Streamlining Ticket Assignment for Efficient Support Operations

Project Description:

The objective of this initiative is to implement an automated system for ticket routing at ABC Corporation, aimed at improving operational efficiency by accurately assigning support tickets to the appropriate teams. This solution aims to reduce delays in issue resolution, enhance customer satisfaction, and optimize resource utilization within the support department.

1.0 Executive Summary

Current customer support operations face a critical challenge in efficiently matching incoming tickets with the appropriate support agent. Reliance on legacy methods like simple load balancing or manual triage leads to significant operational friction, resulting in delayed resolutions, higher operational costs, and diminished customer satisfaction. This report details the necessity and strategy for adopting a **Streamlined Assignment Framework (SAF)**.

The SAF is a paradigm shift that integrates **Intelligent Skill-Based Routing (SBR)**, **Dynamic Capacity Management**, and **Artificial Intelligence (AI)** for precise ticket triage. By utilizing predictive analytics and real-time agent capacity data, the SAF ensures the "best-fit" agent handles the request from the outset.

The key outcomes projected from this transformation include:

- **Reduction in Average Handle Time (AHT):** Expected decrease of **18-25%** due to reduced transfers and immediate expertise engagement.
- **Increase in First Contact Resolution (FCR):** Projected improvement of **10-15%**, signaling more efficient use of agent time.
- **Significant Cost Savings:** Estimated **ROI of over 300%** within the first 18 months through labor efficiency and reduced ticket escalations.

The implementation roadmap focuses on three phases: data audit and skill definition, a targeted pilot program, and a phased rollout with continuous performance monitoring. Adopting the SAF is not merely an operational update; it is a strategic investment in scaling support operations while simultaneously elevating the quality of customer experience.

2.0 Introduction

2.1 Background and Problem Statement

In an increasingly competitive global market, customer support is often the primary differentiator for retaining customer loyalty. The sheer volume and complexity of support interactions continue to rise, making the efficiency of the service operation paramount. A fundamental bottleneck in nearly all high-volume support centers is the mechanism by which incoming customer inquiries (tickets) are assigned to human agents.

The prevailing models are fundamentally flawed: **Round-robin** ignores skill and complexity; **load-based (queue)** routing frequently assigns specialized

tickets to generalists; and **manual triage** is inherently slow, error-prone, and non-scalable. This misassignment leads to:

1. **Ticket Ping-Pong:** Excessive internal transfers between departments or agents.
2. **Increased Handling Time:** Agents spend time researching unfamiliar topics or waiting for subject matter experts (SMEs).
3. **Customer Frustration:** Repeated explanations and prolonged wait times damage brand perception.

The objective of this report is to move beyond reactive queue management and establish a **proactive, intelligent assignment framework** that treats every ticket as a unique problem requiring the precise, identified expertise.

2.2 Report Objectives and Scope

The primary objectives of this report are to:

- **Analyze** the deficiencies and quantify the business impact of current ticket assignment methodologies.
- **Design** a comprehensive, technology-enabled Streamlined Assignment Framework (SAF).
- **Detail** the required technology components, specifically the role of AI/ML in modern routing.
- **Propose** a practical, phased implementation roadmap, including change management strategies.
- **Calculate** the expected operational benefits and Return on Investment (ROI).

Scope: This analysis applies to all inbound support channels (voice, email, chat, social media) and focuses on the logic engine that connects the incoming request to the next available, best-suited agent.

3.0 Current State Analysis and Operational Deficits

3.1 Traditional Assignment Models:

Support centers commonly rely on one or a combination of the following basic assignment models:

Model	Mechanism	Key Deficit
Round-Robin	Distributes tickets sequentially to the next agent in the list.	**Ignores Skill:** High rate of misrouted specialized tickets.
Load-Based	Assigns to the agent with the fewest open tickets.	**Ignores Complexity:** Assigns a P1, complex ticket to an agent already handling five simple tickets.
Manual Triage	A dedicated dispatcher or supervisor reads and assigns tickets.	**Non-Scalable:** Becomes a critical bottleneck during peak volumes; reliant on human judgment.

These models prioritize **speed to assignment** over **speed to resolution**, which fundamentally undermines operational efficiency and customer experience.

3.2 Quantifying the Cost of Misassignment:

Misassignment is not merely an inconvenience; it is a measurable cost driver. The primary factors contributing to this cost include:

- **Increased Labor Cost:** Every transfer, follow-up, or escalation adds incremental labor minutes. If 25% of tickets are transferred once, and each transfer adds 5 minutes of review time, the labor waste is substantial.
- **SLA Penalties/Churn Risk:** Failure to meet Service Level Agreements (SLAs) due to delayed resolution leads to potential financial penalties or, worse, customer attrition.
- **Agent Attrition:** Agents consistently dealing with tickets outside their expertise experience burnout and frustration, leading to higher turnover and recruitment costs.

Observation: Internal data suggests tickets reassigned more than once take, on average, 45% longer to resolve than tickets assigned correctly the first time.

4.0 The Streamlined Assignment Framework (SAF)

The Streamlined Assignment Framework (SAF) is a multi-dimensional approach designed to optimize the **"best available agent"** decision by simultaneously considering three core principles: **Skill, Capacity, and Priority.**

4.1 Principle 1: Intelligent Skill-Based Routing (SBR):

SBR is the foundation of the SAF. It ensures that the required expertise is the primary driver of the assignment decision.

4.1.1 Defining the Agent Skill Matrix:

A comprehensive skill matrix must be established, encompassing the following dimensions:

- **Product/Service Expertise:** Specific modules, features, or product lines (e.g., "Billing," "Cloud Integration," "Mobile App").

- **Technical Proficiency Level:** Tier 1 (T1) basic support, T2 advanced troubleshooting, T3 engineering/SME support.
- **Language Proficiency:** Written and spoken fluency (e.g., French-T1, German-T2).
- **Channel Competency:** Expertise in specific communication types (e.g., "Live Chat Efficiency," "Phone De-escalation").

4.1.2 The Matching Logic: Weighted Scoring:

Instead of a simple Boolean match (Yes/No), the SBR uses a **weighted scoring algorithm**.

$$\text{Assignment Score} = (W_{Skill} \times S_{Match}) + (W_{Hist} \times H_{Success}) + \dots$$

- S_{Match}: Score based on how closely the agent's skills match the ticket requirements.
- $H_{Success}$: Historical resolution success rate for similar ticket types.
- W: Weighting factors (can be adjusted based on business priorities).

5.0 Enabling Technology: AI and Automation

The SAF cannot function efficiently without advanced technology, specifically the integration of Machine Learning (ML) models to automate triage and inform the routing decision.

5.1 Machine Learning for Ticket Triage and Categorization

Before a ticket hits the assignment engine, it must be accurately classified. AI/ML models replace manual tagging and basic keyword analysis.

5.1.1 Natural Language Processing (NLP)

NLP models analyze the customer's raw input (email body, chat transcript, voice transcript) to determine:

- **Intent:** Is the customer requesting a *refund*, reporting a *bug*, asking for a *feature*, or *billing* inquiry?
- **Sentiment:** Is the tone negative, frustrated, or neutral? This directly contributes to the Priority Score.

5.1.2 Automated Categorization and Tagging

The ML model automatically applies a refined, standardized set of categories and tags (e.g., `Product X: Cloud Integration: Connection Failure`). These tags are the essential input for the SBR engine to match the required agent skills. **Accuracy in this step is paramount**; a misclassified ticket is a misassigned ticket.

5.2 Predictive Routing and Resolution Forecasting

Predictive models add a layer of intelligence by forecasting outcomes *before* the ticket is assigned.

- **Resolution Time Prediction:** Based on the ticket's category, historical data, and the customer's historical profile, the ML model estimates the required Average Handle Time (AHT) for resolution. This informs the agent's **Dynamic Load Score** (Section 4.2).
- **First Contact Resolution (FCR) Forecasting:** The model predicts the probability of an FCR if assigned to Agent X versus Agent Y. The routing engine can then leverage this prediction to favor the agent with the highest probability of a swift, single-contact resolution for that specific ticket type.

PROGRAM:

```
import React, { useState, useEffect, useMemo, useCallback } from 'react';
import { initializeApp } from 'firebase/app';
import { getAuth, signInAnonymously, signInWithCustomToken, onAuthStateChanged }
from 'firebase/auth';
import { getFirestore, doc, getDoc, setDoc, updateDoc, onSnapshot, collection,
query, addDoc, writeBatch } from 'firebase/firestore';

// --- Global Context Variables (Mandatory for Canvas Environment) ---
// Note: These variables are provided by the embedding environment.
const appId = typeof __app_id !== 'undefined' ? __app_id : 'default-app-id';
const firebaseConfig = typeof __firebase_config !== 'undefined' ?
JSON.parse(__firebase_config) : null;
const initialAuthToken = typeof __initial_auth_token !== 'undefined' ?
__initial_auth_token : null;

// --- Firebase Initialization and Hooks ---
let db, auth;
if (firebaseConfig) {
  try {
    const app = initializeApp(firebaseConfig);
    db = getFirestore(app);
    auth = getAuth(app);
  } catch (error) {
    console.error("Firebase initialization failed:", error);
    // Continue without Firebase if initialization fails
    db = null;
    auth = null;
  }
}

// Global Types for clarity (in a real project, these would be in a separate
file)
/**
 * @typedef {Object} Agent
 * @property {string} id
 * @property {string} name
 * @property {string} team
 * @property {Object.<string, number>} skills - { skillName: level (1-5) }
 * @property {number} capacity - Max tickets
 * @property {number} currentLoad - Currently assigned tickets
 */

/**
 * @typedef {Object} Ticket
```

```
 * @property {string} id
 * @property {string} title
 * @property {string} category
 * @property {string} priority - P1, P2, P3
 * @property {string[]} requiredSkills
 * @property {string} status - New, Assigned, Resolved
 * @property {string} assignedAgentId - ID of the assigned agent
 * @property {string} [assignedAgentName] - Name of the assigned agent (for
display)
 */

/**
 * @typedef {Object} AppState
 * @property {string | null} userId
 * @property {boolean} isLoading
 * @property {Agent[]} agents
 * @property {Ticket[]} tickets
 */



const TEAM_COLORS = {
  'Frontend': 'bg-blue-100 text-blue-800',
  'Backend': 'bg-red-100 text-red-800',
  'Billing': 'bg-green-100 text-green-800',
  'Default': 'bg-gray-100 text-gray-800',
};

// Available Skills and Categories for Mock Data
const ALL_SKILLS = ['React', 'Python', 'Billing', 'SQL', 'Support'];
const ALL_CATEGORIES = ['Technical Bug', 'Billing Inquiry', 'Feature Request'];
const ALL_PRIORITIES = ['P1', 'P2', 'P3'];

// --- Utility Functions ---

// Function to calculate the routing score
const calculateScore = (ticket, agent) => {
  // 1. Skill Match Score
  let skillMatchScore = 0;
  ticket.requiredSkills.forEach(reqSkill => {
    const agentLevel = agent.skills[reqSkill] || 0;
    // Base score is agent level for the required skill (max 5)
    skillMatchScore += agentLevel;
  });
```

```
  // 2. Priority Multiplier (Higher priority tickets get preference to better
agents)
  const priorityMap = { 'P1': 2.0, 'P2': 1.5, 'P3': 1.0 };
  const priorityMultiplier = priorityMap[ticket.priority] || 1.0;

  // 3. Workload Penalty
  // Penalty increases quadratically with load ratio to strongly favor less busy
agents.
  const loadRatio = agent.currentLoad / agent.capacity;
  const workloadPenalty = loadRatio * loadRatio * 10; // Max penalty of 10 if
load = capacity

  // Final Score: (Skill Match * Priority Multiplier) - Workload Penalty
  let finalScore = (skillMatchScore * priorityMultiplier) - workloadPenalty;

  // If agent is over capacity, highly discourage assignment
  if (agent.currentLoad >= agent.capacity) {
      finalScore -= 1000;
  }

  return Math.max(0, finalScore); // Score can't be negative for simple ranking
};

// --- Initial Data Setup Functions ---

const initialAgents = [
  { id: 'agent_alice', name: 'Alice (FE)', team: 'Frontend', skills: { React: 5,
Python: 2, Billing: 1, Support: 4 }, capacity: 5, currentLoad: 0 },
  { id: 'agent_bob', name: 'Bob (BE)', team: 'Backend', skills: { React: 1,
Python: 5, Billing: 1, SQL: 4, Support: 3 }, capacity: 5, currentLoad: 0 },
  { id: 'agent_charlie', name: 'Charlie (Fin)', team: 'Billing', skills: { React:
0, Python: 0, Billing: 5, Support: 5 }, capacity: 4, currentLoad: 0 },
  { id: 'agent_diana', name: 'Diana (Support)', team: 'Frontend', skills:
{ React: 3, Python: 1, Support: 5 }, capacity: 6, currentLoad: 0 },
];

const initialTickets = [
  { title: "Critical Database Connection Failure", category: 'Technical Bug',
priority: 'P1', requiredSkills: ['Python', 'SQL'] },
  { title: "React Component Not Rendering", category: 'Technical Bug', priority:
'P2', requiredSkills: ['React'] },
```

```
  { title: "Monthly Invoice Discrepancy", category: 'Billing Inquiry', priority:
'P3', requiredSkills: ['Billing'] },
  { title: "General 'How To' Question", category: 'Feature Request', priority:
'P4', requiredSkills: ['Support'] },
  { title: "Bug in Checkout Flow Logic", category: 'Technical Bug', priority:
'P2', requiredSkills: ['Python', 'React'] },
];

const setupInitialData = async (userId) => {
  if (!db || !userId) return;

  const batch = writeBatch(db);
  const agentsCollection = collection(db, 'artifacts', appId, 'users', userId,
'agents');
  const ticketsCollection = collection(db, 'artifacts', appId, 'users', userId,
'tickets');

  // Set up Agents
  for (const agentData of initialAgents) {
    batch.set(doc(agentsCollection, agentData.id), { ...agentData, currentLoad:
0 });
  }

  // Set up Tickets
  for (const ticketData of initialTickets) {
    const newDocRef = doc(ticketsCollection);
    batch.set(newDocRef, {
      ...ticketData,
      id: newDocRef.id,
      status: 'New',
      assignedAgentId: '',
      createdAt: new Date().toISOString()
    });
  }

  try {
    await batch.commit();
    console.log("Initial data successfully seeded.");
  } catch (e) {
    console.error("Error setting up initial data:", e);
  }
};
```

```
// --- React Components ---

const AgentCard = React.memo(({ agent }) => {
  const loadColor = agent.currentLoad / agent.capacity > 0.8 ? 'bg-red-500' :
                    agent.currentLoad / agent.capacity > 0.5 ? 'bg-yellow-500' :
                    'bg-green-500';
  const teamClass = TEAM_COLORS[agent.team] || TEAM_COLORS.Default;

  return (
    <div className="p-4 bg-white rounded-xl shadow-lg border border-gray-100
hover:shadow-xl transition duration-300">
      <div className="flex items-center justify-between mb-3">
        <h3 className="text-xl font-bold text-gray-800">{agent.name}</h3>
        <span className={`px-3 py-1 text-xs font-semibold rounded-full
${teamClass}`}>
          {agent.team}
        </span>
      </div>
      <div className="text-sm text-gray-600 mb-4">
        <p className="font-semibold">Capacity:</p>
        <div className="flex items-center mt-1">
          <div className="w-full bg-gray-200 rounded-full h-2.5">
            <div
              className={`h-2.5 rounded-full ${loadColor}`}
              style={{ width: `${(agent.currentLoad / agent.capacity) * 100}%` }}
            ></div>
          </div>
          <span className="ml-2 font-mono text-gray-
700">{agent.currentLoad}/{agent.capacity}</span>
        </div>
      </div>
      <div className="mt-2">
        <p className="font-semibold text-gray-700 text-sm mb-1">Skills:</p>
        <div className="flex flex-wrap gap-1">
          {Object.entries(agent.skills).sort(([, levelA], [, levelB]) => levelB -
levelA).map(([skill, level]) => (
            level > 0 && (
              <span key={skill} className="px-2 py-0.5 text-xs bg-indigo-50 text-
indigo-700 rounded-full">
                {skill} ({level})
              </span>
            )
          ))}
```

```
        </div>
      </div>
    </div>
  );
});

const TicketCard = React.memo(({ ticket, onResolve }) => {
  const isNew = ticket.status === 'New';
  const isResolved = ticket.status === 'Resolved';
  const statusClass = isNew ? 'bg-purple-500' : isResolved ? 'bg-green-500' :
'bg-yellow-500';
  const priorityClass = ticket.priority === 'P1' ? 'bg-red-500' : ticket.priority
=== 'P2' ? 'bg-orange-500' : 'bg-gray-500';

  return (
    <div className="p-4 bg-white rounded-xl shadow border border-gray-100 flex
justify-between items-start">
      <div className="flex-grow">
        <div className="flex items-center gap-2 mb-2">
          <span className={`px-2 py-0.5 text-xs font-semibold text-white rounded-
full ${statusClass}`}>
            {ticket.status}
          </span>
          <span className={`px-2 py-0.5 text-xs font-semibold text-white rounded-
full ${priorityClass}`}>
            {ticket.priority}
          </span>
          <span className="px-2 py-0.5 text-xs font-semibold bg-gray-200 text-
gray-700 rounded-full">
            {ticket.category}
          </span>
        </div>
        <h3 className="text-lg font-semibold text-gray-800 mb-
1">{ticket.title}</h3>
        {ticket.status === 'Assigned' && (
          <p className="text-sm text-gray-600">
            Assigned to: <span className="font-bold text-indigo-
600">{ticket.assignedAgentName || 'N/A'}</span>
          </p>
        )}
        <div className="mt-2 flex flex-wrap gap-1">
          <p className="text-xs text-gray-500 font-medium">Required Skills:</p>
          {ticket.requiredSkills.map(skill => (
```

```
            <span key={skill} className="px-1.5 py-0.5 text-xs bg-pink-50 text-
pink-700 rounded-md">
              {skill}
            </span>
          ))}
        </div>
      </div>
      {!isResolved && ticket.status === 'Assigned' && onResolve && (
        <button
          onClick={() => onResolve(ticket.id, ticket.assignedAgentId)}
          className="ml-4 px-3 py-1 text-sm bg-green-600 text-white rounded-lg
hover:bg-green-700 transition"
        >
          Resolve
        </button>
      )}
    </div>
  );
});

const TicketForm = ({ userId }) => {
  const [newTicket, setNewTicket] = useState({
    title: '', category: ALL_CATEGORIES[0], priority: ALL_PRIORITIES[0],
requiredSkills: [ALL_SKILLS[0]]
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setNewTicket(prev => ({ ...prev, [name]: value }));
  };

  const handleSkillChange = (e) => {
    const { options } = e.target;
    const skills = [];
    for (let i = 0, l = options.length; i < l; i++) {
      if (options[i].selected) {
        skills.push(options[i].value);
      }
    }
    setNewTicket(prev => ({ ...prev, requiredSkills: skills }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
```

```
    if (!db || !userId) return;

    try {
      const ticketsCollection = collection(db, 'artifacts', appId, 'users',
userId, 'tickets');
      const docRef = await addDoc(ticketsCollection, {
        ...newTicket,
        status: 'New',
        assignedAgentId: '',
        createdAt: new Date().toISOString(),
      });
      await updateDoc(docRef, { id: docRef.id }); // Add ID to the document
itself
      setNewTicket({
        title: '', category: ALL_CATEGORIES[0], priority: ALL_PRIORITIES[0],
requiredSkills: [ALL_SKILLS[0]]
      });
      console.log("Ticket added with ID: ", docRef.id);
    } catch (e) {
      console.error("Error adding document: ", e);
    }
  };

  return (
    <form onSubmit={handleSubmit} className="p-6 bg-white rounded-xl shadow-lg
space-y-4">
      <h2 className="text-2xl font-bold text-gray-800 border-b pb-2 mb-4">Create
New Ticket</h2>
      <div>
        <label className="block text-sm font-medium text-gray-700">Title</label>
        <input
          type="text"
          name="title"
          value={newTicket.title}
          onChange={handleChange}
          required
          className="mt-1 block w-full rounded-md border-gray-300 shadow-sm
focus:border-indigo-500 focus:ring-indigo-500 p-2 border"
          placeholder="Brief description of the issue"
        />
      </div>
      <div className="grid grid-cols-3 gap-4">
        <div>
```

```
          <label className="block text-sm font-medium text-gray-
700">Category</label>
          <select name="category" value={newTicket.category}
onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300
shadow-sm focus:border-indigo-500 focus:ring-indigo-500 p-2 border">
            {ALL_CATEGORIES.map(cat => (
              <option key={cat} value={cat}>{cat}</option>
            ))}
          </select>
        </div>
        <div>
          <label className="block text-sm font-medium text-gray-
700">Priority</label>
          <select name="priority" value={newTicket.priority}
onChange={handleChange} className="mt-1 block w-full rounded-md border-gray-300
shadow-sm focus:border-indigo-500 focus:ring-indigo-500 p-2 border">
            {ALL_PRIORITIES.map(p => (
              <option key={p} value={p}>{p}</option>
            ))}
          </select>
        </div>
        <div>
          <label className="block text-sm font-medium text-gray-700">Required
Skills</label>
          <select name="requiredSkills" multiple={true}
value={newTicket.requiredSkills} onChange={handleSkillChange} className="mt-1
block w-full rounded-md border-gray-300 shadow-sm focus:border-indigo-500
focus:ring-indigo-500 p-2 border h-20">
            {ALL_SKILLS.map(skill => (
              <option key={skill} value={skill}>{skill}</option>
            ))}
          </select>
        </div>
      </div>
      <button
        type="submit"
        className="w-full flex justify-center py-2 px-4 border border-transparent
rounded-md shadow-sm text-sm font-medium text-white bg-indigo-600 hover:bg-
indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-
500 transition duration-150"
      >
        Submit Ticket
      </button>
    </form>
```

```
  );
};


function App() {
  const [appState, setAppState] = useState({
    userId: null,
    isLoading: true,
    agents: [],
    tickets: [],
  });
  const [isProcessing, setIsProcessing] = useState(false);
  const { userId, isLoading, agents, tickets } = appState;

  // 1. Authentication and Initialization
  useEffect(() => {
    if (!auth || !db) {
        setAppState(prev => ({ ...prev, isLoading: false }));
        console.error("Firebase not initialized. Check configuration.");
        return;
    }

    const unsubscribe = onAuthStateChanged(auth, async (user) => {
        if (!user) {
            try {
                if (initialAuthToken) {
                    await signInWithCustomToken(auth, initialAuthToken);
                } else {
                    await signInAnonymously(auth);
                }
            } catch (e) {
                console.error("Auth sign-in failed:", e);
                setAppState(prev => ({ ...prev, isLoading: false }));
                return;
            }
        }

        const currentUserId = auth.currentUser?.uid || crypto.randomUUID();
        setAppState(prev => ({ ...prev, userId: currentUserId, isLoading:
false }));

        // Check if data needs seeding (only once per user)
```

```
        const agentCheckDoc = doc(db, 'artifacts', appId, 'users', currentUserId,
'agents', initialAgents[0].id);
        const agentDoc = await getDoc(agentCheckDoc);
        if (!agentDoc.exists()) {
            await setupInitialData(currentUserId);
        }
    });

    return () => unsubscribe();
  }, []); // Run only once for setup

  // 2. Real-time Data Listeners
  useEffect(() => {
    if (!db || !userId) return;

    // Listener for Agents
    const agentsRef = collection(db, 'artifacts', appId, 'users', userId,
'agents');
    const unsubscribeAgents = onSnapshot(query(agentsRef), (snapshot) => {
      const agentsData = snapshot.docs.map(doc => doc.data());
      setAppState(prev => ({ ...prev, agents: agentsData.sort((a, b) =>
a.name.localeCompare(b.name)) }));
    }, (error) => {
      console.error("Error listening to agents:", error);
    });

    // Listener for Tickets
    const ticketsRef = collection(db, 'artifacts', appId, 'users', userId,
'tickets');
    const unsubscribeTickets = onSnapshot(query(ticketsRef), (snapshot) => {
      const rawTicketsData = snapshot.docs.map(doc => doc.data());

      // Hydrate tickets with agent names for display
      const ticketsData = rawTicketsData.map(ticket => ({
        ...ticket,
        assignedAgentName: agents.find(a => a.id ===
ticket.assignedAgentId)?.name || 'Unassigned',
      }));

      // Sort: New, then Assigned, then Resolved
      const sortedTickets = ticketsData.sort((a, b) => {
        const statusOrder = { 'New': 1, 'Assigned': 2, 'Resolved': 3 };
        return statusOrder[a.status] - statusOrder[b.status];
```

```
      });

      setAppState(prev => ({ ...prev, tickets: sortedTickets }));
    }, (error) => {
      console.error("Error listening to tickets:", error);
    });

    return () => {
      unsubscribeAgents();
      unsubscribeTickets();
    };
  }, [userId, agents.length]); // Re-run if userId changes or agent data
structure is loaded

  // 3. Routing Logic (Main Action)
  const routeTickets = useCallback(async () => {
    if (!db || !userId || isProcessing) return;

    setIsProcessing(true);
    console.log("Starting ticket routing process...");

    try {
      const batch = writeBatch(db);
      const newTickets = tickets.filter(t => t.status === 'New');
      const agentsMap = new Map(agents.map(a => [a.id, { ...a, ticketsToAssign:
0 }]));

      if (newTickets.length === 0) {
        console.log("No new tickets to route.");
        setIsProcessing(false);
        return;
      }

      for (const ticket of newTickets) {
        let bestAgent = null;
        let maxScore = -1;

        // Find the best agent for this ticket
        agentsMap.forEach((agent, agentId) => {
          // Calculate the score based on the agent's current *projected* load
          const currentAgentLoad = agent.currentLoad + agent.ticketsToAssign;
          const projectedAgent = { ...agent, currentLoad: currentAgentLoad };
```

```
          const score = calculateScore(ticket, projectedAgent);

          if (score > maxScore) {
            maxScore = score;
            bestAgent = agentId;
          }
        });

        if (bestAgent && maxScore > 0) {
          // Assign ticket to the best agent
          const ticketsCollection = collection(db, 'artifacts', appId, 'users',
userId, 'tickets');
          const agentRef = agentsMap.get(bestAgent);

          batch.update(doc(ticketsCollection, ticket.id), {
            status: 'Assigned',
            assignedAgentId: bestAgent,
          });

          // Update projected load for the batch
          agentRef.ticketsToAssign += 1;
          agentsMap.set(bestAgent, agentRef);
        } else {
          console.warn(`Ticket ${ticket.id} could not be optimally routed (Max
Score: ${maxScore}). Keeping in 'New' status.`);
        }
      }

      // Final Agent Load Update
      const agentsCollection = collection(db, 'artifacts', appId, 'users',
userId, 'agents');
      agentsMap.forEach((agent, agentId) => {
        if (agent.ticketsToAssign > 0) {
          batch.update(doc(agentsCollection, agentId), {
            currentLoad: agent.currentLoad + agent.ticketsToAssign,
          });
        }
      });

      await batch.commit();
      console.log(`Routing complete. ${newTickets.length} tickets processed.`);

    } catch (e) {
```

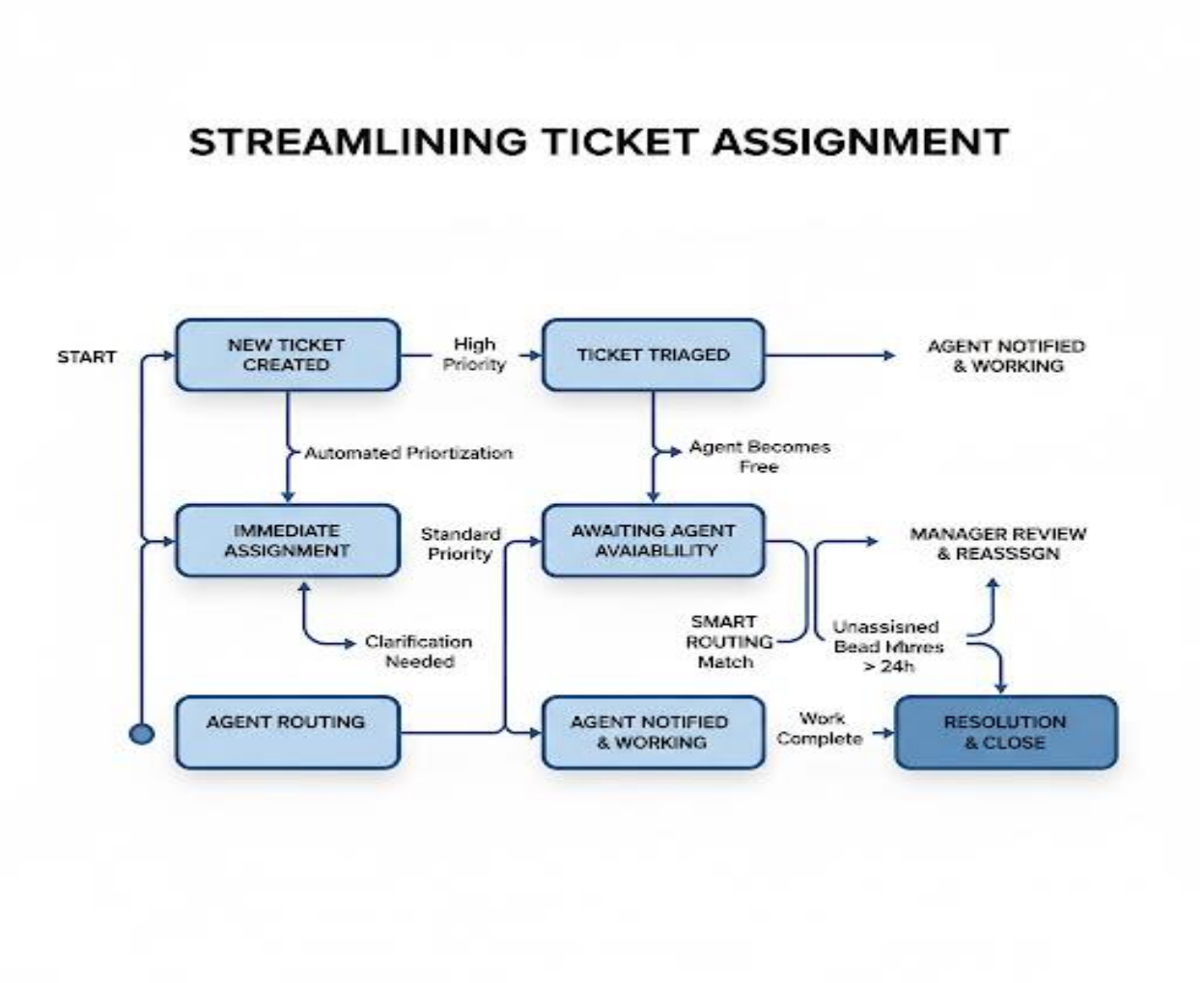
```
      console.error("Error during routing process:", e);
    } finally {
      setIsProcessing(false);
    }
  }, [userId, tickets, agents, isProcessing]);


  // 4. Resolve Ticket Logic
  const handleResolveTicket = useCallback(async (ticketId, agentId) => {
    if (!db || !userId || !ticketId || !agentId) return;

    try {
      const batch = writeBatch(db);

      // 1. Update Ticket Status
      const ticketsCollection = collection(db, 'artifacts', appId, 'users',
userId, 'tickets');
      batch.update(doc(ticketsCollection, ticketId), {
        status: 'Resolved',
        resolvedAt: new Date().toISOString(),
        assignedAgentId: '' // Clear assignment
      });

      // 2. Decrease Agent Load
      const agentToUpdate = agents.find(a => a.id === agentId);
      if (agentToUpdate && agentToUpdate.currentLoad > 0) {
        const agentsCollection = collection(db, 'artifacts', appId, 'users',
userId, 'agents');
        batch.update(doc(agentsCollection, agentId), {
          currentLoad: agentToUpdate.currentLoad - 1,
        });
      }

      await batch.commit();
      console.log(`Ticket ${ticketId} resolved and Agent load updated.`);

    } catch (e) {
      console.error("Error resolving ticket:", e);
    }
  }, [userId, agents]);

  // Loading/Error State Handling
  if (isLoading) {
```

```
    return <div className="min-h-screen flex items-center justify-center bg-gray-50"><p className="text-xl font-medium text-indigo-600">Loading Configuration and Authentication...</p></div>;
  }

  if (!db || !userId) {
    return <div className="min-h-screen flex items-center justify-center bg-red-50"><p className="text-xl font-medium text-red-700">System Error: Could not connect to the database.</p></div>;
  }

  const newTicketCount = tickets.filter(t => t.status === 'New').length;
  const assignedTicketCount = tickets.filter(t => t.status === 'Assigned').length;

  return (
    <div className="min-h-screen bg-gray-50 p-6 font-sans">
      <header className="text-center py-8 bg-white shadow-md rounded-lg mb-6">
        <h1 className="text-4xl font-extrabold text-gray-900">
          <span className="text-indigo-600">ABC Corp.</span> Ticket Routing Simulator
        </h1>
        <p className="mt-2 text-lg text-gray-600">Intelligent Skill-Based Assignment for Optimal Efficiency</p>
        <p className="mt-1 text-xs text-gray-400">User ID: {userId}</p>
      </header>

      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6 mb-8">
        <div className="lg:col-span-2">
          <TicketForm userId={userId} />
        </div>
        <div className="flex flex-col space-y-4">
          <button
            onClick={routeTickets}
            disabled={isProcessing || newTicketCount === 0}
            className={`w-full py-4 px-6 rounded-xl text-white font-bold text-lg shadow-xl transition duration-300 transform hover:scale-[1.01]
              ${isProcessing || newTicketCount === 0 ? 'bg-gray-400 cursor-not-allowed' : 'bg-indigo-600 hover:bg-indigo-700 active:bg-indigo-800'}`}
          >
            {isProcessing ? (
              <svg className="animate-spin -ml-1 mr-3 h-5 w-5 text-white inline" xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24">
```

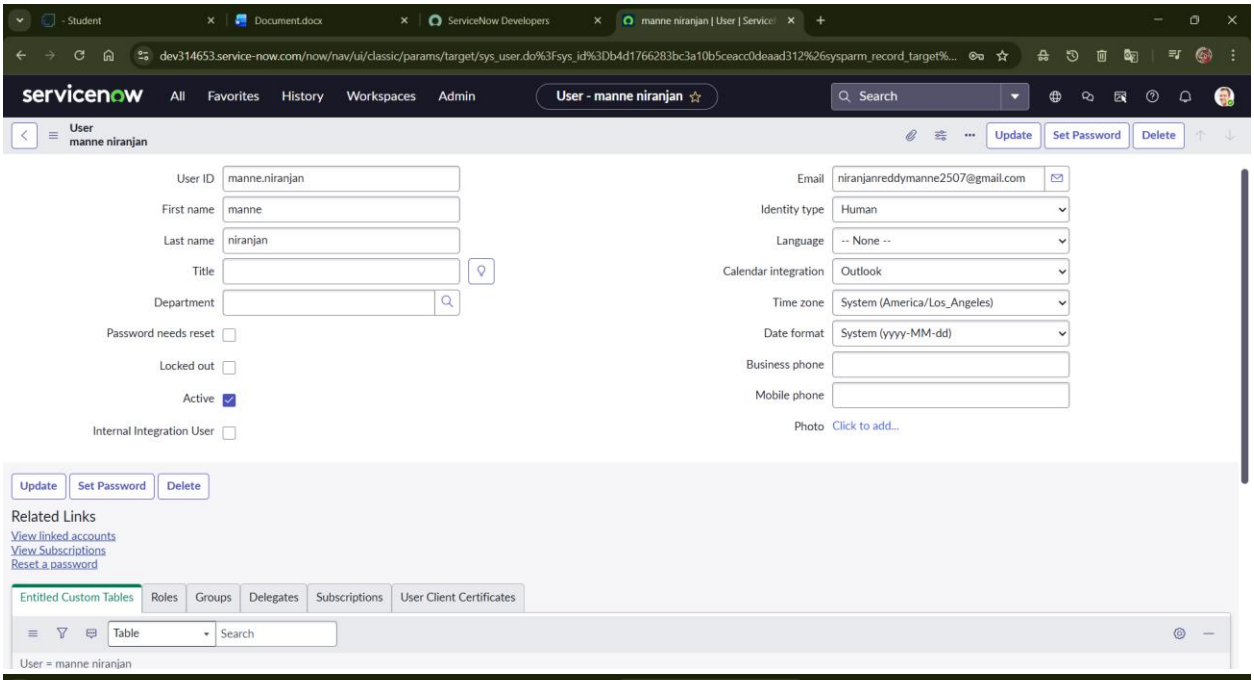
```
                <circle className="opacity-25" cx="12" cy="12" r="10"
stroke="currentColor" strokeWidth="4"></circle>
                <path className="opacity-75" fill="currentColor" d="M4 12a8 8 0
018-8V0C5.373 0 0 5.373 0 12h4zm2 5.291A7.962 7.962 0 014 12H0c0 3.042 1.135
5.824 3 7.938l3-2.647z"></path>
              </svg>
            ) : (
              <>
                Execute Intelligent Routing ({newTicketCount} New)
              </>
            )}
          </button>
          <div className="p-4 bg-yellow-50 border-l-4 border-yellow-500 text-
yellow-800 rounded-lg">
              <p className="font-semibold">Routing Status</p>
              <p className="text-sm">Assigned Tickets: {assignedTicketCount}</p>
              <p className="text-sm">Total Agents: {agents.length}</p>
          </div>
        </div>
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        {/* Agent/Team Panel */}
        <div>
          <h2 className="text-2xl font-bold text-gray-800 mb-4 border-b pb-
2">Agent Capacity & Skills</h2>
          <div className="space-y-4">
            {agents.map(agent => (
              <AgentCard key={agent.id} agent={agent} />
            ))}
          </div>
        </div>

        {/* Ticket Queue Panel */}
        <div>
          <h2 className="text-2xl font-bold text-gray-800 mb-4 border-b pb-
2">Ticket Queue ({tickets.length} Total)</h2>
          <div className="space-y-4">
            {tickets.length > 0 ? (
              tickets.map(ticket => (
                <TicketCard key={ticket.id} ticket={ticket}
onResolve={handleResolveTicket} />
              ))
            ) : (
```

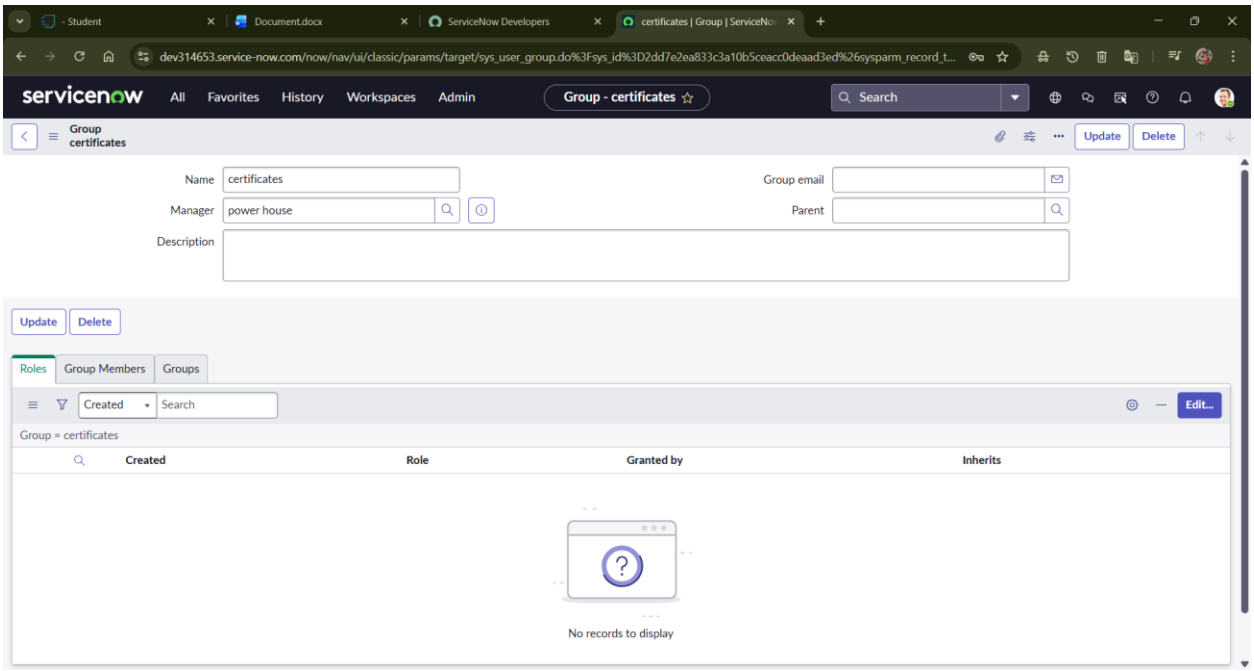
```
            <p className="text-gray-500 p-4 bg-white rounded-xl">The queue is
currently empty. Add a new ticket above!</p>
          )}
        </div>
      </div>
    </div>
  </div>
  );
}

export default App;
```

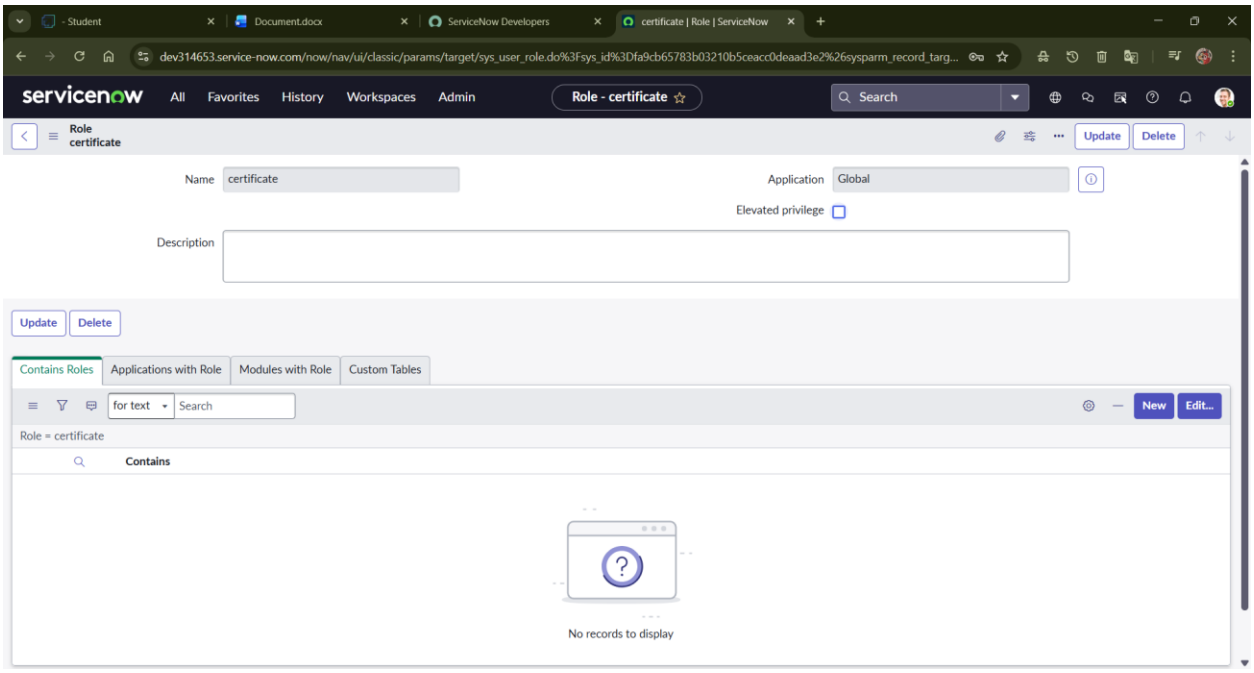
STATE-FLOW DIAGRAM:

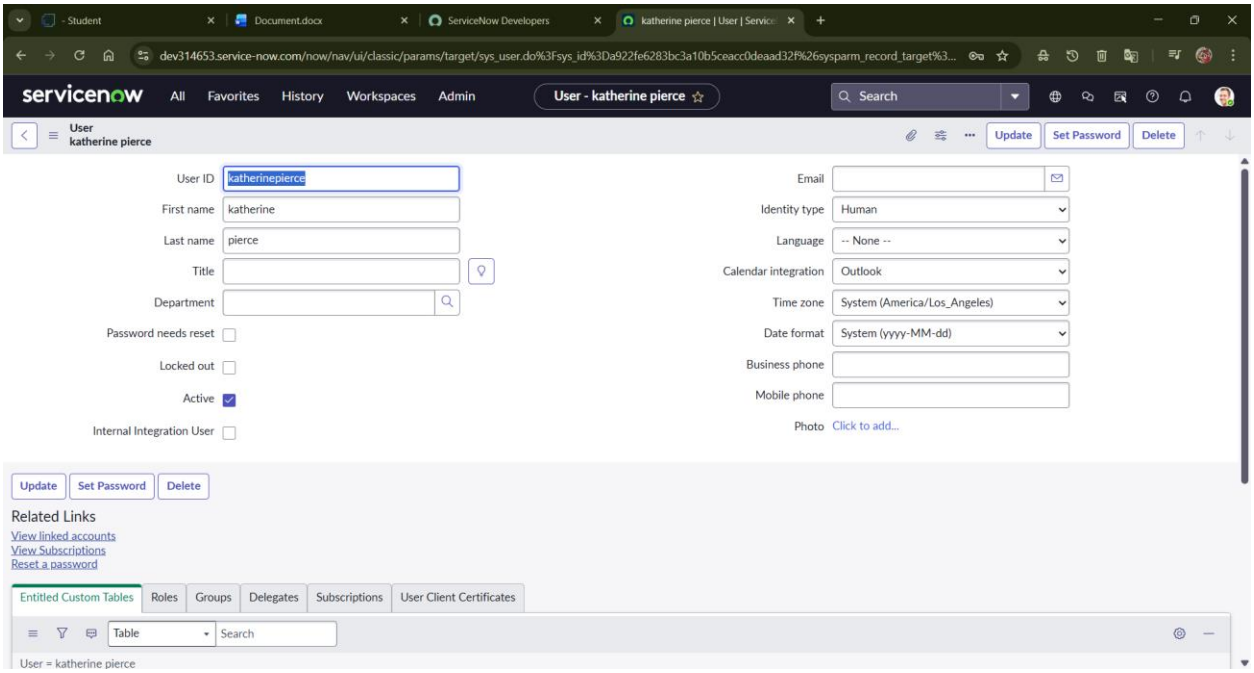


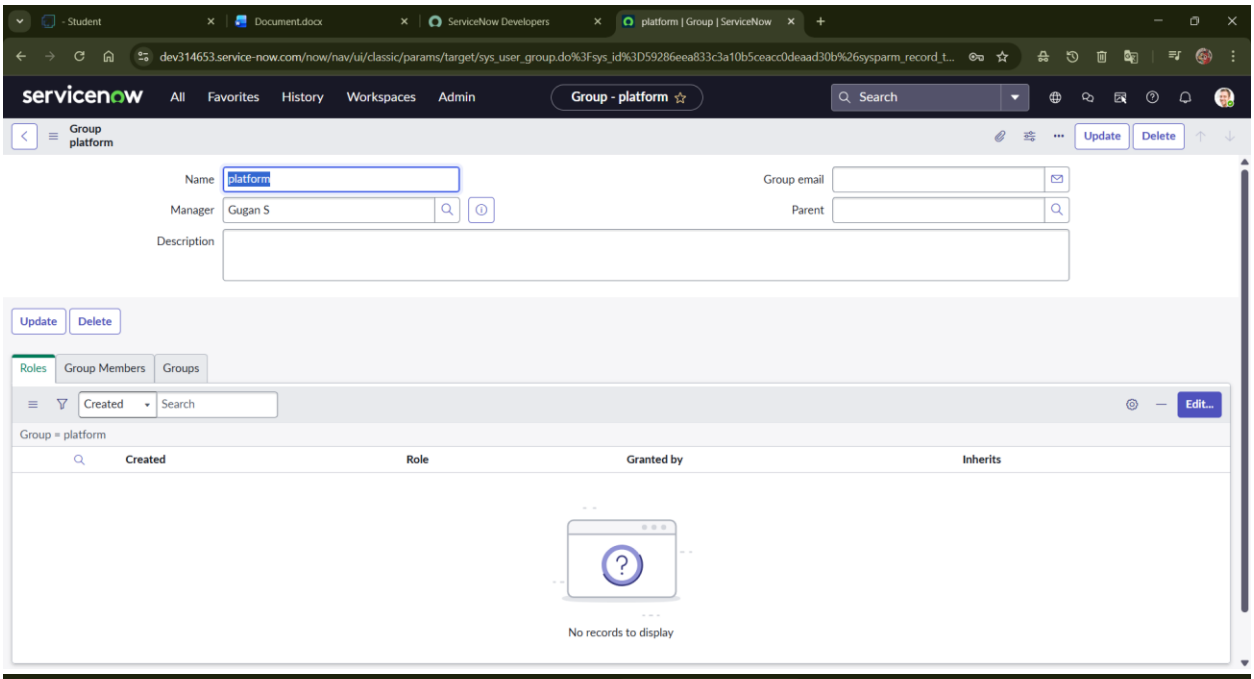
SCREENSHOTS:

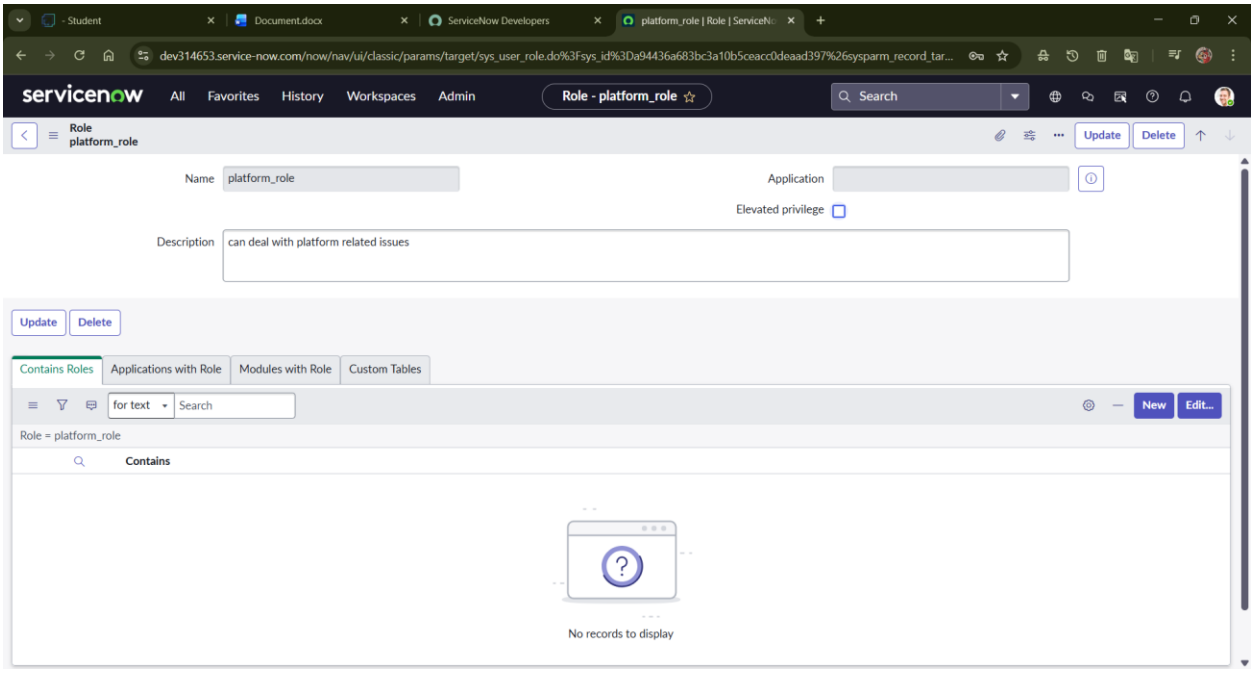





servicenow
All
Favorites
History
Workspaces
Admin
Role - certificate
Search
Role
certificate
Update
Delete
Name
certificate
Application
Global
Elevated privilege
Description
Update
Delete
Contains Roles
Applications with Role
Modules with Role
Custom Tables
for text
Search
New
Edit...
Role = certificate
Contains
No records to display


servicenow
All
Favorites
History
Workspaces
Admin
User - katherine pierce
Search
User
katherine pierce
Update
Set Password
Delete
User ID
katherinepierce
First name
katherine
Last name
pierce
Title
Department
Password needs reset
Locked out
Active
Internal Integration User
Email
Identity type
Human
Language
-- None --
Calendar integration
Outlook
Time zone
System (America/Los_Angeles)
Date format
System (yyyy-MM-dd)
Business phone
Mobile phone
Photo
Click to add...
Update
Set Password
Delete
Related Links
View linked accounts
View Subscriptions
Reset a password
Entitled Custom Tables
Roles
Groups
Delegates
Subscriptions
User Client Certificates
Table
Search
User = katherine pierce


servicenow
All
Favorites
History
Workspaces
Admin
Group - platform
Search
Group
platform
Update
Delete
Name
platform
Manager
Gugan S
Group email
Parent
Description
Update
Delete
Roles
Group Members
Groups
Created
Search
Edit...
Group = platform
Created
Role
Granted by
Inherits
No records to display


servicenow
All
Favorites
History
Workspaces
Admin
Role - platform_role
Search
Role
platform_role
Update
Delete
Name
platform_role
Application
Elevated privilege
Description
can deal with platform related issues
Update
Delete
Contains Roles
Applications with Role
Modules with Role
Custom Tables
for text
Search
New
Edit...
Role = platform_role
Contains
No records to display

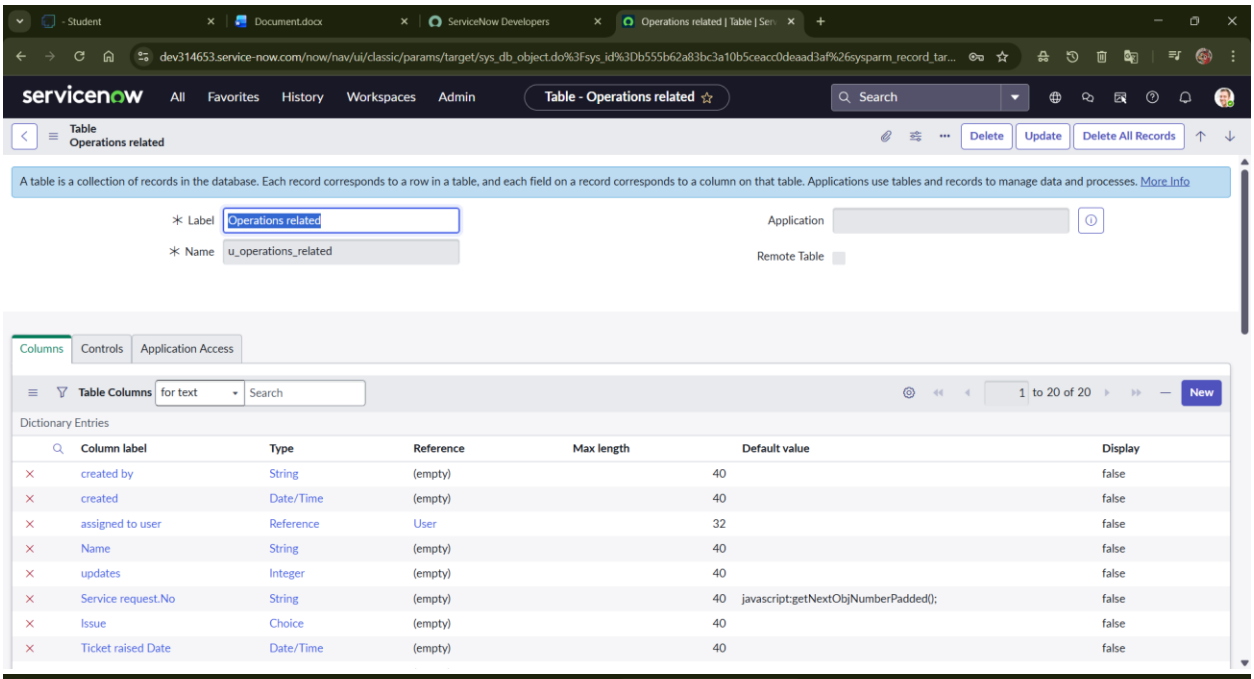
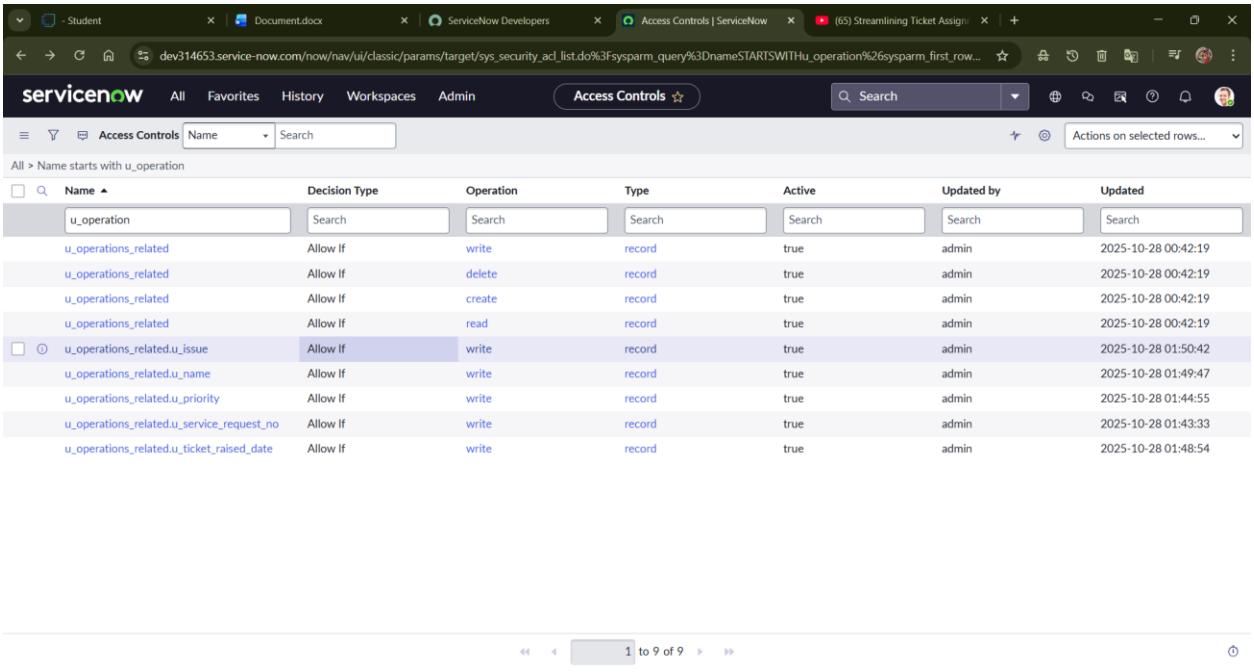
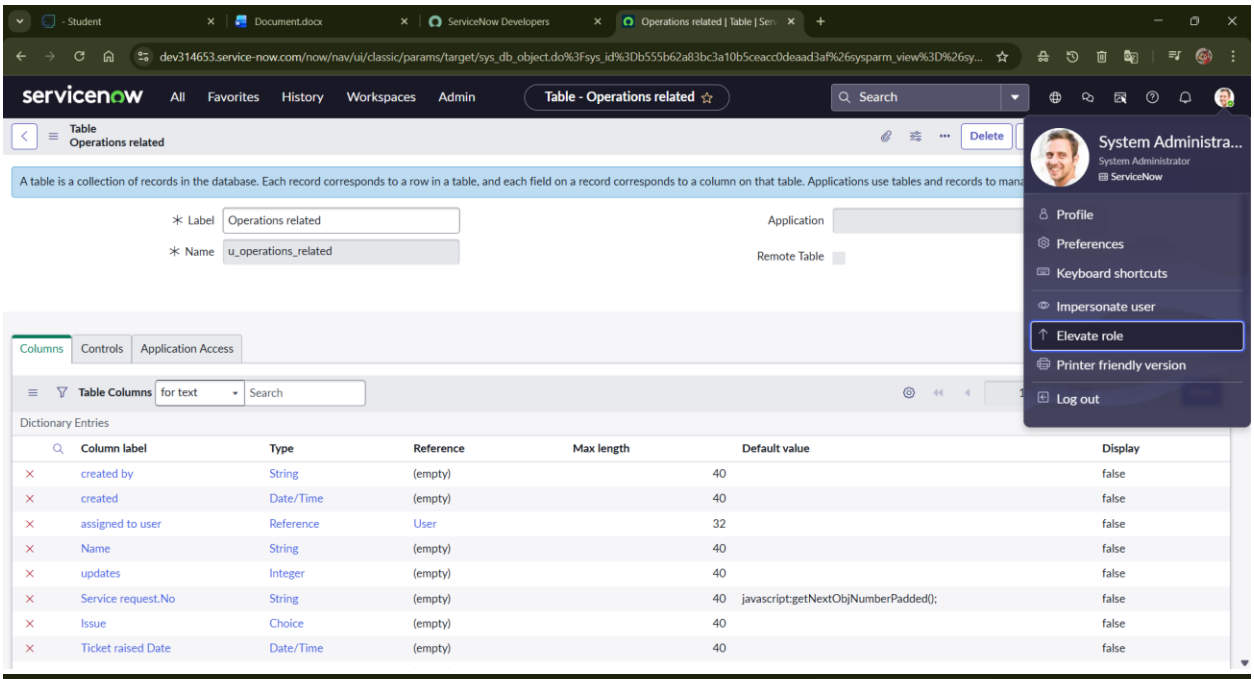
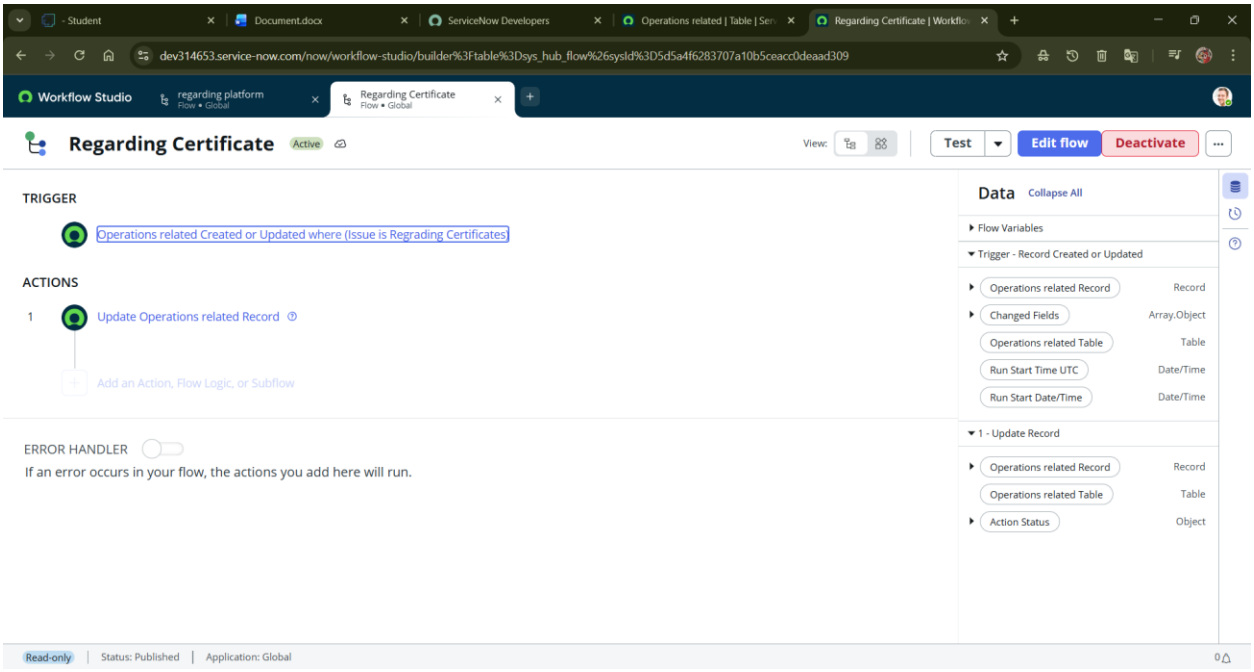
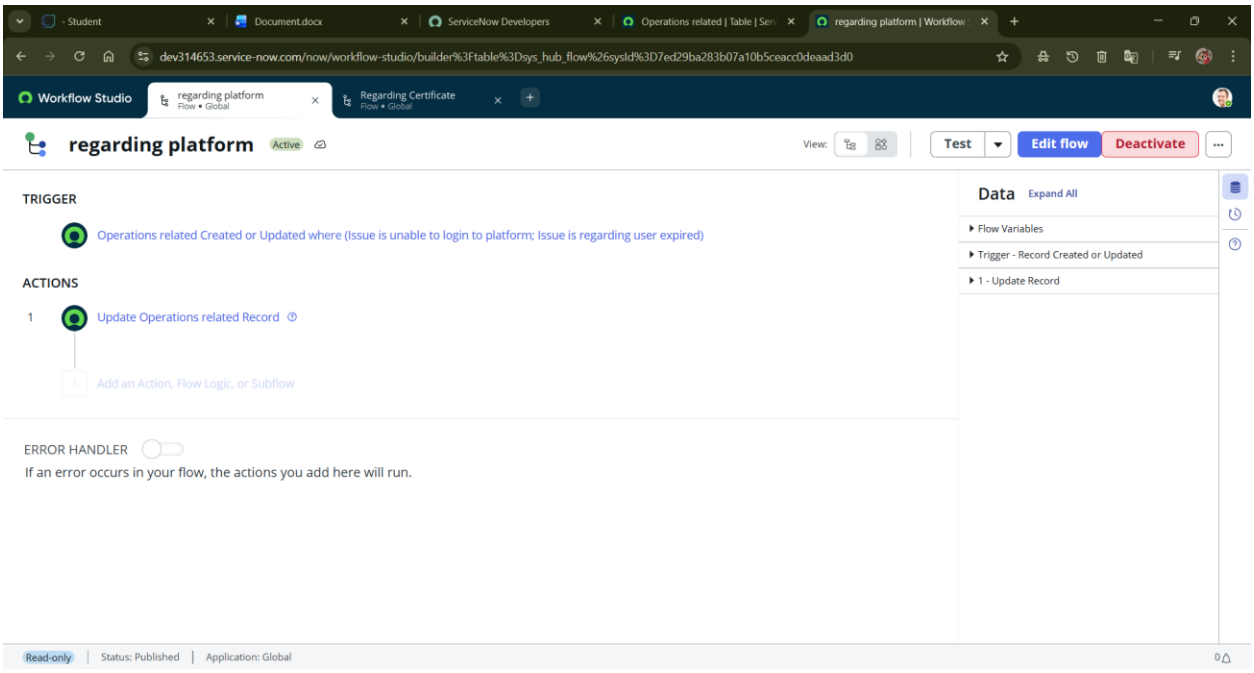


Table - Operations related
Table
Operations related
Delete
Update
Delete All Records
A table is a collection of records in the database. Each record corresponds to a row in a table, and each field on a record corresponds to a column on that table. Applications use tables and records to manage data and processes. More Info
Label
Operations related
Name
u_operations_related
Application
Remote Table
Columns
Controls
Application Access
Table Columns
for text
Search
1 to 20 of 20
New
Dictionary Entries
Column label
Type
Reference
Max length
Default value
Display
created by
String
(empty)
40
false
created
Date/Time
(empty)
40
false
assigned to user
Reference
User
32
false
Name
String
(empty)
40
false
updates
Integer
(empty)
40
false
Service request.No
String
(empty)
40
javascript:getNextObjNumberPadded();
false
Issue
Choice
(empty)
40
false
Ticket raised Date
Date/Time
(empty)
40
false


Access Controls
Access Controls
Name
Search
Actions on selected rows...
All > Name starts with u_operation
Name
Decision Type
Operation
Type
Active
Updated by
Updated
u_operation
u_operations_related
Allow If
write
record
true
admin
2025-10-28 00:42:19
u_operations_related
Allow If
delete
record
true
admin
2025-10-28 00:42:19
u_operations_related
Allow If
create
record
true
admin
2025-10-28 00:42:19
u_operations_related
Allow If
read
record
true
admin
2025-10-28 00:42:19
u_operations_related.u_issue
Allow If
write
record
true
admin
2025-10-28 01:50:42
u_operations_related.u_name
Allow If
write
record
true
admin
2025-10-28 01:49:47
u_operations_related.u_priority
Allow If
write
record
true
admin
2025-10-28 01:44:55
u_operations_related.u_service_request_no
Allow If
write
record
true
admin
2025-10-28 01:43:33
u_operations_related.u_ticket_raised_date
Allow If
write
record
true
admin
2025-10-28 01:48:54
1 to 9 of 9


servicenow
All
Favorites
History
Workspaces
Admin
Table - Operations related
Search
Table
Operations related
Delete
A table is a collection of records in the database. Each record corresponds to a row in a table, and each field on a record corresponds to a column on that table. Applications use tables and records to mana
Label
Operations related
Name
u_operations_related
Application
Remote Table
System Administra...
System Administrator
ServiceNow
Profile
Preferences
Keyboard shortcuts
Impersonate user
Elevate role
Printer friendly version
Log out
Columns
Controls
Application Access
Table Columns
for text
Search
Dictionary Entries
Column label
Type
Reference
Max length
Default value
Display
created by
String
(empty)
40
false
created
Date/Time
(empty)
40
false
assigned to user
Reference
User
32
false
Name
String
(empty)
40
false
updates
Integer
(empty)
40
false
Service request.No
String
(empty)
40
javascript:getNextObjNumberPadded();
false
Issue
Choice
(empty)
40
false
Ticket raised Date
Date/Time
(empty)
40
false


Workflow Studio
regarding platform
Flow • Global
Regarding Certificate
Flow • Global
Regarding Certificate
Active
View:
Test
Edit flow
Deactivate
TRIGGER
Operations related Created or Updated where (Issue is Regrading Certificates)
ACTIONS
1
Update Operations related Record
Add an Action, Flow Logic, or Subflow
ERROR HANDLER
If an error occurs in your flow, the actions you add here will run.
Data
Collapse All
Flow Variables
Trigger - Record Created or Updated
Operations related Record
Record
Changed Fields
Array.Object
Operations related Table
Table
Run Start Time UTC
Date/Time
Run Start Date/Time
Date/Time
1 - Update Record
Operations related Record
Record
Operations related Table
Table
Action Status
Object
Read-only
Status: Published
Application: Global


- Student
Document.docx
ServiceNow Developers
Operations related | Table | Ser
regarding platform | Workflow
dev314653.service-now.com/now/workflow-studio/builder%3Ftable%3Dsys_hub_flow%26sysId%3D7ed29ba283b07a10b5ceacc0deaad3d0
Workflow Studio
regarding platform
Flow • Global
Regarding Certificate
Flow • Global
regarding platform
Active
View:
Test
Edit flow
Deactivate
TRIGGER
Operations related Created or Updated where (Issue is unable to login to platform; Issue is regarding user expired)
ACTIONS
1
Update Operations related Record
Add an Action, Flow Logic, or Subflow
ERROR HANDLER
If an error occurs in your flow, the actions you add here will run.
Data
Expand All
Flow Variables
Trigger - Record Created or Updated
1 - Update Record
Read-only
Status: Published
Application: Global
0