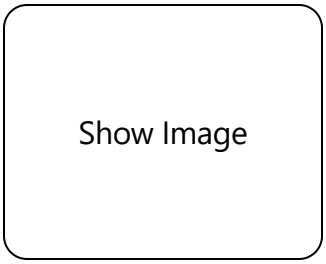
Human-in-the-Loop Salon AI Agent System

System Architecture



Components

1. Voice Communication Layer (LiveKit)

2. AI Agent with Basic Knowledge

3. Request Management System (Firebase)

4. Supervisor Interface (Web UI)

5. Knowledge Base Management

Implementation Details

1. Project Setup

Create a new directory and set up a Python environment:

```bash
mkdir salon-ai-agent
cd salon-ai-agent
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install flask firebase-admin livekit-server-sdk pydantic
```

2. AI Agent Implementation

Create `agent.py`:

python

```python
import json
import re
from datetime import datetime

class SalonAgent:
    def __init__(self, knowledge_base_path="knowledge_base.json"):
        self.knowledge_base_path = knowledge_base_path
        self.load_knowledge_base()

    def load_knowledge_base(self):
        try:
            with open(self.knowledge_base_path, 'r') as file:
                self.knowledge_base = json.load(file)
        except FileNotFoundError:
            # Initialize with basic salon information
            self.knowledge_base = {
                "name": "Style & Smile Salon",
                "address": "123 Beauty Lane, Fashion City",
                "phone": "555-123-4567",
                "hours": {
                    "monday": "9:00 AM - 7:00 PM",
                    "tuesday": "9:00 AM - 7:00 PM",
                    "wednesday": "9:00 AM - 7:00 PM",
                    "thursday": "9:00 AM - 8:00 PM",
                    "friday": "9:00 AM - 8:00 PM",
                    "saturday": "9:00 AM - 6:00 PM",
                    "sunday": "Closed"
                },
                "services": {
                    "haircut": "$45",
                    "color": "$85",
                    "blowout": "$35",
                    "manicure": "$30",
                    "pedicure": "$45"
                },
                "stylists": ["Alex", "Jamie", "Sam", "Taylor"],
                "learned_answers": {}
            }
            self.save_knowledge_base()

    def save_knowledge_base(self):
        with open(self.knowledge_base_path, 'w') as file:
            json.dump(self.knowledge_base, file, indent=2)
```

```python
def process_query(self, query):
    """Process user query and return response or None if unknown"""
    query = query.lower().strip()

    # Check for hours
    if re.search(r'(open|hours|time).*(today|tomorrow|\w+day)', query):
        day_match = re.search(r'(today|tomorrow|\w+day)', query)
        if day_match:
            day = day_match.group(1)
            if day == 'today':
                day = datetime.now().strftime('%A').lower()
            elif day == 'tomorrow':
                # Simple approach for demo purposes
                days = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday',
                today_idx = days.index(datetime.now().strftime('%A').lower())
                day = days[(today_idx + 1) % 7]

            if day in self.knowledge_base["hours"]:
                return f"We're open {day} from {self.knowledge_base['hours'][day]}."

    # Check for service pricing
    service_match = None
    for service in self.knowledge_base["services"]:
        if service in query:
            service_match = service
            break

    if service_match and ("price" in query or "cost" in query or "how much" in query):
        return f"A {service_match} costs {self.knowledge_base['services'][service_match]}."

    # Check learned answers
    for key_phrase, answer in self.knowledge_base["learned_answers"].items():
        if key_phrase in query:
            return answer

    # Unknown query
    return None

def update_knowledge_base(self, query, answer):
    """Add new knowledge based on supervisor response"""
    # Extract a key phrase from the query (simplified approach)
    key_words = query.lower().split()
    key_words = [word for word in key_words if len(word) > 3 and word not in
```

```
                ['what', 'when', 'where', 'which', 'how', 'does', 'is', 'are', 'can', 'will

        if key_words:
            # Use the most specific part of the query as the key phrase
            key_phrase = " ".join(key_words[:3])  # Use up to first 3 key words
            self.knowledge_base["learned_answers"][key_phrase] = answer
            self.save_knowledge_base()
            return key_phrase
        return None
```

3. Request Management System (Firebase)

Create `db_manager.py`:

python

```python
import firebase_admin
from firebase_admin import credentials, firestore
import uuid
from datetime import datetime

class RequestManager:
    def __init__(self):
        # Initialize Firebase (in production, use environment variables)
        cred = credentials.Certificate("serviceAccountKey.json")
        firebase_admin.initialize_app(cred)
        self.db = firestore.client()

    def create_help_request(self, customer_phone, query, call_id):
        """Create a new help request from the AI agent"""
        request_id = str(uuid.uuid4())
        request_data = {
            'id': request_id,
            'customer_phone': customer_phone,
            'query': query,
            'call_id': call_id,
            'status': 'pending',
            'created_at': datetime.now(),
            'updated_at': datetime.now()
        }

        self.db.collection('requests').document(request_id).set(request_data)
        print(f"Help request created: {request_id} for phone: {customer_phone}")

        # In a real system, you would send a notification to the supervisor here
        self._simulate_notify_supervisor(request_id, query)

        return request_id

    def _simulate_notify_supervisor(self, request_id, query):
        """Simulate texting the supervisor"""
        print(f"\n=== NOTIFICATION TO SUPERVISOR ===")
        print(f"Request ID: {request_id}")
        print(f"Customer query: {query}")
        print(f"Please respond at: http://localhost:5000/supervisor")
        print(f"==================================\n")

    def get_pending_requests(self):
        """Get all pending help requests"""
```

```
    requests = self.db.collection('requests').where('status', '==', 'pending').get()
    return [doc.to_dict() for doc in requests]

def get_request_history(self, limit=50):
    """Get request history, both resolved and unresolved"""
    requests = self.db.collection('requests').order_by(
        'created_at', direction=firestore.Query.DESCENDING
    ).limit(limit).get()
    return [doc.to_dict() for doc in requests]

def resolve_request(self, request_id, answer):
    """Resolve a help request with supervisor's answer"""
    request_ref = self.db.collection('requests').document(request_id)
    request_data = request_ref.get().to_dict()

    if not request_data:
        return False, "Request not found"

    if request_data['status'] != 'pending':
        return False, "Request is already resolved or timed out"

    # Update request
    request_ref.update({
        'status': 'resolved',
        'answer': answer,
        'resolved_at': datetime.now(),
        'updated_at': datetime.now()
    })

    # Simulate notification back to customer
    self._simulate_notify_customer(request_data['customer_phone'], answer)

    return True, "Request resolved successfully"

def _simulate_notify_customer(self, phone, answer):
    """Simulate texting the customer with the answer"""
    print(f"\n=== NOTIFICATION TO CUSTOMER ===")
    print(f"To: {phone}")
    print(f"Message: Thank you for your patience. {answer}")
    print(f"================================\n")

def mark_request_unresolved(self, request_id, reason="Timed out"):
    """Mark a request as unresolved (e.g., due to timeout)"""
    request_ref = self.db.collection('requests').document(request_id)
```

```
        request_ref.update({
            'status': 'unresolved',
            'unresolved_reason': reason,
            'updated_at': datetime.now()
        })

        # Could add customer notification here for the unresolved case

        return True
```

4. Voice Communication with LiveKit

Create `call_handler.py`:

python

```python
from livekit import rtc
import asyncio
import json
from agent import SalonAgent
from db_manager import RequestManager

class CallHandler:
    def __init__(self):
        self.agent = SalonAgent()
        self.request_manager = RequestManager()

    async def handle_call(self, room_name, participant_identity):
        """Handle an incoming call through LiveKit"""
        # In a real implementation, connect to LiveKit room
        # For demo purposes, we'll simulate the call

        # Simulated customer query
        customer_phone = "+1234567890"  # In real world, extract from participant data
        customer_query = "Do you offer balayage hair coloring?"

        print(f"Incoming call from {customer_phone}: '{customer_query}'")

        # Process with AI agent
        response = self.agent.process_query(customer_query)

        if response:
            print(f"AI response: {response}")
            # In real implementation, convert to speech and send via LiveKit
            return {"status": "answered", "response": response}
        else:
            # Agent doesn't know, escalate to human
            print("AI response: Let me check with my supervisor and get back to you.")

            # Create help request
            request_id = self.request_manager.create_help_request(
                customer_phone, customer_query, room_name
            )

            return {
                "status": "escalated",
                "request_id": request_id,
                "response": "Let me check with my supervisor and get back to you."
            }
```

```
    def handle_resolution(self, request_id, supervisor_answer):
        """Handle supervisor providing answer to escalated request"""
        # Get the request data from database
        # Update the knowledge base
        request_data = self.request_manager.db.collection('requests').document(request_id).get(

        if request_data:
            # Update the knowledge base
            key_phrase = self.agent.update_knowledge_base(request_data['query'], supervisor_ans
            print(f"Knowledge base updated with key phrase: '{key_phrase}'")

            # Resolve the request in the database
            success, message = self.request_manager.resolve_request(request_id, supervisor_answ

            return success, message

        return False, "Request not found"

# Simple demonstration of how this would work
async def simulate_call():
    handler = CallHandler()
    result = await handler.handle_call("room-123", "customer-456")
    print(f"Call result: {result}")

    if result['status'] == 'escalated':
        # Simulate the supervisor answering after some time
        await asyncio.sleep(5)  # Simulate passage of time

        # Supervisor provides an answer
        handler.handle_resolution(
            result['request_id'],
            "Yes, we offer balayage services starting at $120, depending on hair length."
        )

# Run simulation if this file is executed directly
if __name__ == "__main__":
    asyncio.run(simulate_call())
```

5. Web Interface for Supervisors

Create `app.py`:

python

```python
from flask import Flask, render_template, request, jsonify, redirect, url_for
from db_manager import RequestManager
from agent import SalonAgent
from call_handler import CallHandler
import asyncio

app = Flask(__name__)
request_manager = RequestManager()
salon_agent = SalonAgent()
call_handler = CallHandler()

@app.route('/')
def index():
    """Homepage - redirect to supervisor dashboard"""
    return redirect(url_for('supervisor_dashboard'))

@app.route('/supervisor')
def supervisor_dashboard():
    """Supervisor dashboard showing pending requests"""
    pending_requests = request_manager.get_pending_requests()
    return render_template('supervisor.html', pending_requests=pending_requests)

@app.route('/history')
def request_history():
    """Show history of all requests"""
    requests = request_manager.get_request_history()
    return render_template('history.html', requests=requests)

@app.route('/knowledge')
def knowledge_base():
    """View and edit the knowledge base"""
    salon_agent.load_knowledge_base()  # Refresh from file
    return render_template('knowledge.html', knowledge=salon_agent.knowledge_base)

@app.route('/resolve', methods=['POST'])
def resolve_request():
    """Resolve a pending request with supervisor's answer"""
    request_id = request.form.get('request_id')
    answer = request.form.get('answer')

    if not request_id or not answer:
        return jsonify({"success": False, "message": "Missing request ID or answer"})
```

```
    success, message = call_handler.handle_resolution(request_id, answer)

    return jsonify({"success": success, "message": message})

@app.route('/simulate-call')
def simulate_call():
    """Simulate an incoming call for testing purposes"""
    async def run():
        return await call_handler.handle_call("test-room", "test-customer")

    result = asyncio.run(run())
    return jsonify(result)

if __name__ == '__main__':
    app.run(debug=True)
```

6. HTML Templates

Create directory `templates/` and add the following files:

templates/layout.html

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Salon AI Assistant - {% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/boo
    <style>
        body { padding-top: 20px; }
        .pending { background-color: #fff3cd; }
        .resolved { background-color: #d1e7dd; }
        .unresolved { background-color: #f8d7da; }
        .navbar { margin-bottom: 20px; }
    </style>
</head>
<body>
    <div class="container">
        <nav class="navbar navbar-expand-lg navbar-light bg-light rounded">
            <div class="container-fluid">
                <a class="navbar-brand" href="/">Salon AI Assistant</a>
                <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse" id="navbarNav">
                    <ul class="navbar-nav">
                        <li class="nav-item">
                            <a class="nav-link" href="/supervisor">Pending Requests</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="/history">Request History</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="/knowledge">Knowledge Base</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link" href="/simulate-call">Simulate Call</a>
                        </li>
                    </ul>
                </div>
            </div>
        </nav>

        <div class="my-4">
```

```
            {% block content %}{% endblock %}
        </div>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.mir
    {% block scripts %}{% endblock %}
</body>
</html>
```

templates/supervisor.html

```
html
```

```
{% extends "layout.html" %}

{% block title %}Supervisor Dashboard{% endblock %}

{% block content %}
    <h1>Pending Requests</h1>

    {% if pending_requests %}
        <div class="row">
            {% for req in pending_requests %}
                <div class="col-md-6 mb-3">
                    <div class="card pending">
                        <div class="card-header">
                            Request #{{ req.id[:8] }} - {{ req.created_at.strftime('%Y-%m-%d %H
                        </div>
                        <div class="card-body">
                            <h5 class="card-title">Customer: {{ req.customer_phone }}</h5>
                            <p class="card-text"><strong>Query:</strong> {{ req.query }}</p>

                            <form class="resolve-form mt-3">
                                <input type="hidden" name="request_id" value="{{ req.id }}">
                                <div class="mb-3">
                                    <label for="answer-{{ req.id }}" class="form-label">Your re
                                    <textarea class="form-control" id="answer-{{ req.id }}" nam
                                </div>
                                <button type="submit" class="btn btn-primary">Submit Response</
                            </form>
                        </div>
                    </div>
                </div>
            {% endfor %}
        </div>
    {% else %}
        <div class="alert alert-info">
            No pending requests at this time.
        </div>
    {% endif %}
{% endblock %}

{% block scripts %}
<script>
    document.addEventListener('DOMContentLoaded', function() {
        const forms = document.querySelectorAll('.resolve-form');
```

```
        forms.forEach(form => {
            form.addEventListener('submit', function(e) {
                e.preventDefault();

                const requestId = this.querySelector('[name="request_id"]').value;
                const answer = this.querySelector('[name="answer"]').value;

                fetch('/resolve', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/x-www-form-urlencoded',
                    },
                    body: `request_id=${encodeURIComponent(requestId)}&answer=${encodeURICompor
                })
                .then(response => response.json())
                .then(data => {
                    if (data.success) {
                        alert('Request resolved successfully!');
                        // Reload the page to refresh the list
                        window.location.reload();
                    } else {
                        alert('Error: ' + data.message);
                    }
                })
                .catch(error => {
                    console.error('Error:', error);
                    alert('An error occurred while processing your request.');
                });
            });
        });
    });
</script>
{% endblock %}
```

templates/history.html

```
html
```

```
{% extends "layout.html" %}

{% block title %}Request History{% endblock %}

{% block content %}
    <h1>Request History</h1>

    {% if requests %}
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Customer</th>
                    <th>Query</th>
                    <th>Status</th>
                    <th>Created</th>
                    <th>Response</th>
                </tr>
            </thead>
            <tbody>
                {% for req in requests %}
                    <tr class="{{ req.status }}">
                        <td>{{ req.id[:8] }}</td>
                        <td>{{ req.customer_phone }}</td>
                        <td>{{ req.query }}</td>
                        <td>{{ req.status }}</td>
                        <td>{{ req.created_at.strftime('%Y-%m-%d %H:%M') }}</td>
                        <td>
                            {% if req.status == 'resolved' %}
                                {{ req.answer }}
                            {% elif req.status == 'unresolved' %}
                                Reason: {{ req.unresolved_reason }}
                            {% else %}
                                Pending
                            {% endif %}
                        </td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <div class="alert alert-info">
            No request history available.
```

```
        </div>
    {% endif %}
{% endblock %}
```

templates/knowledge.html

```
html
```

```
{% extends "layout.html" %}

{% block title %}Knowledge Base{% endblock %}

{% block content %}
    <h1>Knowledge Base</h1>

    <div class="row">
        <div class="col-md-6">
            <div class="card mb-4">
                <div class="card-header">
                    <h5>Basic Salon Information</h5>
                </div>
                <div class="card-body">
                    <p><strong>Name:</strong> {{ knowledge.name }}</p>
                    <p><strong>Address:</strong> {{ knowledge.address }}</p>
                    <p><strong>Phone:</strong> {{ knowledge.phone }}</p>
                    <h6>Hours:</h6>
                    <ul>
                        {% for day, hours in knowledge.hours.items() %}
                            <li>{{ day.capitalize() }}: {{ hours }}</li>
                        {% endfor %}
                    </ul>
                </div>
            </div>

            <div class="card">
                <div class="card-header">
                    <h5>Services</h5>
                </div>
                <div class="card-body">
                    <ul>
                        {% for service, price in knowledge.services.items() %}
                            <li>{{ service.capitalize() }}: {{ price }}</li>
                        {% endfor %}
                    </ul>
                </div>
            </div>
        </div>

        <div class="col-md-6">
            <div class="card">
                <div class="card-header">
```

```
                    <h5>Learned Answers</h5>
                </div>
                <div class="card-body">
                    {% if knowledge.learned_answers %}
                        <table class="table table-sm">
                            <thead>
                                <tr>
                                    <th>Key Phrase</th>
                                    <th>Answer</th>
                                </tr>
                            </thead>
                            <tbody>
                                {% for key, answer in knowledge.learned_answers.items() %}
                                    <tr>
                                        <td>{{ key }}</td>
                                        <td>{{ answer }}</td>
                                    </tr>
                                {% endfor %}
                            </tbody>
                        </table>
                    {% else %}
                        <p>No learned answers yet.</p>
                    {% endif %}
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

7. Putting It All Together

Create a `main.py` file to run the system:

```python
from app import app
import asyncio
from call_handler import simulate_call

if __name__ == "__main__":
    # Simulate a call on startup (optional)
    asyncio.run(simulate_call())

    # Start the Flask application
    app.run(debug=True)
```

8. Firebase Setup Instructions

To set up Firebase:

1. Create a Firebase account and project
2. Generate a service account key from Firebase Console
3. Save the key as `serviceAccountKey.json` in your project directory
4. Create Firestore database and configure rules

Running the Application

To run the application:

1. Install dependencies:

```bash
pip install flask firebase-admin livekit-server-sdk
```

2. Start the application:

```bash
python main.py
```

3. Open your browser to `http://localhost:5000/` to access the supervisor interface

Testing the System

1. Use the "Simulate Call" button to generate test calls

2. View pending requests and respond as a supervisor
3. Check that the knowledge base is updated with new information
4. Verify that customers receive follow-up messages (simulated in console)

Future Enhancements

1. Add authentication for supervisors
2. Implement timeouts and automatic follow-up for unresolved requests
3. Improve the AI's natural language understanding capabilities
4. Add real SMS/calling capabilities via Twilio or similar service
5. Implement a more sophisticated learning system for the AI