AI-Based Notes Summarizer & Cloud Share - Deployment Guide

This documentation provides a full guide on how to deploy the AI Notes Summarizer & Cloud Share project using AWS services from scratch.

Project Overview

This AI-based Notes Summarizer allows users to upload text or PDF files, summarizes the content using Amazon Bedrock, and stores results securely in DynamoDB. Users can access their summaries through a unique link and get notifications via email.

AWS Services Used

Service	Purpose
Amazon S3	To store frontend HTML/CSS files and user-uploaded text/PDF files
AWS Lambda	Backend logic for handling file uploads, generating summaries
API Gateway	To expose Lambda functions as REST APIs
Amazon DynamoDB	To store summaries with note IDs
Amazon SNS	To send email notifications when summary is ready
Amazon Bedrock	For generating summaries using LLM (e.g., Meta LLaMA 3)
Amazon CloudWatch	For monitoring Lambda logs and alerts
AWS IAM	For managing permissions and roles securely
Amazon Route 53	For custom domain mapping (e.g., raju.rddi.xyz)

AWS Certificate Managers	For SSL certificate to enable HTTPS via CloudFront
AWS Coudfront	For the fast global access the website

🧭 **Quick Summary: Add Each AWS Service**

1. **S3**
    - Create a bucket (e.g., rddi.xyz) note: Bucket name should br same as Domain name eg my domain name is rddi.xyz hence my bucket name is rddi.xyz
    - Enable static website hosting
    - Upload frontend files (upload.html, view-summary.html)
2. **IAM**
    - Create role for Lambda with access to: S3, DynamoDB, SNS, Bedrock, CloudWatch (note:fullaccess)
    - Attach policies (AWSLambdaBasicExecutionRole, etc.)
3. **Lambda**
    - Create two functions:
        - upload-handler: to receive uploaded text and save to DB
        - summary-handler: to fetch summary using Bedrock and store it(note: go to the environment variable sns and table name)
    - Upload .zip code (via console or S3)
4. **API Gateway**
    - Create Build API

- Link endpoints /upload and /get-summary to respective Lambda functions (note : both api attached with lamda function
- Enable CORS(notes:

5. **DynamoDB**
    - Create table: note_summaries
    - Primary key: note_id
6. **SNS**
    - Create topic: NoteSummaryNotification
    - Add your verified email (e.g., raju24devops@gmail.com)
    - Subscribe to topic
7. **Amazon comprehend**
    - Use via SDK in summary Lambda
8. **Route 53**
    - Create hosted zone (e.g., rddi.xyz)
    - Add A-record pointing to CloudFront or S3 bucket
9. **ACM (SSL)**
    - Request public certificate for domain (e.g., raju.rddi.xyz)
    - Use in CloudFront
10. **CloudWatch**

- Automatically created with Lambda logs
- Create log group metric filters and alerts for error

Imoprtant Notes when create lambda function need to configure

1 upload-handler(paste this code into the upload-handler)

import boto3, json, uuid, os

```
from datetime import datetime


# AWS clients
comprehend = boto3.client("comprehend")
dynamodb   = boto3.resource("dynamodb")
sns        = boto3.client("sns")


# Env variables
TABLE_NAME     = os.environ.get("TABLE_NAME", "note_summaries")
SNS_TOPIC_ARN  = os.environ.get("SNS_TOPIC_ARN")          # must
be verified
MAX_CHARS      = int(os.environ.get("MAX_CHARS", "4800"))


table = dynamodb.Table(TABLE_NAME)


# ------------------- Build human-readable summary -------------------------
def build_human_summary(text):
    """Generate a summary paragraph using AWS Comprehend."""


    senti_resp   = comprehend.detect_sentiment(Text=text,
LanguageCode="en")
    ent_resp     = comprehend.detect_entities(Text=text,
LanguageCode="en")
    phrase_resp  = comprehend.detect_key_phrases(Text=text,
LanguageCode="en")
```

```
    sentiment    = senti_resp.get("Sentiment", "NEUTRAL").lower()
    entities     = list({e["Text"] for e in ent_resp.get("Entities", []) if
len(e["Text"]) > 2})
    key_phrases  = list({p["Text"] for p in
phrase_resp.get("KeyPhrases", []) if len(p["Text"]) > 3})

    summary_parts = [f" **Summary Insight**:\n"]
    summary_parts.append(f"The tone of this note is
*{sentiment}*.")

    if entities:
        summary_parts.append(" It talks about " + ", ".join(entities[:5]) +
".")

    if key_phrases:
        summary_parts.append(“ Main ideas include: " + ",
".join(key_phrases[:8]) + ".")

    return "\n".join(summary_parts)

# ------------------- Lambda Entry Point ---------------------------------
def lambda_handler(event, context):
    try:
        body     = json.loads(event.get("body", "{}"))
```

```
    raw_text = body.get("text", "").strip()

    if not raw_text:
        return _resp(400, {"message": "Text not provided."})

    # Trim for Comprehend limit
    text = raw_text[:MAX_CHARS]

    summary = build_human_summary(text)

    if len(summary.strip()) < 10:
        return _resp(500, {"message": "Unable to generate meaningful summary."})

    note_id = str(uuid.uuid4())
    table.put_item(Item={
        "note_id"   : note_id,
        "summary"   : summary,
        "created_at": datetime.utcnow().isoformat()
    })

    # ---------- SNS Notification (if topic exists) ----------
    if SNS_TOPIC_ARN:
        sns.publish(
```

```
            TopicArn = SNS_TOPIC_ARN,
            Subject  = "New Note Summary Created",
            Message  = f"A new summary was created with ID: {note_id}\n\n{summary}"
        )

    return _resp(200, {"note_id": note_id, "summary": summary})

  except Exception as e:
    print("Exception:", e)

    # Failure Notification
    if SNS_TOPIC_ARN:
      sns.publish(
        TopicArn = SNS_TOPIC_ARN,
        Subject  = " Note Summary FAILED",
        Message  = f"Error occurred: {str(e)}"
      )

    return _resp(500, {"message": "Something went wrong.", "error": str(e)})

# ------------------ Response Formatter ---------------------------------
def _resp(code, body):
```

```
    return {
        "statusCode": code,
        "body": json.dumps(body),
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"
        }
    }
```

After paste this code into the upload-handler need to check test go to the **Test section** beside code see **the json format remove all previous code** and paste below given code

```
{
  "body": "{\"text\": \"this is a test note to check if lamda is working or not fine.\"}"
}
```

After the paste click on create new event and give name as you want and save click on test

2 summary-handler(paste this code into the summary-handler)

```
import boto3
import json
```

```
import os

dynamodb = boto3.resource('dynamodb')
TABLE_NAME = os.environ.get("TABLE_NAME")
print("🔧 Using TABLE_NAME:", TABLE_NAME)

def lambda_handler(event, context):
    print(" Incoming event:", json.dumps(event))  # Log full event

    try:
        params = event.get("queryStringParameters") or {}
        note_id = params.get("noteId")

        print(" Extracted note_id:", note_id)  # Debug note_id

        if not note_id:
            return {
                "statusCode": 400,
                "headers": {
                    "Content-Type": "application/json",
                    "Access-Control-Allow-Origin": "*"
                },
                "body": json.dumps({"error": "Missing noteId"})
            }
```

```
    table = dynamodb.Table(TABLE_NAME)
    response = table.get_item(Key={"note_id": note_id})

    print(" DynamoDB response:", response)  # Debug Dynamo response

    if 'Item' not in response:
        return {
            "statusCode": 404,
            "headers": {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            },
            "body": json.dumps({"error": "No summary found for this Note ID."})
        }

    return {
        "statusCode": 200,
        "headers": {
            "Content-Type": "application/json",
            "Access-Control-Allow-Origin": "*"
        },
```

```
            "body": json.dumps({
                "summary": response['Item'].get('summary', '')
            })
        }

    except Exception as e:
        print(" ERROR CAUGHT:", str(e))  # Full error in logs
        return {
            "statusCode": 500,
            "headers": {
                "Content-Type": "application/json",
                "Access-Control-Allow-Origin": "*"
            },
            "body": json.dumps({"error": "Internal Server Error"})
        }
```

After pastinf this code go to **Test section** beside the code and **paste below give code**

```
{
  "queryStringParameters": {
    "noteId": "4660a686-3bf4-47ea-bc6e-80e158984b63"
  }
}
```

After the paste click on create new event and give name as you want and save click on test

Step to configure dynamodb and sns

First go to the upload-handler lambda function scroll down you see **configuration** under the Enviroment variable click on that again click on edit button then show

Key and **value**

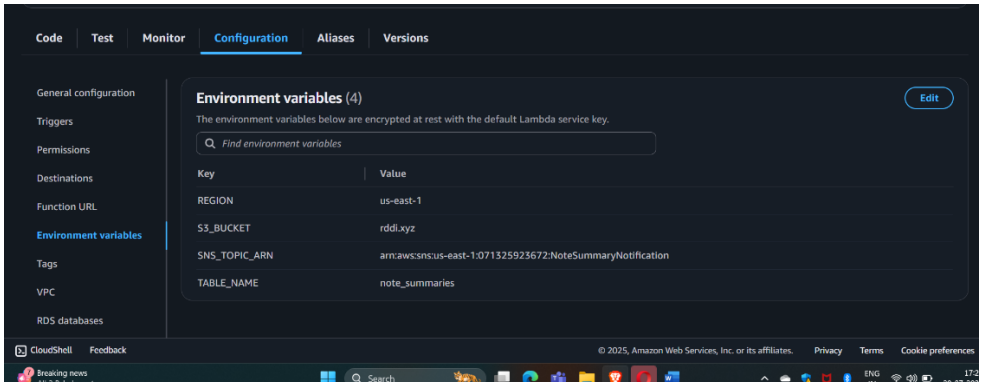
For dynamobd

Key= TABLE_NAME, **value** = note_summaries

For sns service

Key= SNS_TOPIC_ARN , value= arn:aws:sns:us-east-1:071325923672:NoteSummaryNotification (note: paste you sns arn)

Demo it should like this



Now create API and SNS

Steps to Create SNS Topic and Subscription:

1. **Go to AWS SNS Console**
   https://console.aws.amazon.com/sns/v3/home
2. **Create Topic**:

- Click **"Create topic"**
- Choose **Standard**
- **Name**: NoteSummaryNotification
- Click **Create topic**

3. **Create Email Subscription**:
   - On the topic page, click **"Create subscription"**
   - **Protocol**: Email
   - **Endpoint**: your_email@example.com (e.g., raju24devops@gmail.com)
   - Click **Create subscription**
4. **Confirm Email**:
   - Check your email inbox
   - Confirm the subscription by clicking the link in the AWS email
5. **Add Topic ARN to Lambda**:
   - Go to **Lambda > upload-handler**
   - In **Configuration > Environment variables**, add:
     - SNS_TOPIC_ARN = arn:aws:sns:us-east-1:xxxxxxxxxxxx:NoteSummaryNotification

Now Create API

6. Go to **API Gateway Console**

   👉 https://console.aws.amazon.com/apigateway
7. Click **“Create API”**
   - Choose **HTTP API** (faster and simpler)
   - Click **Build**
8. **Set up routes**:

- Example routes:
  - POST /upload → Upload Lambda
  - GET /get-summary → Summary Lambda

9. **Connect Integrations**:
   - For each route, attach respective Lambda function:
     - Upload route → upload-handler
     - Summary route → summary-handler
10. **Deploy API**:
    - Click **Deployments**
    - Create a stage (e.g., prod)
    - Copy the base URL (e.g., https://xyz123.execute-api.us-east-1.amazonaws.com) (note: generated url change with your frontend also)

Enable CORS in API Gateway (for HTTP API)

11. Go to **API Gateway Console**
12. Choose your **HTTP API**
13. In left menu, click **"Routes"**
14. Click your route (e.g., POST /upload)
15. Under **"CORS Configuration"**, click **"Edit"**
16. Set:
    - **Allow origins**: * or your domain (e.g., https://rddi.xyz)
    - **Allow methods**: GET, POST, OPTIONS
    - **Allow headers**: Content-Type
17. Click **"Save changes"**
18. **Deploy the API again**

Create an IAM Role for Lambda

1. Go to the **IAM Console**
   https://console.aws.amazon.com/iam/home#/roles
2. Click **"Create Role"**

3. **Trusted Entity**:
    - Select **AWS service**
    - Use case: **Lambda**
    - Click **Next**
4. **Attach Policies**:
   Choose these managed policies:
    - AmazonS3FullAccess
    - AmazonDynamoDBFullAccess
    - AmazonSNSFullAccess
    - AmazoncomprehendFullAccess *(if not available, create custom policy)*
    - CloudWatchFullAccess
5. **Review & Create**:
    - Role name: ai-notes-lambda-role
    - Click **Create Role**

🔧 Step 2: Attach Role to Lambda Functions

1. Go to **Lambda Console**
2. Select upload-handler function
3. Click **Configuration → Permissions**
4. Under **Execution role**, click the role name
5. Replace or make sure it's set to ai-notes-lambda-role

Repeat for the summary-handler function.

Now frontend code

upload.html

<!DOCTYPE html>

```
<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1" />

  <title>Upload Note | AI Summary</title>

  <script src="https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.11.174/pdf.min.js"></script>

  <script src="https://unpkg.com/@lottiefiles/lottie-player@latest/dist/lottie-player.js"></script>

  <link href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;700&display=swap" rel="stylesheet">

  <style>

    * {

      margin: 0;

      padding: 0;

      box-sizing: border-box;

    }



    body {
```

```
  font-family: 'Montserrat', sans-serif;
  background: #0d1b2a;
  color: #e0e6f8;
  height: 100vh;
  overflow: hidden;
  display: flex;
  justify-content: center;
  align-items: center;
  opacity: 0;
  transition: opacity 0.8s ease-in;
  position: relative;
}

body.loaded {
  opacity: 1;
}

#ai-bg {
  position: fixed;
  top: 0;
```

```
    left: 0;
    z-index: 0;
    width: 100%;
    height: 100%;
    background: radial-gradient(ellipse at bottom, #0f2027,
#203a43, #2c5364);
  }

  #ai-svg-overlay {
    position: fixed;
    inset: 0;
    z-index: 0;
    background:
url('https://www.transparenttextures.com/patterns/circuit-
board.png') repeat;
    opacity: 0.03;
  }

  .container {
    position: relative;
    background: rgba(255, 255, 255, 0.05);
```

```
  padding: 40px 35px;
  border-radius: 20px;
  box-shadow: 0 10px 30px rgba(0,0,0,0.5);
  text-align: center;
  max-width: 480px;
  width: 90%;
  backdrop-filter: blur(20px);
  z-index: 1;
  animation: slideIn 1.2s ease-out;
  transform-style: preserve-3d;
  transition: transform 0.4s;
}

.container:hover {
  transform: perspective(1000px) rotateX(2deg) rotateY(2deg);
}

@keyframes slideIn {
  from { transform: translateY(40px); opacity: 0; }
  to { transform: translateY(0); opacity: 1; }
```

```
}

.project-title {
  font-size: 24px;
  font-weight: 700;
  margin-bottom: 20px;
  animation: glowTitle 2s ease-in-out infinite alternate;
  background: linear-gradient(90deg, #00c9ff, #92fe9d);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

@keyframes glowTitle {
  0% { text-shadow: 0 0 5px #00ffff66; }
  100% { text-shadow: 0 0 20px #00ffffcc; }
}

h1 {
  font-size: 22px;
  margin-bottom: 20px;
```

```
  min-height: 30px;
}

lottie-player {
  margin-bottom: 10px;
}

label {
  display: block;
  font-size: 15px;
  margin-bottom: 10px;
}

input[type="file"] {
  padding: 12px 15px;
  border-radius: 10px;
  border: none;
  background: #334499;
  color: #d8e2ff;
  width: 100%;
```

```
  cursor: pointer;
  margin-bottom: 20px;
}

button {
  background: linear-gradient(90deg, #00d8ff, #00ffab);
  color: #121212;
  font-weight: 700;
  font-size: 16px;
  padding: 14px 0;
  width: 100%;
  border: none;
  border-radius: 12px;
  cursor: pointer;
  transition: 0.3s ease;
  box-shadow: 0 5px 12px #00d8ffaa;
  position: relative;
}

button.loading::after {
```

```
  content: ' ⏳ ';
  position: absolute;
  right: 15px;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  0% { transform: rotate(0deg);}
  100% { transform: rotate(360deg);}
}

button:hover {
  transform: scale(1.03);
}

input#noteIdText {
  border: none;
  background: none;
  color: #00e1ff;
  font-weight: bold;
```

```
  font-size: 16px;
  text-align: center;
  width: 100%;
  margin-top: 8px;
}

.copy-button {
  background: linear-gradient(90deg, #00d8ff, #00ffab);
  color: #121212;
  padding: 10px 16px;
  border: none;
  border-radius: 10px;
  font-weight: bold;
  margin-top: 8px;
  cursor: pointer;
  box-shadow: 0 4px 10px rgba(0, 216, 255, 0.3);
}

.copy-button:hover {
  background: linear-gradient(90deg, #00c9ff, #00bfa6);
```

```
    transform: scale(1.05);
  }

  #status, #error {
    margin-top: 20px;
    font-weight: 600;
    font-size: 15px;
  }

  #status { color: #6dfc8b; }
  #error { color: #ff6e6e; }

  a {
    text-decoration: none;
    font-weight: bold;
    color: #00d8ff;
  }
 </style>
</head>
<body>
```

```
  <canvas id="ai-bg"></canvas>

  <div id="ai-svg-overlay"></div>



  <div class="container">

    <div class="project-title">🧠 AI Notes Summary
Generator</div>



    <img

  src="https://external-
content.duckduckgo.com/iu/?u=https%3A%2F%2Fcdn-icons-
png.flaticon.com%2F512%2F10644%2F10644640.png&f=1&nof
b=1&ipt=0255ac2bc2f4d279d9471bf9a81e3e0359177299d8ceccbd
d6960f76a6e75696"

  alt="AI Circuit Icon"

  style="width:140px; height:auto; margin: 0 auto 20px auto;
display: block;"

/>



    <h1 id="animatedTitle"></h1>

    <form id="uploadForm">

      <label for="fileInput">Select a file (.txt, .pdf):</label>

      <input type="file" id="fileInput" accept=".txt,.pdf" required
/>
```

```
    <button type="submit" id="uploadBtn">Upload</button>
  </form>
  <div id="status"></div>
  <div id="error"></div>
 </div>

 <script>
  const API_ENDPOINT = "https://9vr9ndoril.execute-api.us-east-1.amazonaws.com";
  console.log("Current API:", API_ENDPOINT);

  const uploadForm = document.getElementById('uploadForm');
  const status = document.getElementById('status');
  const error = document.getElementById('error');
  const uploadBtn = document.getElementById('uploadBtn');

  window.onload = () => {
   document.body.classList.add('loaded');
   typeWriter("Upload Your Note");
   animateAIBackground();
```

```
};

function typeWriter(text) {
  let i = 0;
  const target = document.getElementById("animatedTitle");
  function typing() {
    if (i < text.length) {
      target.innerHTML += text.charAt(i);
      i++;
      setTimeout(typing, 70);
    }
  }
  typing();
}

async function extractTextFromPDF(file) {
  const reader = new FileReader();
  return new Promise((resolve, reject) => {
    reader.onload = async function () {
      const typedArray = new Uint8Array(reader.result);
```

```
        const pdf = await pdfjsLib.getDocument({ data: typedArray
}).promise;
        let fullText = '';
        for (let i = 1; i <= pdf.numPages; i++) {
          const page = await pdf.getPage(i);
          const content = await page.getTextContent();
          const strings = content.items.map(item => item.str).join('
');
          fullText += strings + '\n\n';
        }
        resolve(fullText);
      };
      reader.onerror = reject;
      reader.readAsArrayBuffer(file);
    });
  }

  function copyToClipboard() {
    const noteId =
document.getElementById("noteIdText")?.value?.trim();
    if (!noteId) return;
```

```
    const textarea = document.createElement("textarea");
    textarea.value = noteId;
    textarea.setAttribute("readonly", "");
    textarea.style.position = "absolute";
    textarea.style.left = "-9999px";
    document.body.appendChild(textarea);
    textarea.select();
    document.execCommand("copy");
    document.body.removeChild(textarea);

    const copyBtn = document.querySelector(".copy-button");
    copyBtn.textContent = " ✅ Copied!";
    copyBtn.style.background = "#00c47e";
    setTimeout(() => {
      copyBtn.textContent = " 📋 Copy ID";
      copyBtn.style.background = "linear-gradient(90deg, #00d8ff, #00ffab)";
    }, 1500);
  }
```

```
uploadForm.addEventListener('submit', async (e) => {
  e.preventDefault();
  status.innerHTML = '';
  error.textContent = '';
  uploadBtn.classList.add('loading');
  uploadBtn.disabled = true;

  const fileInput = document.getElementById('fileInput');
  if (!fileInput.files.length) {
    error.textContent = 'Please select a file to upload.';
    uploadBtn.classList.remove('loading');
    uploadBtn.disabled = false;
    return;
  }

  const file = fileInput.files[0];
  let extractedText = "";

  if (file.name.endsWith(".pdf")) {
    extractedText = await extractTextFromPDF(file);
```

```
} else {
  const reader = new FileReader();
  extractedText = await new Promise((resolve, reject) => {
    reader.onload = () => resolve(reader.result);
    reader.onerror = reject;
    reader.readAsText(file);
  });
}

const payload = { text: extractedText };

try {
  const response = await fetch(`${API_ENDPOINT}/upload`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(payload)
  });

  const result = await response.json();
  uploadBtn.classList.remove('loading');
```

```
        uploadBtn.disabled = false;


        if (response.ok) {
          const noteId = result.note_id;
          status.innerHTML = `
            ✅ <b>Upload successful!</b><br><br>
            🆔 Your Note ID:<br>
            <input id="noteIdText" value="${noteId}" readonly />
            <button class="copy-button"
onclick="copyToClipboard()">📋 Copy ID</button><br><br>
            <a href="/view-summary.html?noteId=${noteId}">🔍
View Summary</a>
          `;
        } else {
          error.textContent = result.error || 'Upload failed.';
        }
      } catch (err) {
        error.textContent = '⚠️ Network error while uploading.';
        uploadBtn.classList.remove('loading');
        uploadBtn.disabled = false;
```

```
  }
});

function animateAIBackground() {
  const canvas = document.getElementById('ai-bg');
  const ctx = canvas.getContext('2d');
  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;

  let particles = [];
  const total = 100;
  for (let i = 0; i < total; i++) {
    particles.push({
      x: Math.random() * canvas.width,
      y: Math.random() * canvas.height,
      vx: (Math.random() - 0.5) * 0.6,
      vy: (Math.random() - 0.5) * 0.6,
      radius: Math.random() * 2 + 1
    });
  }
```

```
function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.fillStyle = '#00ffff';
  particles.forEach(p => {
    ctx.beginPath();
    ctx.arc(p.x, p.y, p.radius, 0, Math.PI * 2);
    ctx.fill();
  });

  for (let i = 0; i < total; i++) {
    for (let j = i + 1; j < total; j++) {
      let dx = particles[i].x - particles[j].x;
      let dy = particles[i].y - particles[j].y;
      let dist = Math.sqrt(dx * dx + dy * dy);
      if (dist < 100) {
        ctx.beginPath();
        ctx.strokeStyle = `rgba(0,255,255,${1 - dist / 100})`;
        ctx.moveTo(particles[i].x, particles[i].y);
        ctx.lineTo(particles[j].x, particles[j].y);
```

```
        ctx.stroke();
      }
    }
  }

  particles.forEach(p => {
    p.x += p.vx;
    p.y += p.vy;
    if (p.x < 0 || p.x > canvas.width) p.vx *= -1;
    if (p.y < 0 || p.y > canvas.height) p.vy *= -1;
  });

  requestAnimationFrame(draw);
}

draw();

window.addEventListener('resize', () => {
  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;
```

```
    });
  }

  pdfjsLib.GlobalWorkerOptions.workerSrc =
'https://cdnjs.cloudflare.com/ajax/libs/pdf.js/3.11.174/pdf.worker.
min.js';
 </script>
</body>
</html>
```

view-sammary.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8" />
 <meta name="viewport" content="width=device-width, initial-
scale=1" />
 <title>View Note Summary</title>
 <link
href="https://fonts.googleapis.com/css2?family=Montserrat:wght
@400;700&display=swap" rel="stylesheet">
 <style>
```

```
  * {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body {
  font-family: 'Montserrat', sans-serif;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background: #0d1b2a;
  color: #e0e6f8;
  height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  opacity: 0;
  transition: opacity 0.8s ease-in;
  position: relative;
```

```
}

body.loaded {
  opacity: 1;
}

#ai-bg {
  position: fixed;
  top: 0;
  left: 0;
  z-index: 0;
  width: 100%;
  height: 100%;
  background: radial-gradient(ellipse at bottom, #0f2027, #203a43, #2c5364);
}

.container {
  position: relative;
  background: rgba(255, 255, 255, 0.05);
```

```
  padding: 40px 35px;
  border-radius: 20px;
  box-shadow: 0 10px 30px rgba(0,0,0,0.5);
  text-align: center;
  max-width: 600px;
  width: 90%;
  backdrop-filter: blur(14px);
  z-index: 1;
  animation: slideIn 1.2s ease-out;
}

@keyframes slideIn {
  from { transform: translateY(40px); opacity: 0; }
  to { transform: translateY(0); opacity: 1; }
}

.project-title {
  font-size: 26px;
  font-weight: 700;
  margin-bottom: 20px;
```

```
  animation: glowTitle 2s ease-in-out infinite alternate;
  background: linear-gradient(90deg, #00c9ff, #92fe9d);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
}

@keyframes glowTitle {
  0%   { text-shadow: 0 0 5px #00ffff66; }
  100% { text-shadow: 0 0 20px #00ffffcc; }
}

h1 {
  font-size: 20px;
  margin-bottom: 15px;
}

input[type="text"] {
  width: 100%;
  padding: 14px 18px;
  border-radius: 10px;
```

```
  border: none;
  font-size: 16px;
  margin-bottom: 20px;
  outline: none;
  background: #334499;
  color: #d8e2ff;
}

button {
  background: linear-gradient(90deg, #00d8ff, #00ffab);
  border: none;
  padding: 14px 28px;
  font-weight: 700;
  font-size: 16px;
  color: #1a1a1a;
  border-radius: 12px;
  cursor: pointer;
  box-shadow: 0 5px 15px #00d8ffaa;
  transition: background 0.3s ease, transform 0.2s ease;
}
```

```
button:hover {
  transform: scale(1.05);
}

#summaryBox {
  margin-top: 30px;
  background: rgba(255,255,255,0.08);
  padding: 20px;
  border-radius: 10px;
  color: #d9e4ff;
  text-align: left;
  white-space: pre-wrap;
  font-size: 16px;
  line-height: 1.6;
  display: none;
  max-height: 400px;
  overflow-y: auto;
}
```

```
#error {
  margin-top: 15px;
  font-weight: 600;
  color: #ff6e6e;
}

#clearBtn {
  margin-top: 20px;
  background: #ff6e6e;
  color: white;
  display: none;
}

.pulse-button {
  margin-top: 15px;
  padding: 14px 28px;
  border: none;
  border-radius: 50px;
  background: linear-gradient(90deg, #00ffd5, #00aaff);
  color: #0d1b2a;
```

```
  font-weight: bold;
  font-size: 16px;
  cursor: pointer;
  box-shadow: 0 0 0 rgba(0, 255, 213, 0.7);
  animation: pulse 2s infinite;
  transition: transform 0.2s ease;
}

.pulse-button:hover {
  transform: scale(1.05);
}

@keyframes pulse {
  0% {
    box-shadow: 0 0 0 0 rgba(0, 255, 213, 0.4);
  }
  70% {
    box-shadow: 0 0 0 12px rgba(0, 255, 213, 0);
  }
  100% {
```

```
      box-shadow: 0 0 0 0 rgba(0, 255, 213, 0);
    }
  }
 </style>
</head>
<body>
 <canvas id="ai-bg"></canvas>

 <div class="container">
  <div class="project-title">🧠 AI Notes Summary Viewer</div>
  <h1>Get Note Summary</h1>
  <input type="text" id="noteIdInput" placeholder="Enter your Note ID..." />
  <button onclick="fetchSummary()">Get Summary</button>
  <div id="error"></div>
  <div id="summaryBox"></div>

  <div id="voiceBox" style="display:none; margin-top: 25px;">
   <button id="voiceBtn" class="pulse-button">🎙️ Speak Summary</button>
```

```
    <audio id="voiceAudio" preload="auto"></audio>
  </div>

  <button id="clearBtn" onclick="clearAndBack()">Clear & Back</button>
 </div>

 <script>
  const API_ENDPOINT = "https://9vr9ndoril.execute-api.us-east-1.amazonaws.com";
  console.log("Current API:", API_ENDPOINT);

  window.onload = () => {
   document.body.classList.add('loaded');
   animateAIBackground();
  };

 async function fetchSummary() {
  const noteId = document.getElementById('noteIdInput').value.trim();
```

```
    const summaryBox =
document.getElementById('summaryBox');
    const errorBox = document.getElementById('error');
    const clearBtn = document.getElementById('clearBtn');
    const voiceBox = document.getElementById('voiceBox');
    const voiceAudio = document.getElementById('voiceAudio');
    const voiceBtn = document.getElementById('voiceBtn');

    summaryBox.style.display = 'none';
    summaryBox.textContent = '';
    errorBox.textContent = '';
    clearBtn.style.display = 'none';
    voiceBox.style.display = 'none';
    voiceAudio.src = '';

    if (!noteId) {
      errorBox.textContent = 'Please enter a valid Note ID.';
      return;
    }
```

```
    try {
      summaryBox.textContent = ' ⏳ Fetching summary...';
      summaryBox.style.display = 'block';

      const response = await fetch(`${API_ENDPOINT}/get-summary?noteId=${noteId}`);
      const raw = await response.json();

      // ✅ FIX: Properly parse if raw.body is a JSON string
      const result = typeof raw.body === "string" ? JSON.parse(raw.body) : raw;

      if (response.ok) {
        summaryBox.textContent = result.summary || 'No summary found.';
        summaryBox.style.display = 'block';
        clearBtn.style.display = 'inline-block';

        // ✅ SHOW 🎙️ Button if audio_url exists
        if (result.audio_url) {
          voiceAudio.src = result.audio_url;
```

```
    voiceBox.style.display = 'block';


    voiceBtn.textContent = ' 🎙️ Speak Summary';
    voiceBtn.onclick = () => {
      if (voiceAudio.paused) {
        voiceAudio.play();
        voiceBtn.textContent = ' ⏸️ Pause Summary';
      } else {
        voiceAudio.pause();
        voiceBtn.textContent = ' 🎙️ Speak Summary';
      }
    };

    voiceAudio.onended = () => {
      voiceBtn.textContent = ' 🎙️ Speak Summary';
    };
  }
} else {
  summaryBox.style.display = 'none';
```

```
      errorBox.textContent = result.error || 'Summary not found.';
    }
  } catch (err) {
    console.error("Error fetching summary:", err);
    errorBox.textContent = ' ⚠️ Error while fetching summary.';
  }
}

  function clearAndBack() {
    document.body.style.opacity = '0';
    setTimeout(() => {
      window.location.href = 'upload.html';
    }, 500);
  }

  function animateAIBackground() {
    const canvas = document.getElementById('ai-bg');
    const ctx = canvas.getContext('2d');
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
```

```
let particles = [];
const total = 100;
for (let i = 0; i < total; i++) {
  particles.push({
    x: Math.random() * canvas.width,
    y: Math.random() * canvas.height,
    vx: (Math.random() - 0.5) * 0.6,
    vy: (Math.random() - 0.5) * 0.6,
    radius: Math.random() * 2 + 1
  });
}

function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.fillStyle = '#00ffff';
  particles.forEach(p => {
    ctx.beginPath();
    ctx.arc(p.x, p.y, p.radius, 0, Math.PI * 2);
    ctx.fill();
```

```
});

for (let i = 0; i < total; i++) {
  for (let j = i + 1; j < total; j++) {
    let dx = particles[i].x - particles[j].x;
    let dy = particles[i].y - particles[j].y;
    let dist = Math.sqrt(dx * dx + dy * dy);
    if (dist < 100) {
      ctx.beginPath();
      ctx.strokeStyle = `rgba(0,255,255,${1 - dist / 100})`;
      ctx.moveTo(particles[i].x, particles[i].y);
      ctx.lineTo(particles[j].x, particles[j].y);
      ctx.stroke();
    }
  }
}

particles.forEach(p => {
  p.x += p.vx;
  p.y += p.vy;
```

```
      if (p.x < 0 || p.x > canvas.width) p.vx *= -1;
      if (p.y < 0 || p.y > canvas.height) p.vy *= -1;
    });

    requestAnimationFrame(draw);
  }

  draw();
  window.addEventListener('resize', () => {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
  });
}
</script>
</body>
</html>
```
