```
import {
  BedrockAgentRuntimeClient,
  RetrieveCommand,
  RetrieveCommandInput,
} from "@aws-sdk/client-bedrock-agent-runtime";
import { type ClassValue, clsx } from "clsx";
import { twMerge } from "tailwind-merge";

console.log("ð\237\224\221 Have AWS AccessKey?", !!process.env.BAWS_ACCESS_KEY_ID);
console.log("ð\237\224\221 Have AWS Secret?", !!process.env.BAWS_SECRET_ACCESS_KEY);

const bedrockClient = new BedrockAgentRuntimeClient({
  region: "us-east-1", // Make sure this matches your Bedrock region
  credentials: {
    accessKeyId: process.env.BAWS_ACCESS_KEY_ID!,
    secretAccessKey: process.env.BAWS_SECRET_ACCESS_KEY!,
  },
});

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}

export interface RAGSource {
  id: string;
  fileName: string;
  snippet: string;
  score: number;
}

export async function retrieveContext(
  query: string,
  knowledgeBaseId: string,
  n: number = 3,
): Promise<{
  context: string;
  isRagWorking: boolean;
  ragSources: RAGSource[];
}> {
  try {
    if (!knowledgeBaseId) {
      console.error("knowledgeBaseId is not provided");
      return {
        context: "",
        isRagWorking: false,
        ragSources: [],
      };
    }

    const input: RetrieveCommandInput = {
      knowledgeBaseId: knowledgeBaseId,
      retrievalQuery: { text: query },
      retrievalConfiguration: {
        vectorSearchConfiguration: { numberOfResults: n },
      },
    };

    const command = new RetrieveCommand(input);
    const response = await bedrockClient.send(command);

    // Parse results
    const rawResults = response?.retrievalResults || [];
    const ragSources: RAGSource[] = rawResults
      .filter((res: any) => res.content && res.content.text)
      .map((result: any, index: number) => {
        const uri = result?.location?.s3Location?.uri || "";
        const fileName = uri.split("/").pop() || `Source-${index}.txt`;

        return {
          id:
            result.metadata?.["x-amz-bedrock-kb-chunk-id"] || `chunk-${index}`,
          fileName: fileName.replace(/_/g, " ").replace(".txt", ""),
          snippet: result.content?.text || "",
          score: result.score || 0,
        };
      })
      .slice(0, 1);

    console.log("ð\237\224\215 Parsed RAG Sources:", ragSources); // Debug log

    const context = rawResults
      .filter((res: any) => res.content && res.content.text)
      .map((res: any) => res.content.text)
      .join("\n\n");

    return {
      context,
      isRagWorking: true,
      ragSources,
    };
  } catch (error) {
    console.error("RAG Error:", error);
    return { context: "", isRagWorking: false, ragSources: [] };
  }
}
```

```
import Anthropic from "@anthropic-ai/sdk";
import { z } from "zod";
import { retrieveContext, RAGSource } from "@/app/lib/utils";
import crypto from "crypto";
import customerSupportCategories from "@/app/lib/customer_support_categories.json";

const anthropic = new Anthropic({
  apiKey: process.env.ANTHROPIC_API_KEY,
});

// Debug message helper function
// Input: message string and optional data object
// Output: JSON string with message, sanitized data, and timestamp
const debugMessage = (msg: string, data: any = {}) => {
  console.log(msg, data);
  const timestamp = new Date().toISOString().replace(/[^\x20-\x7E]/g, "");
  const safeData = JSON.parse(JSON.stringify(data));
  return JSON.stringify({ msg, data: safeData, timestamp });
};

// Define the schema for the AI response using Zod
// This ensures type safety and validation for the AI’s output
const responseSchema = z.object({
  response: z.string(),
  thinking: z.string(),
  user_mood: z.enum([
    "positive",
    "neutral",
    "negative",
    "curious",
    "frustrated",
    "confused",
  ]),
  suggested_questions: z.array(z.string()),
  debug: z.object({
    context_used: z.boolean(),
  }),
  matched_categories: z.array(z.string()).optional(),
  redirect_to_agent: z
    .object({
      should_redirect: z.boolean(),
      reason: z.string().optional(),
    })
    .optional(),
});

// Helper function to sanitize header values
// Input: string value
// Output: sanitized string (ASCII characters only)
function sanitizeHeaderValue(value: string): string {
  return value.replace(/[^\x00-\x7F]/g, "");
}

// Helper function to log timestamps for performance measurement
// Input: label string and start time
// Output: Logs the duration for the labeled operation
const logTimestamp = (label: string, start: number) => {
  const timestamp = new Date().toISOString();
  const time = ((performance.now() - start) / 1000).toFixed(2);
  console.log(`â\217±ï¸\217 [${timestamp}] ${label}: ${time}s`);
};

// Main POST request handler
export async function POST(req: Request) {
  const apiStart = performance.now();
  const measureTime = (label: string) => logTimestamp(label, apiStart);

  // Extract data from the request body
  const { messages, model, knowledgeBaseId } = await req.json();
  const latestMessage = messages[messages.length - 1].content;

  console.log("ð\237\223\235 Latest Query:", latestMessage);
  measureTime("User Input Received");

  // Prepare debug data
  const MAX_DEBUG_LENGTH = 1000;
  const debugData = sanitizeHeaderValue(
    debugMessage("ð\237\232\200 API route called", {
      messagesReceived: messages.length,
      latestMessageLength: latestMessage.length,
      anthropicKeySlice: process.env.ANTHROPIC_API_KEY?.slice(0, 4) + "****",
    }),
  ).slice(0, MAX_DEBUG_LENGTH);

  // Initialize variables for RAG retrieval
  let retrievedContext = "";
  let isRagWorking = false;
  let ragSources: RAGSource[] = [];

  // Attempt to retrieve context from RAG
  try {
    console.log("ð\237\224\215 Initiating RAG retrieval for query:", latestMessage);
    measureTime("RAG Start");
    const result = await retrieveContext(latestMessage, knowledgeBaseId);
    retrievedContext = result.context;
    isRagWorking = result.isRagWorking;
    ragSources = result.ragSources || [];

    if (!result.isRagWorking) {
      console.warn("ð\237\232¨ RAG Retrieval failed but did not throw!");
    }

    measureTime("RAG Complete");
    console.log("ð\237\224\215 RAG Retrieved:", isRagWorking ? "YES" : "NO");
    console.log(
      "â\234\205 RAG retrieval completed successfully. Context:",
      retrievedContext.slice(0, 100) + "...",
    );
  } catch (error) {
    console.error("ð\237\222\200 RAG Error:", error);
    console.error("â\235\214 RAG retrieval failed for query:", latestMessage);
    retrievedContext = "";
    isRagWorking = false;
    ragSources = [];
  }

  measureTime("RAG Total Duration");

  // Prepare categories context for the system prompt
  const USE_CATEGORIES = true;
  const categoryListString = customerSupportCategories.categories
    .map((c) => c.id)
    .join(", ");

  const categoriesContext = USE_CATEGORIES
    ? `
    To help with our internal classification of inquiries, we would like you to categorize inquiries in additio
n to answering them. We have provided you with ${customerSupportCategories.categories.length} customer support
categories.
    Check if your response fits into any category and include the category IDs in your "matched_categories" arr
ay.
    The available categories are: ${categoryListString}
    If multiple categories match, include multiple category IDs. If no categories match, return an empty array.
  `
    : "";

  // Change the system prompt company for your use case
  const systemPrompt = `You are acting as an Anthropic customer support assistant chatbot inside a chat window
on a website. You are chatting with a human user who is asking for help about Anthropic's products and services
. When responding to the user, aim to provide concise and helpful responses while maintaining a polite and prof
essional tone.

  To help you answer the user's question, we have retrieved the following information for you. It may or may no
t be relevant (we are using a RAG pipeline to retrieve this information):
  ${isRagWorking ? `${retrievedContext}` : "No information found for this query."}

  Please provide responses that only use the information you have been given. If no information is available or
 if the information is not relevant for answering the question, you can redirect the user to a human agent for
further assistance.

  ${categoriesContext}

  If the question is unrelated to Anthropic's products and services, you should redirect the user to a human ag
ent.

  You are the first point of contact for the user and should try to resolve their issue or provide relevant inf
ormation. If you are unable to help the user or if the user explicitly asks to talk to a human, you can redirec
t them to a human agent for further assistance.

  To display your responses correctly, you must format your entire response as a valid JSON object with the fol
lowing structure:
  {
      "thinking": "Brief explanation of your reasoning for how you should address the user's query",
      "response": "Your concise response to the user",
      "user_mood": "positive|neutral|negative|curious|frustrated|confused",
      "suggested_questions": ["Question 1?", "Question 2?", "Question 3?"],
      "debug": {
        "context_used": true|false
      },
      ${USE_CATEGORIES ? '"matched_categories": ["category_id1", "category_id2"],' : ""}
      "redirect_to_agent": {
        "should_redirect": boolean,
        "reason": "Reason for redirection (optional, include only if should_redirect is true)"
      }
    }

  Here are a few examples of how your response should look like:

  Example of a response without redirection to a human agent:
  {
    "thinking": "Providing relevant information from the knowledge base",
    "response": "Here's the information you requested...",
    "user_mood": "curious",
    "suggested_questions": ["How do I update my account?", "What are the payment options?"],
    "debug": {
      "context_used": true
    },
    "matched_categories": ["account_management", "billing"],
    "redirect_to_agent": {
      "should_redirect": false
    }
  }

  Example of a response with redirection to a human agent:
  {
    "thinking": "User request requires human intervention",
    "response": "I understand this is a complex issue. Let me connect you with a human agent who can assist you
better.",
    "user_mood": "frustrated",
    "suggested_questions": [],
    "debug": {
      "context_used": false
    },
    "matched_categories": ["technical_support"],
    "redirect_to_agent": {
      "should_redirect": true,
      "reason": "Complex technical issue requiring human expertise"
    }
  }
  `

  function sanitizeAndParseJSON(jsonString : string) {
    // Replace newlines within string values
    const sanitized = jsonString.replace(/(?<=:\s*")(.|\n)*?(?=")/g, match =>
      match.replace(/\n/g, "\\n")
    );

    try {
      return JSON.parse(sanitized);
    } catch (parseError) {
      console.error("Error parsing JSON response:", parseError);
      throw new Error("Invalid JSON response from AI");
    }
  }

  try {
    console.log(`ð\237\232\200 Query Processing`);
    measureTime("Claude Generation Start");

    const anthropicMessages = messages.map((msg: any) => ({
      role: msg.role,
      content: msg.content,
    }));

    anthropicMessages.push({
      role: "assistant",
      content: "{",
    });

    const response = await anthropic.messages.create({
      model: model,
      max_tokens: 1000,
      messages: anthropicMessages,
      system: systemPrompt,
      temperature: 0.3,
    });

    measureTime("Claude Generation Complete");
    console.log("â\234\205 Message generation completed");

    // Extract text content from the response
    const textContent = "{" + response.content
      .filter((block): block is Anthropic.TextBlock => block.type === "text")
      .map((block) => block.text)
      .join(" ");

    // Parse the JSON response
    let parsedResponse;
    try {
      parsedResponse = sanitizeAndParseJSON(textContent);
    } catch (parseError) {
      console.error("Error parsing JSON response:", parseError);
      throw new Error("Invalid JSON response from AI");
    }

    const validatedResponse = responseSchema.parse(parsedResponse);

    const responseWithId = {
      id: crypto.randomUUID(),
      ...validatedResponse,
    };

    // Check if redirection to a human agent is needed
    if (responseWithId.redirect_to_agent?.should_redirect) {
      console.log("ð\237\232¨ AGENT REDIRECT TRIGGERED!");
      console.log("Reason:", responseWithId.redirect_to_agent.reason);
    }

    // Prepare the response object
    const apiResponse = new Response(JSON.stringify(responseWithId), {
      status: 200,
      headers: {
        "Content-Type": "application/json",
      },
    });

    // Add RAG sources to the response headers if available
    if (ragSources.length > 0) {
      apiResponse.headers.set(
        "x-rag-sources",
        sanitizeHeaderValue(JSON.stringify(ragSources)),
      );
    }

    // Add debug data to the response headers
    apiResponse.headers.set("X-Debug-Data", sanitizeHeaderValue(debugData));

    measureTime("API Complete");

    return apiResponse;
  } catch (error) {
    // Handle errors in AI response generation
    console.error("ð\237\222¥ Error in message generation:", error);
    const errorResponse = {
      response:
        "Sorry, there was an issue processing your request. Please try again later.",
      thinking: "Error occurred during message generation.",
      user_mood: "neutral",
      debug: { context_used: false },
    };
    return new Response(JSON.stringify(errorResponse), {
      status: 500,
      headers: { "Content-Type": "application/json" },
    });
  }
}
```

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
    :root {
        --background: 0 0% 100%;
        --foreground: 0 0% 3.9%;
        --card: 0 0% 100%;
        --card-foreground: 0 0% 3.9%;
        --popover: 0 0% 100%;
        --popover-foreground: 0 0% 3.9%;
        --primary: 0 0% 9%;
        --primary-foreground: 0 0% 98%;
        --secondary: 0 0% 96.1%;
        --secondary-foreground: 0 0% 9%;
        --muted: 0 0% 96.1%;
        --muted-foreground: 0 0% 45.1%;
        --accent: 0 0% 96.1%;
        --accent-foreground: 0 0% 9%;
        --destructive: 0 84.2% 60.2%;
        --destructive-foreground: 0 0% 98%;
        --border: 0 0% 89.8%;
        --input: 0 0% 89.8%;
        --ring: 0 0% 3.9%;
        --radius: 0.75rem;
        --chart-1: 12 76% 61%;
        --chart-2: 173 58% 39%;
        --chart-3: 197 37% 24%;
        --chart-4: 43 74% 66%;
        --chart-5: 27 87% 67%;
    }

    .dark {
        --background: 0 0% 3.9%;
        --foreground: 0 0% 98%;
        --card: 0 0% 3.9%;
        --card-foreground: 0 0% 98%;
        --popover: 0 0% 3.9%;
        --popover-foreground: 0 0% 98%;
        --primary: 0 0% 98%;
        --primary-foreground: 0 0% 9%;
        --secondary: 0 0% 14.9%;
        --secondary-foreground: 0 0% 98%;
        --muted: 0 0% 14.9%;
        --muted-foreground: 0 0% 63.9%;
        --accent: 0 0% 14.9%;
        --accent-foreground: 0 0% 98%;
        --destructive: 0 62.8% 30.6%;
        --destructive-foreground: 0 0% 98%;
        --border: 0 0% 14.9%;
        --input: 0 0% 14.9%;
        --ring: 0 0% 83.1%;
        --chart-1: 220 70% 50%;
        --chart-2: 160 60% 45%;
        --chart-3: 30 80% 55%;
        --chart-4: 280 65% 60%;
        --chart-5: 340 75% 55%;
    }
}

@layer base {
    * {
        @apply border-border;
    }
    body {
        @apply bg-background text-foreground;
    }
}
```

```bash
#!/bin/bash

# Function to print usage
print_usage() {
    echo "Usage: $0 <output_pdf> <directory1> [<directory2> ...]"
    echo "Example: $0 all_code.pdf /path/to/project1 /path/to/project2"
}

# Check if at least two arguments are provided
if [ $# -lt 2 ]; then
    print_usage
    exit 1
fi

# Output PDF file name
OUTPUT="$1"
shift

# Temporary PS file
TEMP_PS="temp.ps"

# Array of file extensions to include
EXTENSIONS=("c" "cpp" "h" "hpp" "py" "java" "js" "html" "css" "php" "rb" "go" "rs" "swift" "kt" "scala" "pl" "s
h" "sql" "js" "ts")

# Function to get the appropriate language for enscript
get_language() {
    case "$1" in
        c|h|cpp|hpp) echo "c" ;;
        py) echo "python" ;;
        java) echo "java" ;;
        js|ts) echo "javascript" ;;
        html|css) echo "html" ;;
        php) echo "php" ;;
        rb) echo "ruby" ;;
        go) echo "go" ;;
        rs) echo "rust" ;;
        swift) echo "swift" ;;
        kt) echo "kotlin" ;;
        scala) echo "scala" ;;
        pl) echo "perl" ;;
        sh) echo "bash" ;;
        sql) echo "sql" ;;
        json) echo "javascript" ;;
        *) echo "text" ;;
    esac
}

# Function to generate the find command for a single directory
generate_find_command() {
    local dir="$1"
    local cmd="find \"$dir\" \( -type d \( -name node_modules -o -name .next \) -prune \) -o \( -type f \("
    for i in "${!EXTENSIONS[@]}"; do
        if [ $i -ne 0 ]; then
            cmd+=" -o"
        fi
        cmd+=" -name \"*.${EXTENSIONS[$i]}\""
    done
    cmd+=" \) -print \)"
    echo "$cmd"
}

# Clear the temporary PS file if it exists
> "$TEMP_PS"

# Process each directory
for dir in "$@"; do
    if [ ! -d "$dir" ]; then
        echo "Warning: $dir is not a valid directory. Skipping."
        continue
    fi

    echo "Processing directory: $dir"

    # Generate and execute the find command
    eval "$(generate_find_command "$dir")" | while read -r file; do
        echo "Converting $file"
        # Get the file extension
        ext="${file##*.}"
        # Get the appropriate language for enscript
        lang=$(get_language "$ext")
        # Use the appropriate syntax highlighting based on the file extension
        enscript -p - --highlight="$lang" --color=1 -fCourier8 --header="$file|Page \$% of \$=" "$file" >> "$TE
MP_PS"
    done
done
```

```
# Check if any files were processed
if [ ! -s "$TEMP_PS" ]; then
    echo "Error: No files were found or processed. The output PDF will not be created."
    rm "$TEMP_PS"
    exit 1
fi

# Convert PostScript to PDF
ps2pdf "$TEMP_PS" "$OUTPUT"

# Remove temporary PostScript file
rm "$TEMP_PS"

echo "Conversion complete. Output saved as $OUTPUT"
```

```
/// <reference types="next" />
/// <reference types="next/image-types/global" />

// NOTE: This file should not be edited
// see https://nextjs.org/docs/basic-features/typescript for more information.
```

```
import type { Config } from "tailwindcss";

const config = {
  darkMode: ["class"],
  content: [
    "./pages/**/*.{ts,tsx}",
    "./components/**/*.{ts,tsx}",
    "./app/**/*.{ts,tsx}",
    "./src/**/*.{ts,tsx}",
  ],
  prefix: "",
  theme: {
    container: {
      center: true,
      padding: "2rem",
      screens: {
        "2xl": "1400px",
      },
    },
    extend: {
      colors: {
        border: "hsl(var(--border))",
        input: "hsl(var(--input))",
        ring: "hsl(var(--ring))",
        background: "hsl(var(--background))",
        foreground: "hsl(var(--foreground))",
        primary: {
          DEFAULT: "hsl(var(--primary))",
          foreground: "hsl(var(--primary-foreground))",
        },
        secondary: {
          DEFAULT: "hsl(var(--secondary))",
          foreground: "hsl(var(--secondary-foreground))",
        },
        destructive: {
          DEFAULT: "hsl(var(--destructive))",
          foreground: "hsl(var(--destructive-foreground))",
        },
        muted: {
          DEFAULT: "hsl(var(--muted))",
          foreground: "hsl(var(--muted-foreground))",
        },
        accent: {
          DEFAULT: "hsl(var(--accent))",
          foreground: "hsl(var(--accent-foreground))",
        },
        popover: {
          DEFAULT: "hsl(var(--popover))",
          foreground: "hsl(var(--popover-foreground))",
        },
        card: {
          DEFAULT: "hsl(var(--card))",
          foreground: "hsl(var(--card-foreground))",
        },
      },
      borderRadius: {
        lg: "var(--radius)",
        md: "calc(var(--radius) - 2px)",
        sm: "calc(var(--radius) - 4px)",
      },
      keyframes: {
        "accordion-down": {
          from: { height: "0" },
          to: { height: "var(--radix-accordion-content-height)" },
        },
        "accordion-up": {
          from: { height: "var(--radix-accordion-content-height)" },
          to: { height: "0" },
        },
        fadeIn: {
          "0%": { opacity: "0" },
          "100%": { opacity: "1" },
        },
        fadeInUp: {
          "0%": { opacity: "0", transform: "translateY(8px)" },
          "100%": { opacity: "1", transform: "translateY(0)" },
        },
        shimmer: {
          "100%": { transform: "translateX(100%)" },
        },
      },
      animation: {
        "accordion-down": "accordion-down 0.2s ease-out",
        "accordion-up": "accordion-up 0.2s ease-out",
        "fade-in": "fadeIn 0.5s ease-out forwards",
        "fade-in-up": "fadeInUp 0.5s ease-out forwards",
        "fade-in-up-fast": "fadeInUp 0.3s ease-out forwards",
        "fade-in-up-slow":
          "fadeInUp 0.8s cubic-bezier(0.4, 0, 0.2, 1) forwards",
        shimmer: "shimmer 2s linear infinite",
      },
    },
  },
  plugins: [require("tailwindcss-animate")],
} satisfies Config;

export default config;
```

```
// themes.ts

export const themes = {
  neutral: {
    light: {
      background: "0 0% 100%",
      foreground: "0 0% 3.9%",
      card: "0 0% 100%",
      "card-foreground": "0 0% 3.9%",
      popover: "0 0% 100%",
      "popover-foreground": "0 0% 3.9%",
      primary: "0 0% 9%",
      "primary-foreground": "0 0% 98%",
      secondary: "0 0% 96.1%",
      "secondary-foreground": "0 0% 9%",
      muted: "0 0% 96.1%",
      "muted-foreground": "0 0% 45.1%",
      accent: "0 0% 96.1%",
      "accent-foreground": "0 0% 9%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "0 0% 98%",
      border: "0 0% 89.8%",
      input: "0 0% 89.8%",
      ring: "0 0% 3.9%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "0 0% 3.9%",
      foreground: "0 0% 98%",
      card: "0 0% 3.9%",
      "card-foreground": "0 0% 98%",
      popover: "0 0% 3.9%",
      "popover-foreground": "0 0% 98%",
      primary: "0 0% 98%",
      "primary-foreground": "0 0% 9%",
      secondary: "0 0% 14.9%",
      "secondary-foreground": "0 0% 98%",
      muted: "0 0% 14.9%",
      "muted-foreground": "0 0% 63.9%",
      accent: "0 0% 14.9%",
      "accent-foreground": "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "0 0% 98%",
      border: "0 0% 14.9%",
      input: "0 0% 14.9%",
      ring: "0 0% 83.1%",
      radius: "0.75rem",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  red: {
    light: {
      background: "0 0% 100%",
      foreground: "0 0% 3.9%",
      card: "0 0% 100%",
      "card-foreground": "0 0% 3.9%",
      popover: "0 0% 100%",
      "popover-foreground": "0 0% 3.9%",
      primary: "0 72.2% 50.6%",
      "primary-foreground": "0 85.7% 97.3%",
      secondary: "0 0% 96.1%",
      "secondary-foreground": "0 0% 9%",
      muted: "0 0% 96.1%",
      "muted-foreground": "0 0% 45.1%",
      accent: "0 0% 96.1%",
      "accent-foreground": "0 0% 9%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "0 0% 98%",
      border: "0 0% 89.8%",
      input: "0 0% 89.8%",
      ring: "0 72.2% 50.6%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "0 0% 3.9%",
      foreground: "0 0% 98%",
      card: "0 0% 3.9%",
      "card-foreground": "0 0% 98%",
      popover: "0 0% 3.9%",
      "popover-foreground": "0 0% 98%",
      primary: "0 72.2% 50.6%",
      "primary-foreground": "0 85.7% 97.3%",
      secondary: "0 0% 14.9%",
      "secondary-foreground": "0 0% 98%",
      muted: "0 0% 14.9%",
      "muted-foreground": "0 0% 63.9%",
      accent: "0 0% 14.9%",
      "accent-foreground": "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "0 0% 98%",
      border: "0 0% 14.9%",
      input: "0 0% 14.9%",
      ring: "0 72.2% 50.6%",
      radius: "0.75rem",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  violet: {
    light: {
      background: "0 0% 100%",
      foreground: "224 71.4% 4.1%",
      card: "0 0% 100%",
      "card-foreground": "224 71.4% 4.1%",
      popover: "0 0% 100%",
      "popover-foreground": "224 71.4% 4.1%",
      primary: "262.1 83.3% 57.8%",
      "primary-foreground": "210 20% 98%",
      secondary: "220 14.3% 95.9%",
      "secondary-foreground": "220.9 39.3% 11%",
      muted: "220 14.3% 95.9%",
      "muted-foreground": "220 8.9% 46.1%",
      accent: "220 14.3% 95.9%",
      "accent-foreground": "220.9 39.3% 11%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "210 20% 98%",
      border: "220 13% 91%",
      input: "220 13% 91%",
      ring: "262.1 83.3% 57.8%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "224 71.4% 4.1%",
      foreground: "210 20% 98%",
      card: "224 71.4% 4.1%",
      "card-foreground": "210 20% 98%",
      popover: "224 71.4% 4.1%",
      "popover-foreground": "210 20% 98%",
      primary: "263.4 70% 50.4%",
      "primary-foreground": "210 20% 98%",
      secondary: "215 27.9% 16.9%",
      "secondary-foreground": "210 20% 98%",
      muted: "215 27.9% 16.9%",
      "muted-foreground": "217.9 10.6% 64.9%",
      accent: "215 27.9% 16.9%",
      "accent-foreground": "210 20% 98%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "210 20% 98%",
      border: "215 27.9% 16.9%",
      input: "215 27.9% 16.9%",
      ring: "263.4 70% 50.4%",
      radius: "0.75rem",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  blue: {
    light: {
      background: "0 0% 100%",
      foreground: "222.2 84% 4.9%",
      card: "0 0% 100%",
      "card-foreground": "222.2 84% 4.9%",
      popover: "0 0% 100%",
      "popover-foreground": "222.2 84% 4.9%",
      primary: "221.2 83.2% 53.3%",
      "primary-foreground": "210 40% 98%",
      secondary: "210 40% 96.1%",
      "secondary-foreground": "222.2 47.4% 11.2%",
      muted: "210 40% 96.1%",
      "muted-foreground": "215.4 16.3% 46.9%",
      accent: "210 40% 96.1%",
      "accent-foreground": "222.2 47.4% 11.2%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "210 40% 98%",
      border: "214.3 31.8% 91.4%",
      input: "214.3 31.8% 91.4%",
      ring: "221.2 83.2% 53.3%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "222.2 84% 4.9%",
      foreground: "210 40% 98%",
      card: "222.2 84% 4.9%",
      "card-foreground": "210 40% 98%",
      popover: "222.2 84% 4.9%",
      "popover-foreground": "210 40% 98%",
      primary: "217.2 91.2% 59.8%",
      "primary-foreground": "222.2 47.4% 11.2%",
      secondary: "217.2 32.6% 17.5%",
      "secondary-foreground": "210 40% 98%",
      muted: "217.2 32.6% 17.5%",
      "muted-foreground": "215 20.2% 65.1%",
      accent: "217.2 32.6% 17.5%",
      "accent-foreground": "210 40% 98%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "210 40% 98%",
      border: "217.2 32.6% 17.5%",
      input: "217.2 32.6% 17.5%",
      ring: "224.3 76.3% 48%",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  tangerine: {
    light: {
      background: "0 0% 100%",
      foreground: "20 14.3% 4.1%",
      card: "0 0% 100%",
      "card-foreground": "20 14.3% 4.1%",
      popover: "0 0% 100%",
      "popover-foreground": "20 14.3% 4.1%",
      primary: "24.6 95% 53.1%",
      "primary-foreground": "60 9.1% 97.8%",
      secondary: "60 4.8% 95.9%",
      "secondary-foreground": "24 9.8% 10%",
      muted: "60 4.8% 95.9%",
      "muted-foreground": "25 5.3% 44.7%",
      accent: "60 4.8% 95.9%",
      "accent-foreground": "24 9.8% 10%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "60 9.1% 97.8%",
      border: "20 5.9% 90%",
      input: "20 5.9% 90%",
      ring: "24.6 95% 53.1%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "60 9.1% 97.8%",
      card: "20 14.3% 4.1%",
      "card-foreground": "60 9.1% 97.8%",
      popover: "20 14.3% 4.1%",
      "popover-foreground": "60 9.1% 97.8%",
      primary: "20.5 90.2% 48.2%",
      "primary-foreground": "60 9.1% 97.8%",
      secondary: "12 6.5% 15.1%",
      "secondary-foreground": "60 9.1% 97.8%",
      muted: "12 6.5% 15.1%",
      "muted-foreground": "24 5.4% 63.9%",
      accent: "12 6.5% 15.1%",
      "accent-foreground": "60 9.1% 97.8%",
      destructive: "0 72.2% 50.6%",
      "destructive-foreground": "60 9.1% 97.8%",
      border: "12 6.5% 15.1%",
      input: "12 6.5% 15.1%",
      ring: "20.5 90.2% 48.2%",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  emerald: {
    light: {
      background: "0 0% 100%",
      foreground: "240 10% 3.9%",
      card: "0 0% 100%",
      "card-foreground": "240 10% 3.9%",
      popover: "0 0% 100%",
      "popover-foreground": "240 10% 3.9%",
      primary: "142.1 76.2% 36.3%",
      "primary-foreground": "355.7 100% 97.3%",
      secondary: "240 4.8% 95.9%",
      "secondary-foreground": "240 5.9% 10%",
      muted: "240 4.8% 95.9%",
      "muted-foreground": "240 3.8% 46.1%",
      accent: "240 4.8% 95.9%",
      "accent-foreground": "240 5.9% 10%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "0 0% 98%",
      border: "240 5.9% 90%",
      input: "240 5.9% 90%",
      ring: "142.1 76.2% 36.3%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "0 0% 95%",
      card: "24 9.8% 10%",
      "card-foreground": "0 0% 95%",
      popover: "0 0% 9%",
      "popover-foreground": "0 0% 95%",
      primary: "142.1 70.6% 45.3%",
      "primary-foreground": "144.9 80.4% 10%",
      secondary: "240 3.7% 15.9%",
      "secondary-foreground": "0 0% 98%",
      muted: "0 0% 15%",
      "muted-foreground": "240 5% 64.9%",
      accent: "12 6.5% 15.1%",
      "accent-foreground": "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "0 85.7% 97.3%",
      border: "240 3.7% 15.9%",
      input: "240 3.7% 15.9%",
      ring: "142.4 71.8% 29.2%",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
  amber: {
    light: {
      background: "0 0% 100%",
      foreground: "20 14.3% 4.1%",
      card: "0 0% 100%",
      "card-foreground": "20 14.3% 4.1%",
      popover: "0 0% 100%",
      "popover-foreground": "20 14.3% 4.1%",
      primary: "47.9 95.8% 53.1%",
      "primary-foreground": "26 83.3% 14.1%",
      secondary: "60 4.8% 95.9%",
      "secondary-foreground": "24 9.8% 10%",
      muted: "60 4.8% 95.9%",
      "muted-foreground": "25 5.3% 44.7%",
      accent: "60 4.8% 95.9%",
      "accent-foreground": "24 9.8% 10%",
      destructive: "0 84.2% 60.2%",
      "destructive-foreground": "60 9.1% 97.8%",
      border: "20 5.9% 90%",
      input: "20 5.9% 90%",
      ring: "20 14.3% 4.1%",
      radius: "0.75rem",
      "chart-1": "12 76% 61%",
      "chart-2": "173 58% 39%",
      "chart-3": "197 37% 24%",
      "chart-4": "43 74% 66%",
      "chart-5": "27 87% 67%",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "60 9.1% 97.8%",
      card: "20 14.3% 4.1%",
      "card-foreground": "60 9.1% 97.8%",
      popover: "20 14.3% 4.1%",
      "popover-foreground": "60 9.1% 97.8%",
      primary: "47.9 95.8% 53.1%",
      "primary-foreground": "26 83.3% 14.1%",
      secondary: "12 6.5% 15.1%",
      "secondary-foreground": "60 9.1% 97.8%",
      muted: "12 6.5% 15.1%",
      "muted-foreground": "24 5.4% 63.9%",
      accent: "12 6.5% 15.1%",
      "accent-foreground": "60 9.1% 97.8%",
      destructive: "0 62.8% 30.6%",
      "destructive-foreground": "60 9.1% 97.8%",
      border: "12 6.5% 15.1%",
      input: "12 6.5% 15.1%",
      ring: "35.5 91.7% 32.9%",
      "chart-1": "220 70% 50%",
      "chart-2": "160 60% 45%",
      "chart-3": "30 80% 55%",
      "chart-4": "280 65% 60%",
      "chart-5": "340 75% 55%",
    },
  },
};
```

```ts
import { type ClassValue, clsx } from "clsx";
import { twMerge } from "tailwind-merge";

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}
```

/Users/abdulazizdot/Desktop/customer-support-agent/config.ts

```ts
type Config = {
  includeLeftSidebar: boolean;
  includeRightSidebar: boolean;
};

const config: Config = {
  includeLeftSidebar: process.env.NEXT_PUBLIC_INCLUDE_LEFT_SIDEBAR === "true",
  includeRightSidebar: process.env.NEXT_PUBLIC_INCLUDE_RIGHT_SIDEBAR === "true",
};

export default config;
```