Ideia de Arquitetura de Software - Dashboard de Atendimento Multicanal

Esta arquitetura detalha os componentes e as interações para o seu dashboard de atendimento multicanal, incorporando as tecnologias e funcionalidades solicitadas no backlog do projeto.

1. Visão Geral da Arquitetura

Imagem de highlevel architecture diagram showing Frontend, Backend, Database, and external integrations

A arquitetura proposta segue um modelo de microsserviços ou um monólito bem modularizado, utilizando **Next.js** para o frontend, **NestJS** para o backend e **PostgreSQL** como banco de dados principal. A comunicação em tempo real será gerenciada via WebSockets.

2. Componentes Principais

2.1. Frontend (Next.js)

- **Tecnologia:** Next.js (React Framework)
- **Função:** Responsável pela interface do usuário (UI) e experiência do usuário (UX). Oferece renderização no lado do servidor (SSR) e geração de sites estáticos (SSG) para melhor performance e SEO.
- **Módulos/Páginas:**
  - **Dashboard de Visão Geral:** Exibe métricas e status de todos os canais (WhatsApp, e-mail, Facebook).
  - **Interface de Chat:** Simula a experiência do WhatsApp, com histórico de mensagens, envio de texto, arquivos e sugestões de IA.
  - **Gestão de Canais:** Configuração de contas de WhatsApp (com QR Code), e-mail e Facebook. Inclui interface para cadastro, validação e teste de conexão, e gerenciamento de canais.
  - **Gestão de Filas:** Criação e gerenciamento de filas de atendimento por canal.
  - **Gestão de Usuários e Permissões:** Interface para adicionar/remover usuários, associá-los a contas, e definir permissões por canal e fila, incluindo a gestão de papéis personalizados.
  - **Construtor Visual de Fluxos de Automação:** Interface de arrastar e soltar para criar e editar fluxos (inspirado no Typebot).
  - **Configurações:** Modo claro/escuro e outras preferências do usuário.
  - **Visualização de Logs/Auditoria:** Interface para administradores rastrearem ações importantes no sistema.

- **Estilo:** Utilização de bibliotecas de UI (ex: Tailwind CSS ou Material-UI) para um visual limpo, profissional e responsivo.

2.2. Backend (NestJS)

- **Tecnologia:** NestJS (Node.js Framework)
- **Função:** Camada de API robusta e escalável. Gerencia a lógica de negócios, autenticação, comunicação com o banco de dados e integração com serviços externos.
- **Endpoints e Módulos/Serviços (Baseado no Backlog):**
  - **Módulo de Autenticação e Autorização:**
    - Endpoints: /accounts (cadastro de conta), /users (cadastro de usuário), /auth/login (login).
    - Implementa autenticação com JWT (geração de tokens de acesso e refresh).
    - Middleware de verificação de token.
    - Gerencia controle de acesso baseado em permissões (RBAC) e papéis personalizados (custom_roles).
  - **Módulo de Usuários e Contas:**
    - Endpoints: /accounts/{id}/users (listar, adicionar, remover usuários de uma conta).
    - CRUD para gestão de usuários e seus perfis.
    - Associação de usuários a contas.
    - Envio de e-mail de confirmação e ativação de conta.
  - **Módulo de Canais:**
    - Endpoints: /channels/whatsapp (cadastro de canal WhatsApp), /channels/{id}/test-connection, /channels/{id} (edição/remoção).
    - **Serviço WhatsApp:** Integração com a API do WhatsApp Business (ou solução de terceiros para multi-contas). Gerencia a conexão (QR Code), envio/recebimento de mensagens e status.
    - **Serviço E-mail:** Integração com provedores de e-mail (ex: SendGrid, Mailgun).
    - **Serviço Facebook:** Integração com a API do Facebook Messenger.
  - **Módulo de Filas de Atendimento:** Lógica para roteamento de conversas para as filas corretas e atribuição a atendentes.
  - **Módulo de Chat (WebSockets):** Utiliza Socket.IO para comunicação em tempo real entre o frontend e o backend, permitindo o envio e recebimento instantâneo de mensagens.
  - **Módulo de Histórico de Conversas:** Armazena e recupera o histórico completo de todas as interações.

- **Módulo de IA para Sugestões:** Integração com um modelo de linguagem (ex: Gemini API) para gerar sugestões de respostas com base no contexto da conversa.
- **Módulo de Automação de Fluxos:**
  - **Interpretador de Fluxos:** Executa as regras e etapas definidas no construtor visual de fluxos.
  - **Engine de Transição:** Gerencia a transição de um fluxo automatizado para um atendente humano.
- **Módulo de Armazenamento de Arquivos:** Integração com serviços de armazenamento em nuvem (ex: AWS S3, Google Cloud Storage) para upload e download de arquivos anexados às conversas.
- **Módulo de Contatos:**
  - Endpoints: /contacts (listar, buscar, filtrar).
  - Lógica para criar/atualizar contatos automaticamente ao receber mensagens.
- **Módulo de Webhooks:**
  - Endpoint seguro para receber mensagens do WhatsApp (e outros canais).
  - Verificação de autenticidade das requisições e garantia de tolerância a falhas.
- **Módulo de Logs e Auditoria:**
  - Registra eventos importantes como login, envio de mensagem, alteração de permissões na tabela audits.

2.3. Banco de Dados (PostgreSQL)

- **Tecnologia:** PostgreSQL
- **Função:** Banco de dados relacional robusto e escalável para armazenar dados estruturados.
- **Tabelas Principais (Com base no Backlog):**
  - accounts:
    - id (PK)
    - name (nome da empresa)
    - support_email
    - domain (domínio único)
    - created_at, updated_at
  - users:
    - id (PK)
    - account_id (FK para accounts.id)
    - email
    - password_hash

        - is_active (para confirmação de e-mail)
        - created_at, updated_at
    - access_tokens:
        - id (PK)
        - user_id (FK para users.id)
        - token (token de acesso ou refresh)
        - type (e.g., 'access', 'refresh')
        - expires_at
        - created_at
    - roles: (Pré-definidos, e.g., 'admin', 'agent')
        - id (PK)
        - name
        - description
    - custom_roles: (Para papéis personalizados por conta)
        - id (PK)
        - account_id (FK para accounts.id)
        - name
        - permissions (JSONB para armazenar permissões específicas)
    - user_roles: (Relação entre usuários e papéis pré-definidos)
        - user_id (FK para users.id)
        - role_id (FK para roles.id)
        - (Chave composta: user_id, role_id)
    - user_custom_roles: (Relação entre usuários e papéis personalizados)
        - user_id (FK para users.id)
        - custom_role_id (FK para custom_roles.id)
        - (Chave composta: user_id, custom_role_id)
    - channels:
        - id (PK)
        - account_id (FK para accounts.id)
        - type (e.g., 'whatsapp', 'email', 'facebook')
        - name
        - status (e.g., 'connected', 'disconnected')
        - provider_config (JSONB para configurações específicas do provedor, como numero, provedor, templates para WhatsApp)
        - credentials (JSONB para credenciais criptografadas)
        - created_at, updated_at
    - queues:
        - id (PK)
        - account_id (FK para accounts.id)

        - name
        - description
        - channel_id (FK para channels.id, opcional se a fila for multicanal)
        - created_at, updated_at
    - user_channel_permissions:
        - user_id (FK para users.id)
        - channel_id (FK para channels.id)
        - permission_type (e.g., 'read', 'write', 'manage')
        - (Chave composta: user_id, channel_id, permission_type)
    - user_queue_permissions:
        - user_id (FK para users.id)
        - queue_id (FK para queues.id)
        - permission_type (e.g., 'assign', 'view_conversations')
        - (Chave composta: user_id, queue_id, permission_type)
    - contacts:
        - id (PK)
        - account_id (FK para accounts.id)
        - name
        - phone_number (se for WhatsApp/SMS)
        - email (se for e-mail)
        - external_id (ID do contato no provedor externo, ex: ID do WhatsApp)
        - metadata (JSONB para dados adicionais do contato)
        - created_at, updated_at
    - conversations:
        - id (PK)
        - account_id (FK para accounts.id)
        - channel_id (FK para channels.id)
        - contact_id (FK para contacts.id)
        - status (e.g., 'open', 'pending', 'closed', 'in_progress')
        - assigned_agent_id (FK para users.id, se atribuído)
        - queue_id (FK para queues.id, se aplicável)
        - last_message_at
        - created_at, updated_at
    - messages:
        - id (PK)
        - conversation_id (FK para conversations.id)
        - sender_type (e.g., 'contact', 'agent')
        - sender_id (FK para contacts.id ou users.id dependendo do sender_type)
        - text

        - timestamp
        - message_type (e.g., 'text', 'image', 'file')
        - attachment_id (FK para attachments.id, opcional)
        - is_read
        - created_at
    - automation_flows:
        - id (PK)
        - account_id (FK para accounts.id)
        - name
        - definition (JSONB para a estrutura completa do fluxo, etapas e regras)
        - is_active
        - created_at, updated_at
    - flow_steps: (Pode ser parte do definition em automation_flows ou uma tabela separada para granularidade)
        - id (PK)
        - flow_id (FK para automation_flows.id)
        - step_type (e.g., 'message', 'question', 'redirect_to_agent')
        - config (JSONB para configurações da etapa)
        - order
    - attachments:
        - id (PK)
        - message_id (FK para messages.id)
        - file_name
        - file_type (MIME type)
        - file_size
        - storage_url (URL para o arquivo no serviço de armazenamento em nuvem)
        - created_at
    - audits:
        - id (PK)
        - account_id (FK para accounts.id, se aplicável)
        - user_id (FK para users.id, se aplicável)
        - event_type (e.g., 'login', 'message_sent', 'permission_changed', 'channel_connected')
        - entity_type (e.g., 'user', 'channel', 'conversation')
        - entity_id (ID da entidade afetada)
        - details (JSONB para detalhes adicionais do evento)
        - timestamp

2.4. Comunicação em Tempo Real (WebSockets / Socket.IO)

- **Tecnologia:** Socket.IO (implementado no NestJS backend e consumido pelo Next.js frontend).
- **Função:** Habilita a troca de mensagens instantâneas no chat, atualizações de status de conversas e notificações em tempo real.

2.5. Integração com IA (Gemini API)

- **Tecnologia:** Gemini API (via chamadas HTTP do NestJS backend).
- **Função:** O módulo de IA no backend fará chamadas à Gemini API, enviando o contexto da conversa e recebendo sugestões de respostas para os atendentes.

3. Fluxos de Interação Chave

3.1. Conexão Multi-WhatsApp

1. **Frontend:** Usuário clica em "Conectar WhatsApp" no módulo de Gestão de Canais.
2. **Frontend -> Backend:** Requisição para o endpoint /channels/whatsapp para iniciar a conexão e gerar QR Code.
3. **Backend (Serviço WhatsApp):** Inicia uma nova sessão de WhatsApp, gerando um QR Code e persistindo o canal na tabela channels.
4. **Backend -> Frontend:** Envia o QR Code (como imagem base64 ou URL temporária).
5. **Frontend:** Exibe o QR Code para o usuário escanear.
6. **Backend (Serviço WhatsApp):** Monitora o status da sessão até a conexão ser estabelecida e atualiza o status do canal na tabela channels.
7. **Backend -> Frontend (via WebSocket):** Notifica o frontend sobre a conexão bem-sucedida.

3.2. Chat com Sugestões de IA

1. **Contato Externo:** Envia mensagem para um canal (WhatsApp, e-mail, Facebook).
2. **Webhook (Backend):** Recebe a mensagem (verificando autenticidade) e a persiste na tabela messages, associando-a a um contact (criando ou atualizando se necessário) e uma conversation.
3. **Backend (via WebSocket):** Emite a nova mensagem para o frontend do atendente.
4. **Frontend:** Exibe a nova mensagem no chat.
5. **Frontend -> Backend (requisição assíncrona):** Solicita sugestões de IA para a mensagem recebida.
6. **Backend (Módulo de IA):** Envia o contexto da conversa para a Gemini API.
7. **Gemini API:** Retorna sugestões de resposta.
8. **Backend -> Frontend:** Envia as sugestões.

9. **Frontend:** Exibe as sugestões para o atendente.
10. **Atendente:** Envia sua resposta via interface do chat.
11. **Frontend -> Backend (via WebSocket):** Envia a resposta do atendente.
12. **Backend (Serviço do Canal):** Envia a resposta para o contato externo através da API do canal e persiste a resposta na tabela messages.

3.3. Execução de Fluxo de Automação

1. **Contato Externo:** Inicia uma conversa.
2. **Serviço do Canal (Backend):** Identifica que um fluxo de automação está configurado para este canal/contato, consultando automation_flows.
3. **Backend (Módulo de Automação):** Começa a executar o fluxo (utilizando o Interpretador de Fluxos), enviando mensagens e coletando respostas do contato.
4. **Backend (Módulo de Automação):** Se o fluxo atingir uma etapa de "direcionar para atendente" (redirect_to_agent), ele marca a conversa como "pendente de atendimento humano" no campo status da tabela conversations.
5. **Backend (via WebSocket):** Notifica o frontend sobre a nova conversa pendente.
6. **Frontend:** Atendente é notificado e pode assumir a conversa.

4. Considerações de Segurança

- **Autenticação:** JWT (JSON Web Tokens) para autenticação segura entre frontend e backend.
- **Autorização:** Controle de acesso baseado em papéis (RBAC) através das tabelas roles, custom_roles, user_roles, user_custom_roles, user_channel_permissions, user_queue_permissions. Middleware de checagem de permissões no backend. Acesso limitado a recursos por conta.
- **Criptografia:** Senhas de usuários armazenadas com hash (ex: bcrypt). Credenciais de canais criptografadas. Dados sensíveis em trânsito protegidos com HTTPS/WSS.
- **Validação de Entrada:** Todas as entradas de dados devem ser rigorosamente validadas no backend para prevenir ataques.
- **Webhooks:** Endpoint seguro para webhooks com verificação de autenticidade (assinaturas, tokens).
- **Logs e Auditoria:** Tabela audits para rastrear ações importantes no sistema, auxiliando na segurança e compliance.

5. Implantação

- **Frontend:** Pode ser implantado em plataformas como Vercel, Netlify ou como um build estático em um CDN.
- **Backend:** Pode ser implantado em serviços de nuvem como AWS EC2/ECS, Google Cloud Run/App Engine ou DigitalOcean Droplets, utilizando Docker para

conteinerização.
- **Banco de Dados:** Serviços gerenciados de PostgreSQL (ex: AWS RDS, Google Cloud SQL) para alta disponibilidade e escalabilidade.

Esta arquitetura fornece uma base sólida para o desenvolvimento do seu dashboard de atendimento, permitindo escalabilidade, manutenibilidade e a integração de todas as funcionalidades desejadas, com base no backlog fornecido.