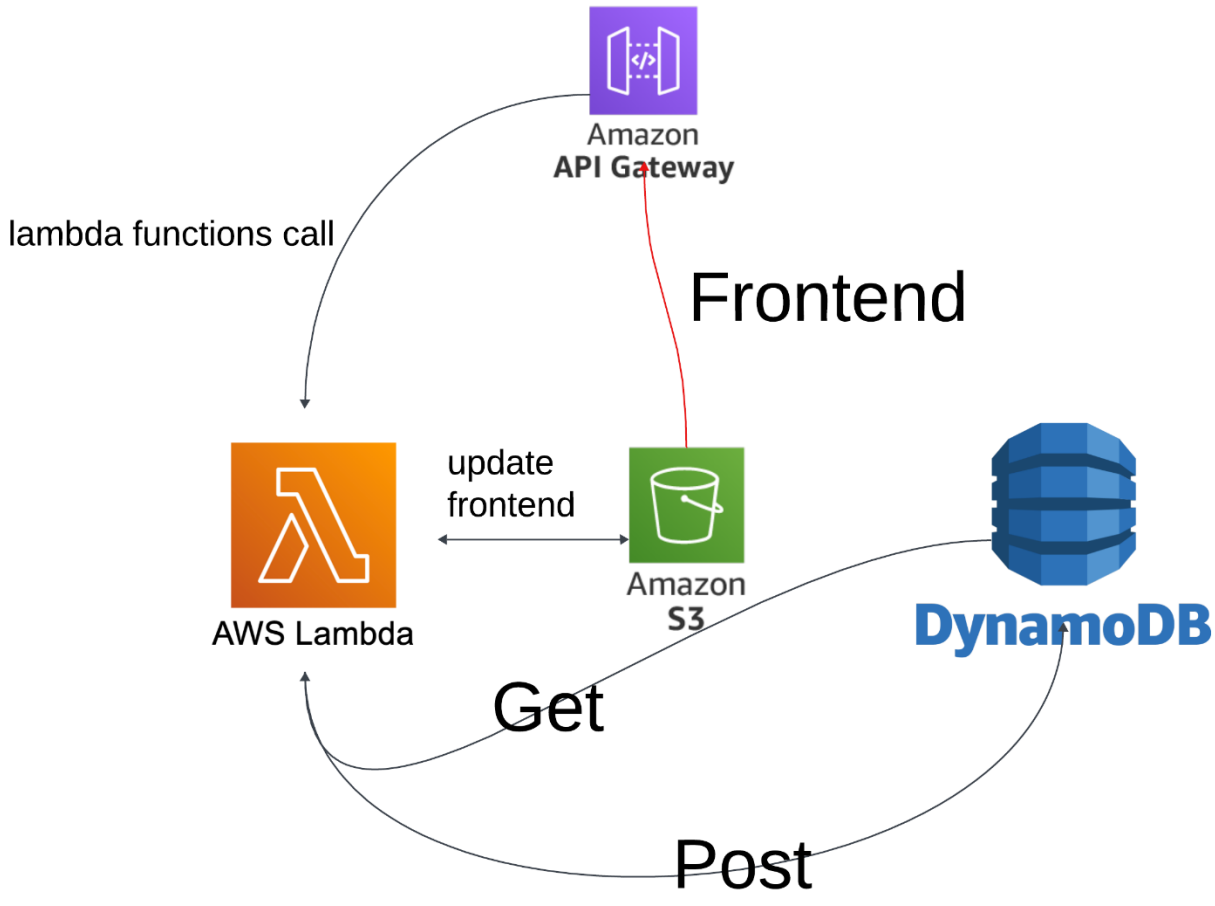
- **Purpose**:
  S3 is used for hosting the front-end web application and storing any supplementary project resources.
- **Buckets Used**:
    - **Frontend Bucket**:
        - Stores the static files for the bank's kiosk frontend (HTML, CSS, JavaScript).
        - Configured for public access to serve the web application.
        - Linked to the API Gateway for dynamic backend integration.
- **Configuration**:
    - **Static Website Hosting**:
        - The bucket is configured to host the web application by enabling static website hosting.
    - **Security**:
        - Policies restrict access to the bucket, ensuring only authorized services (e.g., the frontend) can read or write data.
        - Encrypted data at rest using server-side encryption (SSE-S3 or SSE-KMS).
- **Advantages of S3**:
    - Global Access: Provides fast and secure access to frontend resources globally.
    - Scalability: Automatically scales to handle any number of users accessing the application.
    - Durability: Offers 99.999999999% (11 9's) durability for stored data.



Interaction Between Components

- **DynamoDB**:
  - Stores and retrieves ticket and counter-related data based on API requests.
  - Lambda functions interact with DynamoDB via AWS SDK to perform operations (e.g., `PutCommand`, `GetCommand`).
- **S3**:
  - Serves as the frontend interface for end-users to interact with the system.
  - Acts as a bridge between the user interface and the API Gateway, ensuring seamless communication with backend services.

By integrating DynamoDB for backend data management and S3 for frontend hosting, the system achieves a balance of efficiency, reliability, and scalability.



This shows an overview of the interactions between the Dynamo DB and S3, they don't communicate directly but they do through AWS Lambda.

Workload Characterization:

HTTP Request	HTTP Method	Json Input Format
Generate Ticket	**Post**	{ "serviceType": "Tellers", "status": "waiting" }
Generate Ticket	**Get**	**None (It retrieves all tickets in the DB)**
Fetch Ticket	**Post**	{ "serviceType": "Tellers" }
Fetch Ticket	**Get**	**None (It retrieves all tickets in the DB)**
Next Ticket	**Post**	{ "counterID": "c1", "serviceType": "Tellers" }
Next Ticket	**Get**	{ "counterID": "c1" }
Authenticate Ticket	**Post**	{ "counterID": "c1", "password": "counter123" }

System Architecture:

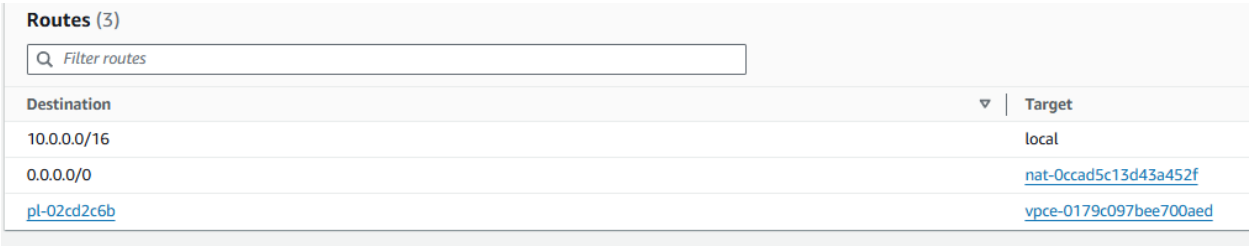
General Architecture Overview

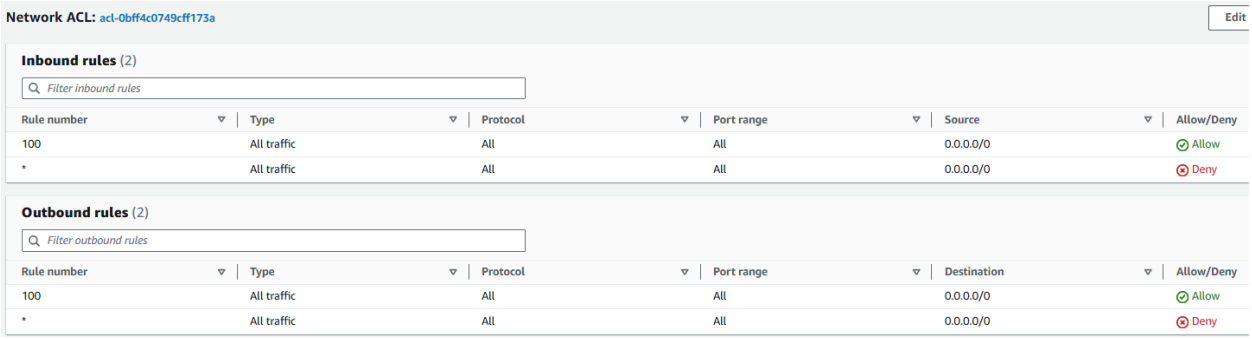
1. **Frontend**:
    - **S3 Bucket**: Hosts the frontend (HTML, CSS, JS) of the application.
    - **Integration**: The frontend communicates with the backend via **API Gateway**.
2. **Backend**:
    - **API Gateway**:
        - Acts as the main entry point for all frontend requests.
        - Routes requests to respective **AWS Lambda Functions**.
    - **Lambda Functions**:
        - **GenerateTicketsFunction**: Creates a ticket in DynamoDB.
        - **FetchTicketsFunction**: Retrieves ticket data.
        - **NextTicketFunction**: Processes the next ticket in the queue.
        - **AuthenticateCounterFunction**: Authenticates counters based on credentials in DynamoDB.
    - **DynamoDB**:

        - Stores tickets and counter authentication details.
        - Two tables: `Tickets` and `Counters`.
    - **VPC**:
        - **Public Subnet**:
            - Internet Gateway for the frontend (S3 bucket) and public-facing Lambda functions.
            - NAT Gateway for enabling private subnet communication.
        - **Private Subnet**:
            - Houses Lambda functions requiring access to DynamoDB via the **VPC Endpoint**.
            - DynamoDB Gateway Endpoint enables low-latency, private access.

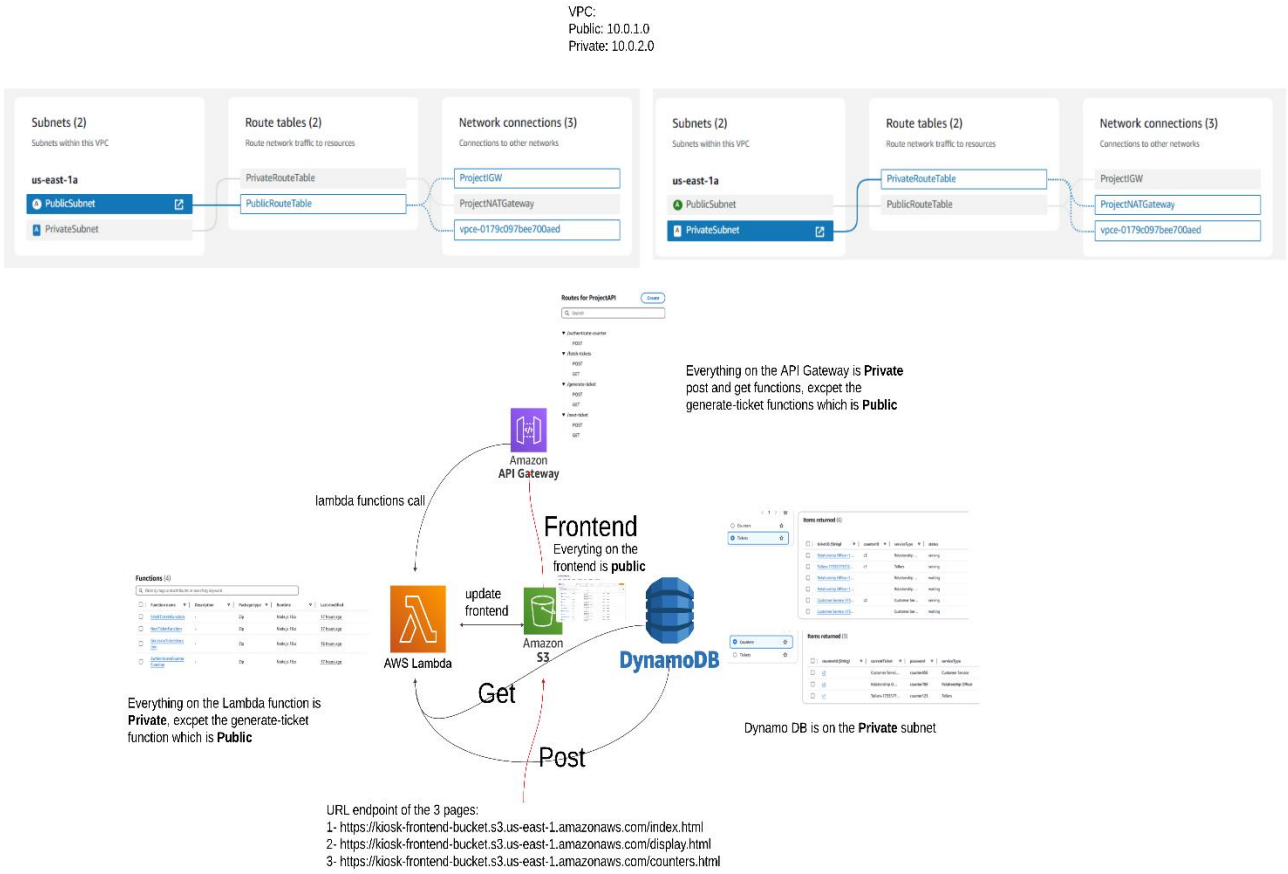
Private Subnet Implementation:

The private subnet is truly private because it does not have a direct route to the **Internet Gateway**, preventing any inbound traffic from the internet. Instead, outbound traffic is routed through a **NAT Gateway**, allowing instances to access external services without exposing them to the internet. Additionally, a **DynamoDB Gateway Endpoint** enables secure, private communication with DynamoDB without relying on the public internet. While Network ACL rules currently allow all traffic, the Route Table ensures no direct internet access, making the subnet private and secure.





3. **Communication**:
    - Frontend (S3) → API Gateway → Lambda Functions → DynamoDB.
    - Lambda functions in the private subnet access DynamoDB using the VPC Endpoint.
    - Public-facing Lambda functions handle lightweight tasks without accessing private resources.



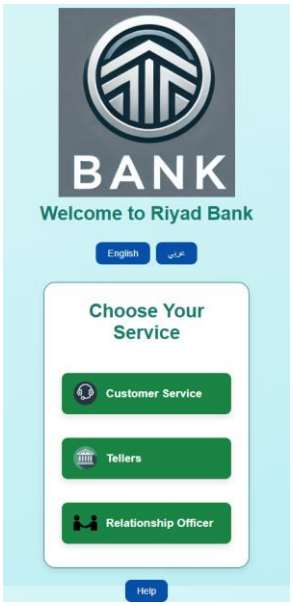
Snapshots of Bank:

Page1 (Kiosk):

URL → https://kiosk-frontend-bucket.s3.us-east-1.amazonaws.com/index.html

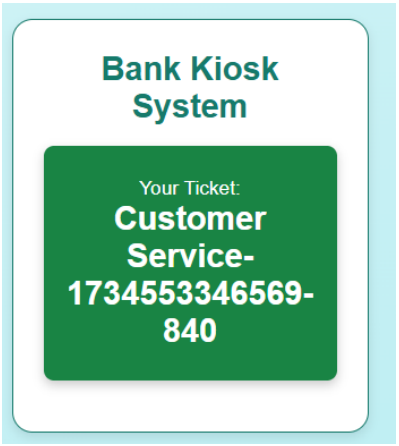
We offer in the Kiosk ticket dispensing in two languages Arabic and English with a Help button to give instruction of how to use the machine if needed, with the three main services:

English Version:



Arabic Version with Help button:

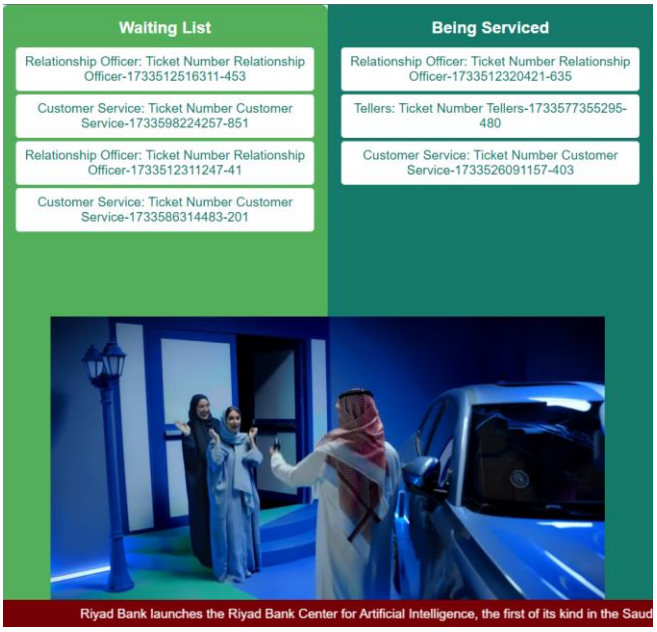
Dispensing a customer service ticket:



Page2 (Display):

URL → https://kiosk-frontend-bucket.s3.us-east-1.amazonaws.com/display.html

In the second page we have the waiting list and the one who is being serviced, with a video and news writing to make waiting a little less boring.

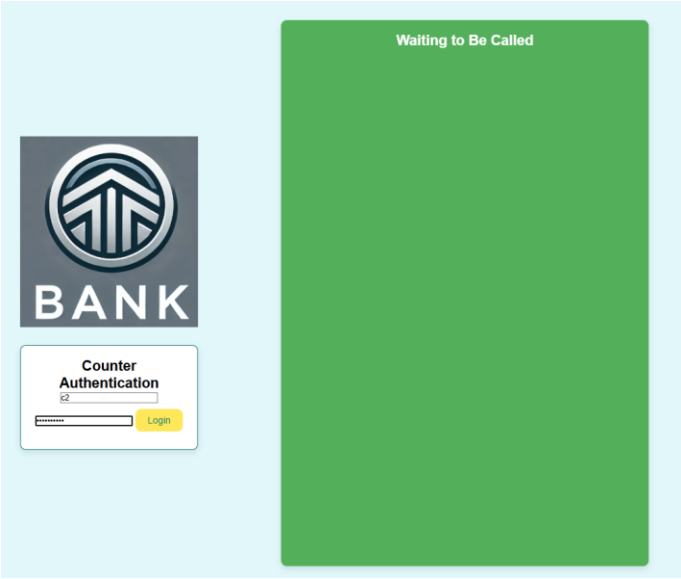


Page3 (Counters):

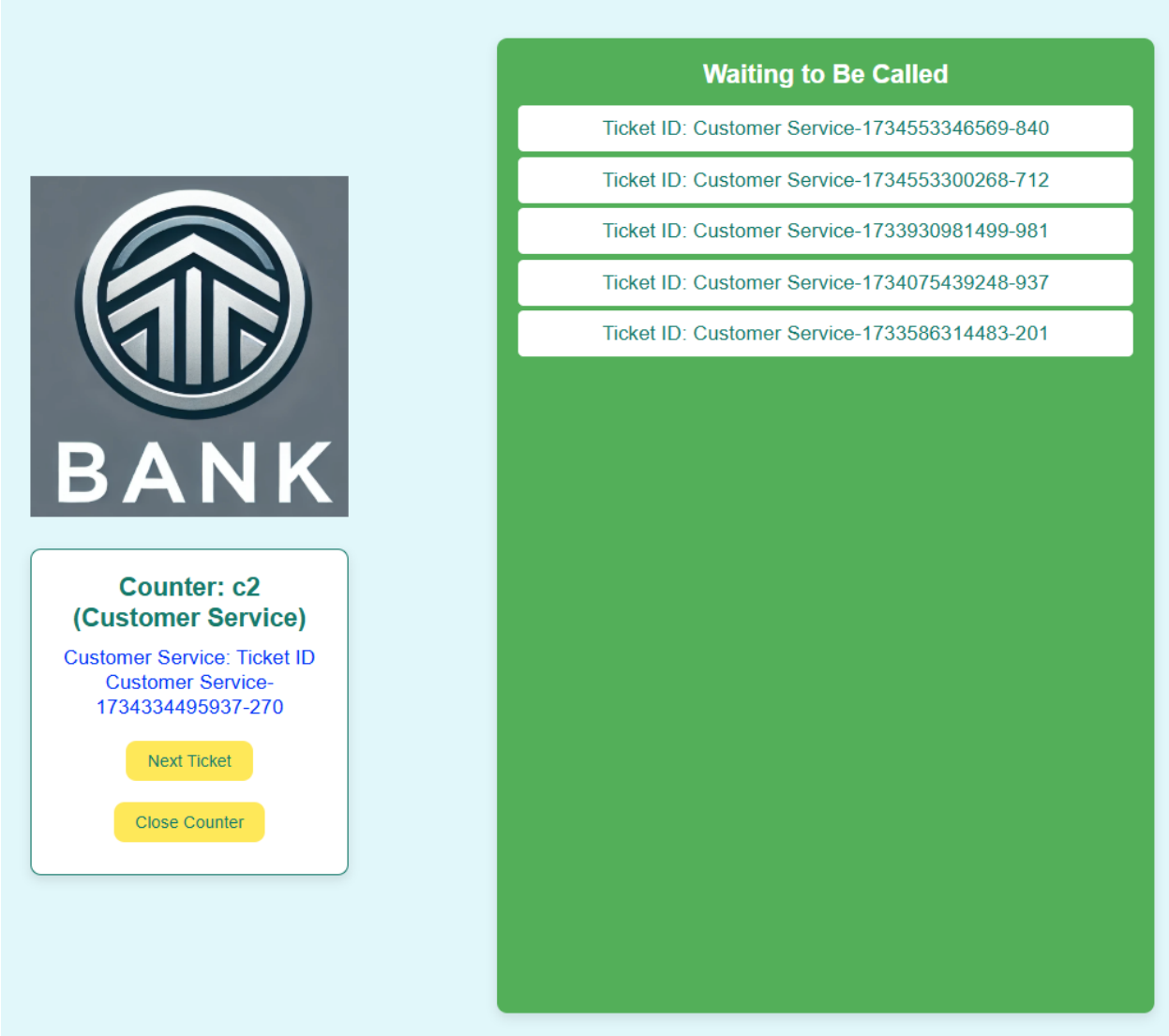
URL → https://kiosk-frontend-bucket.s3.us-east-1.amazonaws.com/counters.html

Finally the counters page for registration and serving customers.

1- Register client:



2- Server for client to dispense tickets:



Appendix:

1- GenerateTicketsFunction Lambda code:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

// Initialize DynamoDB client
const client = new DynamoDBClient();
const dynamodb = DynamoDBDocumentClient.from(client);
```

```
export const handler = async (event) => {
    try {
        const { serviceType, status } = JSON.parse(event.body || "{}");

        // Validate inputs
        if (!serviceType || !status) {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: "Missing serviceType or status" }),
            };
        }

        const validStatuses = ["waiting", "serving", "completed"];
        if (!validStatuses.includes(status)) {
            return {
                statusCode: 400,
                body: JSON.stringify({ message: "Invalid status value" }),
            };
        }

        // Generate a unique ticketID using the current timestamp and a random number
        const timestamp = Date.now(); // Current time in milliseconds
        const randomNum = Math.floor(Math.random() * 1000); // Random number (0-999)
        const ticketID = `${serviceType}-${timestamp}-${randomNum}`;

        // Create the item for DynamoDB
        const ticket = {
            ticketID, // Partition Key
            serviceType,
            status,
        };

        // Store the ticket in DynamoDB
        await dynamodb.send(
            new PutCommand({
                TableName: "Tickets",
                Item: ticket,
            })
        );

        // Return success response
        return {
            statusCode: 200,
            body: JSON.stringify({ ticketID, status }),
        };
```

```
    } catch (error) {
        console.error("Error creating ticket:", error);
        return {
            statusCode: 500,
            body: JSON.stringify({ message: "Failed to create ticket", error: error.message }),
        };
    }
};
```

2- FetchTicketsFunction Lambda code:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { ScanCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient();
const dynamodb = DynamoDBDocumentClient.from(client);

export const handler = async () => {
    try {
        const params = {
            TableName: "Tickets",
        };

        console.log("Scanning DynamoDB table with params:", params);

        const result = await dynamodb.send(new ScanCommand(params));

        console.log("Scan result:", result);

        return {
            statusCode: 200,
            headers: {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*",
            },
            body: JSON.stringify(result.Items),
        };
    } catch (error) {
        console.error("Error fetching tickets:", error);
        return {
            statusCode: 500,
            body: JSON.stringify({ message: "Failed to fetch tickets", error: error.message }),
        };
    }
};
```

3- NextTicketsFunction Lambda code:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand, UpdateCommand, DeleteCommand, GetCommand } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient();
const dynamoDB = DynamoDBDocumentClient.from(client);

export const handler = async (event) => {
    try {
        console.log("Incoming event:", JSON.stringify(event)); // Log the incoming event
        const { counterID, serviceType } = JSON.parse(event.body || "{}");

        console.log("Parsed input:", { counterID, serviceType });

        if (!counterID) {
            throw new Error("Missing counterID.");
        }

        // Fetch the counter's details from the Counters table
        const getCounterParams = {
            TableName: "Counters",
            Key: { counterId: counterID },
        };

        console.log("Fetching counter details with params:", getCounterParams);
        const counterData = await dynamoDB.send(new GetCommand(getCounterParams));
        console.log("Counter data:", JSON.stringify(counterData));

        if (!counterData.Item) {
            return {
                statusCode: 401,
                body: JSON.stringify({ message: "Unauthorized: Counter not found." }),
            };
        }

        // Use the exact `serviceType` from the Counters table
        const exactServiceType = counterData.Item.serviceType;
        console.log("Exact service type:", exactServiceType);

        // Fetch the currently served ticket from the Counters table
        if (counterData.Item.currentTicket) {
            const currentTicketID = counterData.Item.currentTicket;
```

```
            // Delete the current ticket from the Tickets table
            const deleteParams = {
                TableName: "Tickets",
                Key: { ticketID: currentTicketID },
            };
            console.log("Deleting currently served ticket with params:", deleteParams);
            await dynamoDB.send(new DeleteCommand(deleteParams));
            console.log(`Ticket ${currentTicketID} removed from the database.`);

            // Clear the current ticket from the Counters table
            const clearCurrentTicketParams = {
                TableName: "Counters",
                Key: { counterId: counterID },
                UpdateExpression: "REMOVE #currentTicket",
                ExpressionAttributeNames: {
                    "#currentTicket": "currentTicket",
                },
            };
            console.log("Clearing current ticket for counter with params:",
clearCurrentTicketParams);
            await dynamoDB.send(new UpdateCommand(clearCurrentTicketParams));
            console.log(`Counter ${counterID} current ticket cleared.`);
        }

        // Scan for the next waiting ticket for the service type
        let scanParams = {
            TableName: "Tickets",
            FilterExpression: "#st = :waiting AND #svc = :service",
            ExpressionAttributeNames: {
                "#st": "status",
                "#svc": "serviceType",
            },
            ExpressionAttributeValues: {
                ":waiting": "waiting",
                ":service": exactServiceType,
            },
        };

        let lastEvaluatedKey = null;
        let ticket = null;

        console.log("Starting scan for tickets...");
        do {
            if (lastEvaluatedKey) {
```

```
            scanParams.ExclusiveStartKey = lastEvaluatedKey;
        }

        const scanResult = await dynamoDB.send(new ScanCommand(scanParams));
        console.log("Scan result:", JSON.stringify(scanResult));

        // Check if any items match
        if (scanResult.Items && scanResult.Items.length > 0) {
            ticket = scanResult.Items[0]; // Take the first matching ticket
            break;
        }

        lastEvaluatedKey = scanResult.LastEvaluatedKey;
    } while (lastEvaluatedKey);

    if (!ticket) {
        return {
            statusCode: 404,
            body: JSON.stringify({ message: "No tickets available for this service type." }),
        };
    }

    console.log("Fetched ticket:", JSON.stringify(ticket));

    // Update ticket status to "serving"
    const updateParams = {
        TableName: "Tickets",
        Key: { ticketID: ticket.ticketID },
        UpdateExpression: "SET #st = :serving, #ctr = :counter",
        ExpressionAttributeNames: {
            "#st": "status",
            "#ctr": "counterID",
        },
        ExpressionAttributeValues: {
            ":serving": "serving",
            ":counter": counterID,
        },
        ReturnValues: "ALL_NEW",
    };

    console.log("Updating ticket with params:", JSON.stringify(updateParams));
    const updatedTicket = await dynamoDB.send(new UpdateCommand(updateParams));
    console.log("Updated ticket:", JSON.stringify(updatedTicket));

    // Update the counter's current ticket in the Counters table
```

```
            const updateCounterParams = {
                TableName: "Counters",
                Key: { counterId: counterID },
                UpdateExpression: "SET #currentTicket = :ticket",
                ExpressionAttributeNames: {
                    "#currentTicket": "currentTicket",
                },
                ExpressionAttributeValues: {
                    ":ticket": ticket.ticketID,
                },
            };
            console.log("Updating counter with params:", JSON.stringify(updateCounterParams));
            await dynamoDB.send(new UpdateCommand(updateCounterParams));

            return {
                statusCode: 200,
                body: JSON.stringify(updatedTicket.Attributes),
            };
        } catch (error) {
            console.error("Error handling ticket:", error);
            return {
                statusCode: 500,
                body: JSON.stringify({ message: "Failed to process the ticket", error: error.message
    }),
            };
        }
    };
```

4- AuthinticateCounterFunction Lambda code:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

// Initialize the DynamoDB client
const client = new DynamoDBClient();
const dynamo = DynamoDBDocumentClient.from(client);

export const handler = async (event) => {
    try {
        // Parse the input
        const { counterId, password } = JSON.parse(event.body);

        if (!counterId || !password) {
            return {
```

```
                statusCode: 400,
                body: JSON.stringify({ message: "Missing counterId or password" }),
            };
        }

        // Fetch counter details from DynamoDB
        const params = {
            TableName: "Counters",
            Key: { counterId },
        };

        const result = await dynamo.send(new GetCommand(params));

        if (!result.Item) {
            return {
                statusCode: 404,
                body: JSON.stringify({ message: "Counter not found" }),
            };
        }

        const { password: storedPassword, serviceType } = result.Item;

        // Simple password comparison
        if (password !== storedPassword) {
            return {
                statusCode: 401,
                body: JSON.stringify({ message: "Invalid credentials" }),
            };
        }

        // Return success
        return {
            statusCode: 200,
            body: JSON.stringify({ isAuthenticated: true, counterId, serviceType }),
        };
    } catch (error) {
        console.error("Error authenticating counter:", error);
        return {
            statusCode: 500,
            body: JSON.stringify({ message: "Internal server error" }),
        };
    }
};
```

Frontend:

Kiosk:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Bank Kiosk</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <body class="index-page">
      <!-- Riyad Bank Logo at the Top Center -->
   <div class="top-logo-container">
      <img src="RIYAD-bank-logo.png" alt="Riyad Bank Logo" class="bank-logo">
   </div>

   <!-- Welcome Section -->
   <div id="welcome-section">
      <h1>Welcome to Riyad Bank</h1>
   </div>

   <div id="language-switch">
      <button onclick="switchLanguage('en')" class="language-btn">English</button>
      <button onclick="switchLanguage('ar')" class="language-btn">عربي</button>
   </div>




      <!-- Choose Service Section -->
      <div id="choose-service-section" class="kiosk-container">
         <h1 id="choose-service-header">Choose Your Service</h1>
         <button id="customer-service-btn" class="service-btn">
            <img src="customer-service-icon.png" alt="Customer Service Icon" class="service-icon"> Customer Service
         </button>
         <button id="tellers-btn" class="service-btn">
```

```
            <img src="teller-service-icon.png" alt="Teller Service Icon" class="service-icon"> Tellers
        </button>
        <button id="relationship-officer-btn" class="service-btn">
            <img src="relationship.png" alt="Relationship Officer Icon" class="service-icon"> Relationship Officer
        </button>
    </div>



    <!-- Turn Number Display Section (Initially Hidden) -->
    <div id="turn-number-section" class="kiosk-container" style="display: none;">
        <h1>Bank Kiosk System</h1>
        <div class="display">
            <p>Your Turn Number:</p>
            <h2 id="turn-number">--</h2>
        </div>
    </div>

    <!-- Help Modal -->
    <button id="help-btn" class="help-btn">Help</button>
    <div id="help-modal" class="modal" style="display: none;">
        <div class="modal-content">
            <h2>
                How to Use the Kiosk
                <span class="close-btn" onclick="closeHelpModal()">&times;</span>
            </h2>
            <p>1. Select your desired service from the options above.</p>
            <p>2. Wait for your turn number to be displayed.</p>
            <p>3. Proceed to the counter when your number is called.</p>
        </div>
    </div>

  <script>
    const apiUrl = "https://dgq0s9dpr3.execute-api.us-east-1.amazonaws.com/prod/generate-ticket";
    let turnNumber = 0;

// Function to show the turn number and update the waiting list
// Event listeners for service buttons
```

```
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("customer-service-btn").addEventListener("click", () => {
        showTurnNumber("Customer Service");
    });

    document.getElementById("tellers-btn").addEventListener("click", () => {
        showTurnNumber("Tellers");
    });

    document.getElementById("relationship-officer-btn").addEventListener("click", () => {
        showTurnNumber("Relationship Officer");
    });

    // Update the waiting list when the page loads
    updateWaitingList();
});



// Function to show the turn number and update the waiting list
// Object to track ticket numbers for each service
const ticketCounters = {
    "Customer Service": 0,
    "Tellers": 0,
    "Relationship Officer": 0,
};

async function showTurnNumber(serviceType) {
    try {
        const response = await fetch(apiUrl, {
            method: "POST",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify({ serviceType, status: "waiting" }),
        });

        if (!response.ok) {
```

```
            throw new Error("Failed to generate ticket");
        }

        const data = await response.json();

        // Extract ticket ID from the API response
        const ticketNumber = data.ticketID;

        // Display the ticket information
        document.getElementById("turn-number-section").style.display = "block";
        document.getElementById("turn-number-section").innerHTML = `
            <h1>Bank Kiosk System</h1>
            <div class="display">
                <p>Your Ticket:</p>
                <h2 id="turn-number">${ticketNumber}</h2>
            </div>
        `;

        // Hide the "Choose Your Service" section
        document.getElementById("choose-service-section").style.display = "none";

        // Automatically switch back to the "Choose Your Service" section after 5 seconds
        setTimeout(() => {
            document.getElementById("turn-number-section").style.display = "none";
            document.getElementById("choose-service-section").style.display = "block";
        }, 5000);
    } catch (error) {
        console.error("Error generating ticket:", error);
        alert("An error occurred while generating your ticket. Please try again.");
    }
}
```

```
// Function to update the Waiting List UI
function updateWaitingList() {
    const waitingListElem = document.getElementById('waiting-list');
    if (!waitingListElem) return;
    waitingListElem.innerHTML = '';

    // Display each ticket in the waiting list with service type
    waitingList.forEach(({ number, service }) => {
        const li = document.createElement('li');
        li.textContent = `${service}: Ticket Number ${number}`;
        waitingListElem.appendChild(li);
    });
}



// Language content mapping
const languageContent = {
    en: {
        welcome: "Welcome to Riyad Bank Kiosk",
        chooseService: "Choose Your Service",
        customerService: "Customer Service",
        tellers: "Tellers",
        relationshipOfficer: "Relationship Officer", // Added
        help: "Help",
        helpModalTitle: "How to Use the Kiosk",
        helpModalSteps: [
            "Select your desired service from the options above.",
            "Wait for your turn number to be displayed.",
            "Proceed to the counter when your number is called.",
        ],
    },
    ar: {
        welcome: "مرحبًا بكم في كشك بنك الرياض",
        chooseService: "اختر خدمتك",
        customerService: "خدمة العملاء",
        tellers: "الصرافين",
        relationshipOfficer: "مسؤول العلاقات", // Added
```

```
    help: "مساعدة",
    helpModalTitle: "كيفية استخدام الكشك",
    helpModalSteps: [
      "اختر الخدمة المطلوبة من الخيارات أعلاه.",
      "انتظر حتى يتم عرض رقم دورك.",
      "توجه إلى الكاونتر عند استدعاء رقمك.",
    ],
  },
};



// Function to switch language
function switchLanguage(lang) {
  // Update the <html> tag's lang attribute for consistency
  document.querySelector("html").setAttribute("lang", lang);

  // Add or remove the "rtl" class for Arabic
  if (lang === "ar") {
    document.body.classList.add("rtl");
  } else {
    document.body.classList.remove("rtl");
  }

  // Update welcome message
  document.querySelector("#welcome-section h1").innerText =
    languageContent[lang].welcome;

  // Update button text for services
  document.querySelector("#customer-service-btn").innerHTML = `
    <img src="customer-service-icon.png" alt="${languageContent[lang].customerService} Icon" class="service-icon">
    ${languageContent[lang].customerService}
  `;
  document.querySelector("#tellers-btn").innerHTML = `
    <img src="teller-service-icon.png" alt="${languageContent[lang].tellers} Icon" class="service-icon">
    ${languageContent[lang].tellers}
```

```
`;
 document.querySelector("#relationship-officer-btn").innerHTML = `
    <img src="relationship.png" alt="${languageContent[lang].relationshipOfficer} Icon" class="service-icon">
    ${languageContent[lang].relationshipOfficer}
`;
// Update "Choose Your Service" header dynamically
document.querySelector("#choose-service-header").innerText =
    languageContent[lang].chooseService;

// Update Help Button
document.getElementById("help-btn").innerText = languageContent[lang].help;

// Update Help Modal Title and Steps
const helpModal = document.querySelector("#help-modal");
helpModal.querySelector("h2").innerHTML = `
    ${languageContent[lang].helpModalTitle}
    <span class="close-btn" onclick="closeHelpModal()">&times;</span>
`;

const modalSteps = helpModal.querySelectorAll("p");
modalSteps.forEach((p, index) => {
    p.innerText = languageContent[lang].helpModalSteps[index];
});

// Update ticket screen content if visible
const ticketSection = document.getElementById("turn-number-section");
if (ticketSection.style.display === "block") {
    const ticketScreenContent = {
        en: {
            title: "Bank Kiosk System",
            ticketLabel: "Your Ticket:",

        },
        ar: {
            title: "نظام الكشك المصرفي",
            ticketLabel: "تذكرتك:",
```

```
        },
    };

    ticketSection.querySelector("h1").innerText = ticketScreenContent[lang].title;

    const ticketLabel = ticketSection.querySelector("#turn-number");
    const ticketNumber = ticketLabel.innerText.split(":")[1].trim(); // Keep ticket number
    ticketLabel.innerText = `${ticketScreenContent[lang].ticketLabel} ${ticketNumber}`;


  }
}



// Open Help Modal
function openHelpModal() {
  document.getElementById("help-modal").style.display = "flex";
}

// Close Help Modal
function closeHelpModal() {
  document.getElementById("help-modal").style.display = "none";
}

// Event Listener for Help Button
document.getElementById("help-btn").addEventListener("click", openHelpModal);




  </script>
</body>
</html>
```

Display:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Display Area</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="waiting-section">
    <h2>Waiting List</h2>
    <ul id="waiting-list"></ul>
  </div>
  <div class="serviced-section">
    <h2>Being Serviced</h2>
    <ul id="serviced-list"></ul>
  </div>
  <!-- Video -->
  <div class="video-container">
    <video id="myVideo" autoplay muted loop playsinline>
      <source src="video.mp4" type="video/mp4">
      Your browser does not support the video tag.
    </video>
  </div>

  <!-- Moving Text -->
  <div class="moving-text">
    <div>
      Riyad Bank launches the Riyad Bank Center for Artificial Intelligence, the first of its kind in the Saudi banking sector.
      <span> | </span>
      Riyad Bank launches the Riyad Bank SME Index during Bibaan 24.
      <span> | </span>
      Riyad Bank becomes a strategic partner in the 8th edition of the Future Investment Initiative.
    </div>
  </div>

  <script>
```

```
// DOM elements
const waitingListElem = document.getElementById('waiting-list');
const servicedListElem = document.getElementById('serviced-list');

// Fetch tickets from the API
const fetchTicketsApiUrl = "https://dgq0s9dpr3.execute-api.us-east-1.amazonaws.com/prod/fetch-tickets";
async function fetchTickets() {
    try {
        const response = await fetch(fetchTicketsApiUrl, {
            method: "GET",
            headers: {
                "Content-Type": "application/json",
            },
        });

        if (!response.ok) {
            throw new Error("Failed to fetch tickets");
        }

        const tickets = await response.json();

        // Separate tickets by status
        const waitingList = tickets.filter(ticket => ticket.status === "waiting");
        const servicedList = tickets.filter(ticket => ticket.status === "serving");

        // Update the UI
        updateWaitingListDisplay(waitingList);
        updateServicedListDisplay(servicedList);
    } catch (error) {
        console.error("Error fetching tickets:", error);
        alert("Failed to fetch ticket data. Please try again later.");
    }
}

// Update waiting list display
function updateWaitingListDisplay(waitingList) {
    waitingListElem.innerHTML = ''; // Clear list
```

```
        waitingList.forEach(ticket => {
            const li = document.createElement('li');
            li.textContent = `${ticket.serviceType}: Ticket Number ${ticket.ticketID}`;
            waitingListElem.appendChild(li);
        });
    }

    // Update serviced list display
    function updateServicedListDisplay(servicedList) {
        servicedListElem.innerHTML = ''; // Clear list
        servicedList.forEach(ticket => {
            const li = document.createElement('li');
            li.textContent = `${ticket.serviceType}: Ticket Number ${ticket.ticketID}`;
            servicedListElem.appendChild(li);
        });
    }

    // On page load
    document.addEventListener("DOMContentLoaded", () => {
        fetchTickets(); // Fetch tickets immediately when the page loads

        // Fetch tickets every 1 second
        setInterval(fetchTickets, 1000);

        // Play video
        const video = document.getElementById('myVideo');
        if (video) {
            video.muted = true;
        }
    });
  </script>
</body>
</html>
```

Counters:

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Counters</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="main-container">
    <!-- Authentication and Counter Section (Right Half) -->
    <div class="auth-counter-container">
      <img src="RIYAD-bank-logo.png" alt="Riyad Bank Logo" class="bank-logo">

      <!-- Authentication Section -->
      <div id="auth-section">
        <h2>Counter Authentication</h2>
        <input type="text" id="auth-input" placeholder="Enter Counter ID (e.g., c1)">
        <input type="password" id="password-input" placeholder="Enter Password">
        <button id="auth-btn">Login</button>
        <p id="auth-error" style="display: none; color: red;">Invalid ID or Password. Please try again.</p>
      </div>

      <!-- Counters Container (Initially Hidden) -->
      <div class="counter" id="counter-container" style="display: none;">
        <h2 id="counter-title"></h2>
        <p id="ticket-display">No Ticket</p>
        <button id="next-ticket-btn">Next Ticket</button>
        <button id="close-counter-btn">Close Counter</button>
      </div>
    </div>

    <!-- Waiting Section (Left Half) -->
    <div class="waiting-section">
      <h2>Waiting to Be Called</h2>
      <ul id="waiting-list"></ul>
    </div>
  </div>
```

```
<script>
    const authApiUrl = "https://dgq0s9dpr3.execute-api.us-east-1.amazonaws.com/prod/authenticate-counter";
    const nextTicketApiUrl = "https://dgq0s9dpr3.execute-api.us-east-1.amazonaws.com/prod/next-ticket";
    const fetchTicketsApiUrl = "https://dgq0s9dpr3.execute-api.us-east-1.amazonaws.com/prod/fetch-tickets";
    let counterId = null;
    let serviceType = null;

    // Authenticate Counter User
    async function authenticateUser() {
        const authInput = document.getElementById("auth-input").value.trim().toLowerCase();
        const passwordInput = document.getElementById("password-input").value.trim();

        try {
            const response = await fetch(authApiUrl, {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ counterId: authInput, password: passwordInput }),
            });

            const result = await response.json();

            if (response.status === 200 && result.isAuthenticated) {
                counterId = authInput; // Set counter ID
                serviceType = result.serviceType; // Set service type
                console.log(`Authenticated: Counter ${counterId}, Service Type: ${serviceType}`);

                initializeCounter(); // Initialize UI for the counter
                fetchTickets(serviceType); // Fetch tickets for the authenticated counter's service type

                // Refresh tickets every 5 seconds
                setInterval(() => fetchTickets(serviceType), 5000);
            } else {
                displayAuthError("Invalid ID or Password.");
            }
        } catch (error) {
            console.error("Error during authentication:", error);
```

```
        displayAuthError("Authentication failed. Please try again.");
    }
}

// Initialize Counter After Successful Authentication
function initializeCounter() {
    document.getElementById("auth-section").style.display = "none";
    document.getElementById("counter-container").style.display = "block";
    document.getElementById("counter-title").textContent = `Counter: ${counterId} (${serviceType})`;
    document.getElementById("auth-error").style.display = "none";
}

// Display Authentication Error
function displayAuthError(message) {
    const errorElement = document.getElementById("auth-error");
    errorElement.textContent = message;
    errorElement.style.display = "block";
}

// Fetch Waiting Tickets for the Counter's Service Type
async function fetchTickets(serviceType) {
    try {
        const response = await fetch(fetchTicketsApiUrl, {
            method: "GET",
            headers: {
                "Content-Type": "application/json",
            },
        });

        if (!response.ok) {
            throw new Error("Failed to fetch tickets");
        }

        const tickets = await response.json();

        // Filter tickets for the specific service type
        const waitingList = tickets.filter(ticket => ticket.status === "waiting" && ticket.serviceType === serviceType);
```

```
        // Update the waiting list display
        updateWaitingListDisplay(waitingList);
    } catch (error) {
        console.error("Error fetching tickets:", error);
        alert("Failed to fetch ticket data. Please try again later.");
    }
}

// Update Waiting List UI
function updateWaitingListDisplay(waitingList) {
    const waitingListElem = document.getElementById("waiting-list");
    waitingListElem.innerHTML = ""; // Clear the list

    waitingList.forEach(ticket => {
        const listItem = document.createElement("li");
        listItem.textContent = `Ticket ID: ${ticket.ticketID}`;
        waitingListElem.appendChild(listItem);
    });
}
// Get the Next Ticket for the Counter
async function getNextTicket() {
    if (!counterId || !serviceType) {
        alert("Counter is not authenticated.");
        return;
    }
    try {
        const response = await fetch(nextTicketApiUrl, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ counterID: counterId, serviceType }),
        });

        const data = await response.json();
        if (response.status === 200) {
            updateTicketDisplay(data.ticketID, data.serviceType);
            fetchTickets(serviceType); // Refresh waiting list
```

```
            } else if (response.status === 404) {
                alert("No tickets available for this service type.");
            } else {
                console.error("Failed to fetch next ticket:", data.message);
                alert("Failed to fetch next ticket. Please try again.");
            }
        } catch (error) {
            console.error("Error fetching next ticket:", error);
            alert("An error occurred. Please try again.");
        }
    }

    // Update the Ticket Display for the Counter
    function updateTicketDisplay(ticketID, serviceType) {
        const ticketElement = document.getElementById("ticket-display");
        ticketElement.textContent = `${serviceType}: Ticket ID ${ticketID}`;
        console.log(`Serving Ticket: ${ticketID}`);
    }
    // Close the Counter and Reset
    function closeCounter() {
        document.getElementById("counter-container").style.display = "none";
        document.getElementById("auth-section").style.display = "block";
        document.getElementById("ticket-display").textContent = "No Ticket";
        document.getElementById("auth-input").value = "";
        document.getElementById("password-input").value = "";
        counterId = null;
        serviceType = null;
        console.log("Counter closed and reset.");
    }
    // Initialize Event Listeners
    document.addEventListener("DOMContentLoaded", () => {
        document.getElementById("auth-btn").addEventListener("click", authenticateUser);
        document.getElementById("next-ticket-btn").addEventListener("click", getNextTicket);
        document.getElementById("close-counter-btn").addEventListener("click", closeCounter);
    });
  </script>
</body>
```

```
</html>
```

Conclusion

In this project, we successfully optimized and enhanced Riyadh Bank's ticketing system by implementing a cloud-based, scalable, and redundant solution. Using AWS services such as Lambda, DynamoDB, S3, and API Gateway, we developed an efficient system capable of handling high workloads while ensuring reliability and ease of access. The segregation of services into public and private subnets within a VPC enhanced the system's security and allowed for controlled communication between components. By adopting serverless architecture and leveraging modern cloud capabilities, we reduced infrastructure complexity and operational overhead.

This project not only addressed the primary goal of improving redundancy and optimizing the ticketing process but also served as an excellent opportunity to explore cloud-native development. The experience provided invaluable insights into designing, implementing, and managing robust cloud applications, setting a solid foundation for future projects in scalable and secure environments.