agentic-ivr - Detalhamento de Épicos

Visão Geral

Este documento fornece o detalhamento completo de épicos e histórias para o agentic-ivr, decompondo os requisitos do PRD e da Arquitetura em histórias implementáveis. Não existe documento de Design UX — este é um serviço de ponte de áudio exclusivamente backend.

Inventário de Requisitos

Requisitos Funcionais

50 FRs distribuídos em 10 capacidades (FR15 absorvido em FR13+FR31 conforme histórico de elicitação do PRD)

Capacidade 1: Ingestão de Chamadas & Gerenciamento de Ciclo de Vida (6 FRs)

- FR1: O sistema DEVE receber eventos `IncomingCall` do Azure Event Grid via webhook HTTPS e iniciar a sequência de atendimento da chamada dentro do timeout de toque configurado (C-03).
- FR2: O sistema DEVE atender chamadas recebidas usando ACS Call Automation com streaming de mídia bidirecional habilitado (PCM 24K mono).
- FR3: O sistema DEVE tratar eventos de callback mid-call do ACS (`CallConnected`, `CallDisconnected`, `PlayFailed`, `PlayCompleted`) e atualizar o estado interno da chamada adequadamente.
- FR4: O sistema DEVE realizar teardown gracioso de todos os recursos (ambas as conexões WebSocket, estado da chamada, métricas) quando uma chamada termina, independentemente de qual lado inicia a desconexão.
- FR5: O sistema DEVE deduplicar eventos de callback do ACS usando um cache com chave `callId` + `eventType` + `correlationId` com TTL configurável (C-56), garantindo processamento idempotente.
- FR6: Quando a chamada se aproximar da duração máxima (C-29 menos C-30 segundos restantes), o sistema DEVE injetar uma mensagem de contexto ao Voice Live solicitando que a IA encerre a conversa naturalmente, e forçar uma desconexão no tempo máximo caso a chamada não termine voluntariamente.

Capacidade 2: Comunicação de Áudio em Tempo Real (4 FRs)

- FR7: O sistema DEVE aceitar conexões WebSocket de entrada do ACS no caminho versionado `/ws/v1` e fazer parse dos frames binários recebidos em pacotes `AudioMetadata` e `AudioData`.
- FR8: O sistema DEVE enviar áudio de saída para o ACS via a mesma conexão WebSocket usando o formato de dados de streaming de saída do SDK ACS.
- FR9: O sistema DEVE encaminhar áudio bidirecionalmente entre ACS e Voice Live em tempo real sem buffering, inspeção ou transformação do conteúdo de áudio (apenas tradução de protocolo).
- FR10: O sistema DEVE impor um ciclo de vida vinculado entre as conexões WebSocket do ACS e do Voice Live — o fechamento de qualquer um dos lados DEVE disparar teardown gracioso de ambos dentro do timeout configurado (C-31).

Capacidade 3: Integração com IA Conversacional (5 FRs)

- FR11: O sistema DEVE estabelecer uma conexão WebSocket de saída para o endpoint da API Voice Live (C-07) com a versão de API configurada (C-08) dentro do timeout de conexão (C-10).
- FR12: O sistema DEVE autenticar-se na API Voice Live usando credenciais de identidade gerenciada, sem chaves de API no código ou configuração.
- FR13: O sistema DEVE enviar um evento `session.update` ao Voice Live após a conexão, configurando a sessão com o prompt de sistema externalizado (C-12), modelo de voz (C-13), definições de ferramentas carregadas do arquivo de definições externalizado (C-60), e configurações de detecção de turno — incluindo tipo de VAD, threshold, duração de silêncio, supressão de ruído e cancelamento de eco (C-14 a C-18). Se o Voice Live rejeitar a sessão, o sistema DEVE registrar em log a rejeição em nível WARN incluindo `correlationId`, código/mensagem de erro e tamanho do prompt de sistema.
- FR14: Ao receber `input_audio_buffer.speech_started` do Voice Live, o sistema DEVE imediatamente (i) parar de encaminhar frames de áudio do Voice Live para o ACS e (ii) cancelar qualquer operação Play do ACS em andamento, tratando ambas as ações como operações idempotentes seguras para invocar redundantemente.
- FR16: O sistema DEVE suportar a injeção de mensagens de nível de sistema ao Voice Live durante a sessão (ex.: notificação de encerramento por duração máxima) quando acionado por eventos de ciclo de vida.

Capacidade 4: Resiliência de Conexão & Recuperação (6 FRs)

- FR17: O sistema DEVE implementar heartbeat/keepalive em ambas as conexões WebSocket e detectar obsolescência de conexão dentro do timeout de inatividade configurado (C-05, C-11). Ao detectar obsolescência, o sistema DEVE forçar o fechamento da conexão WebSocket obsoleta, disparando o teardown de ciclo de vida vinculado conforme FR10.
- FR18: O sistema DEVE implementar um circuit breaker nas conexões Voice Live que abre para estado OPEN após N falhas consecutivas de conexão (C-22), impedindo novas chamadas de tentarem conexões fadadas ao fracasso.
- FR19: O circuit breaker DEVE transicionar pelos estados CLOSED → OPEN → HALF-OPEN → CLOSED, com atraso half-open configurável (C-23) e threshold de sucesso para recuperação (C-24).
- FR20: Se o ACS reconectar dentro da janela de reconexão configurada (C-28), o sistema DEVE tentar resumir a sessão Voice Live existente em vez de iniciar uma nova. Se a retomada falhar, o sistema DEVE recorrer ao comportamento do FR21.
- FR21: Quando o circuit breaker estiver OPEN ou a retomada de sessão falhar, o sistema DEVE reproduzir o prompt de áudio de fallback apropriado para o chamador antes de desconectar graciosamente.
- FR46: O sistema DEVE detectar quando uma resposta de áudio do Voice Live trava no meio da transmissão — definido como receber eventos `response.audio.delta` seguidos de nenhum evento de áudio adicional e nenhum `response.done` dentro de C-55 (padrão 5000ms). Na detecção de stall, o sistema DEVE registrar em log um WARN, reproduzir o prompt de desculpas (C-37) e iniciar teardown gracioso.

Capacidade 5: Segurança & Autenticação (6 FRs)

- FR22: O sistema DEVE validar tokens JWT no cabeçalho `Authorization` durante o handshake de upgrade WebSocket, rejeitando conexões com HTTP 401 antes do upgrade se o token for inválido, expirado ou ausente.

- FR23: O sistema DEVE autenticar-se no ACS Call Automation usando credenciais de identidade gerenciada, sem strings de conexão ou chaves de API no código, configuração ou secret stores.
- FR24: O sistema DEVE autenticar-se na API Voice Live usando credenciais de identidade gerenciada. Pré-requisito de infraestrutura: role RBAC `Cognitive Services User` no recurso Foundry.
- FR25: O sistema DEVE validar tokens bearer Entra ID em requisições de webhook do Event Grid e completar o handshake `SubscriptionValidationEvent` durante o registro de assinatura.
- FR26: Todas as conexões WebSocket (tanto do lado ACS quanto do lado Voice Live) DEVEM usar protocolo `wss://` exclusivamente; o sistema DEVE rejeitar ou nunca iniciar conexões `ws://`.
- FR27: O sistema DEVE implementar uma abstração testável de saúde de credenciais que reporta o status do token em cache sem disparar refresh de token ativo, usada pela probe de prontidão.

Capacidade 6: Observabilidade Operacional (5 FRs)

- FR28: O sistema DEVE emitir todos os eventos de log em formato JSON estruturado, com cada entrada incluindo campos `correlationId`, `timestamp`, `level` e `component`. Quando um contexto de trace OTel estiver ativo, as entradas DEVEM incluir `traceId` e `spanId`.
- FR29: O sistema DEVE emitir spans de rastreamento distribuído para as 4 operações de ciclo de vida de chamada definidas (`ivr.call`, `ivr.call.answer`, `ivr.call.voicelive.connect`, `ivr.call.audio.bridge`).
- FR30: O sistema DEVE expor health probes de três camadas: startup (primeira aquisição de token de identidade gerenciada), liveness (contexto Spring ativo), readiness (status de credencial em cache + resolução DNS do Voice Live).
- FR31: Todos os parâmetros configuráveis (conforme o Catálogo de Configuração) DEVEM ser externalizados via propriedades de configuração Spring com restrições de validação, suportando overrides por ambiente via Spring profiles (`test`, `local`, `prod`).
- FR32: O sistema DEVE expor métricas Micrometer para todos os 14 nomes de métrica definidos (prefixados com `ivr.*`) cobrindo chamadas ativas, distribuições de latência, estado do circuit breaker, contagens de rejeição de segurança e contadores de encaminhamento de áudio.

Capacidade 7: Admissão de Chamadas & Casos de Borda de Ciclo de Vida (6 FRs)

- FR33: O sistema DEVE rastrear chamadas concorrentes ativas locais da instância e rejeitar chamadas recebidas com um prompt falado de ocupado (C-38) quando a contagem de chamadas ativas locais da instância exceder o threshold de admissão por instância (C-26).
- FR34: Em SIGTERM, o sistema DEVE marcar a probe de readiness como unhealthy imediatamente, enviar uma notificação de encerramento para sessões Voice Live em andamento, e permitir que chamadas ativas completem dentro do timeout de drenagem (C-34) antes de encerrar conexões forçosamente.
- FR35: O sistema DEVE reproduzir prompts de áudio pré-gravados via ACS Play action para: (a) tom de conforto se conexão VL exceder C-33, (b) saudação de fallback se nenhum áudio de IA dentro de C-32, (c) prompt de desculpas em desconexão inesperada do VL, (d) prompt de ocupado em rejeição de admissão, (e) prompt de serviço indisponível quando circuit breaker estiver aberto.
- FR42: O sistema DEVE realizar varredura periódica de sessões de chamada que excedam duração máxima mais buffer (intervalo C-52) e encerrar forçosamente sessões órfãs onde nem o fechamento de WebSocket nem o callback chegaram.
- FR43: O sistema DEVE encerrar conexões WebSocket que não recebam nenhum frame de áudio dentro do timeout de inatividade (C-54) após upgrade bem-sucedido.

- FR47: Quando uma ação Play do ACS falhar (callback `PlayFailed`), o sistema NÃO DEVE tentar novamente ou reproduzir um prompt substituto. Ele DEVE registrar em log `PlayFailed` em nível ERROR com `correlationId`, identificador do prompt e razão de falha do ACS, e então prosseguir diretamente para encerramento gracioso da chamada.

Capacidade 8: Segurança em Profundidade (5 FRs)

- FR36: O sistema DEVE gerar um token criptograficamente aleatório com comprimento mínimo (C-50) e incorporá-lo em cada URL de callback registrada com o ACS, validando o token em cada requisição de callback.
- FR37: O sistema DEVE validar o `eventTime` de cada evento do Event Grid e rejeitar eventos mais antigos que C-45 para prevenir ataques de replay.
- FR38: O sistema DEVE impor limitação de taxa por IP em tentativas de handshake WebSocket, rejeitando conexões que excedam C-46 por janela C-47 com HTTP 429.
- FR39: O sistema DEVE vincular o Spring Boot Actuator a uma porta de gerenciamento separada (C-48), expondo apenas endpoints de health/liveness/readiness/startup, retornando 404 para todos os outros caminhos do actuator.
- FR40: O sistema DEVE retornar corpos JSON genéricos de erro em todas as respostas de erro HTTP, sem stack traces, nomes de classe ou detalhes específicos do framework.

Capacidade 9: Redes de Segurança Operacional (3 FRs)

- FR41: O sistema DEVE sondar periodicamente a conectividade da API Voice Live (intervalo C-25) e acionar preventivamente o circuit breaker se as sondas falharem.
- FR44: O sistema DEVE processar lotes de eventos do Event Grid individualmente, retornando HTTP 200 para o lote mesmo se eventos individuais falharem no processamento, registrando em log falhas por evento.
- FR45: O sistema DEVE usar deserialização JSON leniente para eventos do Voice Live (ignorando campos desconhecidos) e incrementar um contador de métricas de erro de parse nas falhas de deserialização.

Capacidade 10: Framework de Chamadas de Ferramentas (4 FRs)

- FR48: Ao receber um evento `tool_call` do Voice Live, o sistema DEVE despachar uma requisição HTTPS para a URL base do gateway de API configurado (C-57) usando `java.net.http.HttpClient` em uma virtual thread, autenticando com `DefaultAzureCredential` (escopo C-57b). A requisição DEVE incluir o nome da chamada de ferramenta, argumentos e `correlationId`.
- FR49: Ao receber uma resposta HTTP bem-sucedida do gateway de API, o sistema DEVE encaminhar o resultado ao Voice Live como um evento `tool_call_output` contendo o ID da chamada de ferramenta e o payload da resposta.
- FR50: Se o gateway de API não responder dentro de C-58, ou retornar um erro HTTP (4xx/5xx), o sistema DEVE retornar um evento `tool_call_output` ao Voice Live com um payload de erro, registrar em log a falha em nível WARN e incrementar `ivr.tool_call.errors`. O sistema NÃO DEVE encerrar a chamada devido a uma falha de chamada de ferramenta.
- FR51: Na inicialização da sessão, o sistema DEVE carregar definições de ferramentas de C-60, validar a estrutura JSON e incluir as definições de ferramentas no evento `session.update` enviado ao Voice Live (FR13). Se o arquivo de definições estiver ausente ou contiver JSON inválido, o sistema DEVE registrar em log em nível ERROR e iniciar a sessão sem definições de ferramentas (modo degradado).

Requisitos Não-Funcionais

47 NFRs distribuídos em 6 categorias

Categoria 1: Performance (10 NFRs)

- NFR-P1: Tempo até Primeiro Áudio — tempo decorrido de `CallConnected` até primeiro byte de áudio enviado ao ACS. Meta: < 3s p95. Cache JWKS DEVE ser pré-aquecido durante a inicialização.
- NFR-P2: Latência de Resposta da IA — tempo decorrido do fim de fala do VAD até primeiro `response.audio.delta` do Voice Live. Meta: < 1s p95.
- NFR-P3: Latência de Barge-In — tempo decorrido de `speech_started` recebido até `StopAudio` enviado ao ACS. Meta: < 100ms p95.
- NFR-P4: Latência de Encaminhamento de Áudio — travessia de um frame de áudio pela ponte. Meta: < 50ms p99.
- NFR-P5: Latência de Atendimento de Chamada — `IncomingCall` recebido até resposta de `answerCall()`. Meta: < 3s p95.
- NFR-P6: Latência de Conexão Voice Live — handshake WebSocket até `session.created`. Meta: ≤ 2s p95, ≤ 3s p99.
- NFR-P7: Memória por Chamada — heap máximo por chamada ativa. Meta: ≤ 200 KB por chamada.
- NFR-P8: Encaminhamento de Áudio Zero-Copy — sem cópias intermediárias de buffer (tradução de protocolo permitida).
- NFR-P9: Tempo de Inicialização da Aplicação — lançamento da JVM até probe de readiness 200. Meta: ≤ 30s (JVM fria). Aquisição de credencial limitada a 15s.
- NFR-P10: Latência Round-Trip de Chamada de Ferramenta — `tool_call` recebido até `tool_call_output` enviado. Meta: ≤ 5s p95.

Categoria 2: Confiabilidade (14 NFRs)

- NFR-R1: Disponibilidade do Serviço — ≥ 99,9% de uptime.
- NFR-R2: Teardown de Ciclo de Vida Vinculado — WebSocket par fechado + recursos liberados ≤ 3s. Aplica-se apenas a fechamentos não-reconectáveis (códigos 1000, 1001).
- NFR-R3: Ativação do Circuit Breaker — aciona exatamente em C-22 falhas consecutivas.
- NFR-R4: Drenagem de Shutdown Gracioso — 100% das chamadas em andamento drenadas em SIGTERM dentro de C-34.
- NFR-R5: Zero Condições de Corrida — transições de estado concorrentes não produzem estado inconsistente, vazamentos de recursos ou teardown duplicado.
- NFR-R6: Deduplicação de Callback — callbacks duplicados do ACS processados exatamente uma vez.
- NFR-R7: Zero Sessões Órfãs — nenhuma sessão Voice Live permanece após fim da chamada + timeout.
- NFR-R8: Taxa de Conclusão de Chamadas — ≥ 99% encerramentos graciosos / total atendidas.
- NFR-R9: Latência de Detecção de Heartbeat — obsolescência detectada ≤ 3s desde último heartbeat.
- NFR-R10: Fail-Fast de Configuração — aplicação falha ao iniciar dentro de 5s se configuração obrigatória estiver ausente/inválida.
- NFR-R11: Completude de Sinais de Observabilidade — ≥ 99% de todos os eventos de log + métricas aplicáveis emitidos por ciclo de vida de chamada.
- NFR-R12: Conformidade da Janela de Reconexão — sessão VL mantida por C-28 (5s) em fechamento anormal do ACS (código 1006), depois teardown. Fechamentos normais disparam teardown R2 imediato.

- NFR-R13: Precisão de Detecção de Stall — < 1% falsos positivos na detecção de stall em meio de transmissão.
- NFR-R14: Tratamento de Timeout de Chamada de Ferramenta — 100% cobertura de tratamento de timeout; 0 encerramentos de chamada por falhas de chamadas de ferramentas.

Categoria 3: Escalabilidade (6 NFRs)

- NFR-S1: Capacidade de Chamadas Concorrentes — 50 chamadas por instância sem degradação de latência.
- NFR-S2: Estabilidade sob Carga Sustentada — 50 chamadas × 30 min: zero OOM, zero starvation de threads.
- NFR-S3: Limite de Memória por Instância — ≤ 1 GB de memória total do container em capacidade máxima.
- NFR-S4: Escalabilidade Horizontal Stateless — zero estado mutável compartilhado entre instâncias.
- NFR-S5: Isolamento de Controle de Admissão — rejeição é local à instância, sem impacto entre instâncias.
- NFR-S6: Largura de Banda de Rede — ≤ 10 MB/s por instância em capacidade máxima.

Categoria 4: Segurança (10 NFRs)

- NFR-SEC1: Cobertura de Autenticação WebSocket — 100% validação JWT em upgrades de `/ws/v1`.
- NFR-SEC2: Zero Segredos em Código/Config — todos os segredos resolvidos em runtime via identidade gerenciada ou K8s CSI.
- NFR-SEC3: Imposição de WSS-Only — zero conexões `ws://`.
- NFR-SEC4: Zero CVEs Críticos/Altos — nenhum CVSS ≥ 7.0 na árvore de dependências de produção.
- NFR-SEC5: Zero Vazamento de Informação — sem stack traces, nomes de classe ou detalhes internos em respostas externas.
- NFR-SEC6: Isolamento do Actuator — porta de gerenciamento (C-48), apenas endpoints de health expostos.
- NFR-SEC7: Proteção contra Replay de Eventos — rejeitar eventos com `eventTime` mais antigo que C-45.
- NFR-SEC8: Zero Áudio em Saída de Log — nenhum byte de áudio raw em nenhum nível de log.
- NFR-SEC9: Zero Bytes de Áudio em Repouso — áudio existe apenas na heap da JVM durante encaminhamento ativo.
- NFR-SEC10: Zero Falsos Positivos do Rate-Limiter para Tráfego ACS — faixas de IP do ACS isentas de limitação de taxa por IP.

Categoria 5: Confiabilidade de Integração (7 NFRs)

- NFR-I1: Cobertura de Prompts de Fallback — 100% cobertura para todos os 5 modos de falha (FR35a–e).
- NFR-I2: Sem Tipos Reativos — zero `Mono`, `Flux`, `Publisher` no código de produção.
- NFR-I3: Deserialização Leniente do Voice Live — campos desconhecidos ignorados, falhas de parse incrementam contador.
- NFR-I4: Alterações de Versão do Voice Live Apenas por Config — atualização de versão = deploy apenas de configuração.

- NFR-I5: Suíte de Testes Sem Nuvem — `./mvnw test` passa com zero variáveis de ambiente Azure, zero chamadas de rede. Emissor JWT de teste in-process exercita o caminho completo de validação.
- NFR-I6: Velocidade de Onboarding de Desenvolvedor — git clone até primeira chamada ≤ 30 minutos (recursos Azure pré-provisionados).
- NFR-I7: Build Determinístico com Único Comando — `./mvnw clean package` produz JAR executável com zero etapas manuais.

Requisitos Adicionais

Da Arquitetura: Template Inicial (Impacta Épico 1 História 1)

- **Projeto greenfield Spring Initializr** (Opção C selecionada) com: starters `web`, `websocket`, `actuator`, `validation`, `configuration-processor`
- Baseline Spring Boot 4.0.x (ADR-8)
- `spring.threads.virtual.enabled=true` DEVE estar presente desde o primeiro dia
- Registro de bean `ServerEndpointExporter` obrigatório (Spring Boot não registra automaticamente endpoints JSR 356)
- Layout de pacotes: `com.sofia.ivr.{config,webhook,websocket,bridge,toolcall,model,exception}`

Da Arquitetura: Restrições Técnicas (17)

- TC-1: Java 21 LTS, Spring Boot 4.x, Spring MVC + virtual threads
- TC-2: JSR 356 `@ServerEndpoint` para servidor WebSocket ACS
- TC-3: `java.net.http.WebSocket` para cliente Voice Live
- TC-4: SDK `azure-communication-callautomation` para controle de chamada
- TC-5: `DefaultAzureCredential` para toda autenticação Azure
- TC-6: Sem chaves de API em nenhum lugar — apenas identidade gerenciada
- TC-7: PCM 24K mono — sem transcodificação
- TC-8: Caminho de áudio WebSocket direto — sem gateways HTTP
- TC-9: Event Grid → Serviço direto (sem APIM)
- TC-10: Callbacks ACS → Serviço direto (sem APIM)
- TC-11: APIM apenas para saída para backends
- TC-12: Zero persistência de áudio — restrição de dados biométricos
- TC-13: Serviço Spring Boot único — sem decomposição em microsserviços
- TC-14: Deploy em AKS, região Brazil South
- TC-15: Guarda contra pinning de virtual thread — sem blocos `synchronized`
- TC-16: Serialização de envio WebSocket — um remetente por conexão
- TC-17: Pureza do caminho de áudio — zero lógica de negócio no caminho de encaminhamento de áudio

Da Arquitetura: Preocupações Transversais (7)

- CC-1: Correlação — passagem explícita de `CallContext` com `correlationId` (ID de chamada ACS)
- CC-2: Máquina de Estados da Chamada — tabela de transição `EnumMap`, 4 estados comportamentais: INITIALIZING → STREAMING → TERMINATING → TERMINATED

- CC-3: Multiplexação de Auth — 3 fluxos de validação JWT (Entra ID, webhook ACS, WebSocket ACS) + refresh preventivo de token a 75% do tempo de vida
- CC-4: Hierarquia de Degradação Graciosa — comportamento de fallback ordenado por tipo de falha
- CC-5: Validação de Configuração — `@ConfigurationProperties` com restrições de validação, fail-fast no boot
- CC-6: Determinismo de Limpeza de Recursos — `try-finally` em todos os caminhos WebSocket
- CC-7: Isolamento de Falha Correlacionada — falha de uma chamada não pode cascatear para outras

Da Arquitetura: Decisões-Chave & ADRs

- ADR-7: Manter transporte HTTP Netty padrão do Azure SDK (sem troca para JDK HttpClient)
- ADR-8: Baseline Spring Boot 4.0.x
- ADR-9: Autoconfig OTel Spring Boot sobre Java Agent
- ADR-10: ConcurrentHashMap sobre Redis para estado de chamada
- ADR-11: Nimbus JOSE+JWT direto para todos os 3 caminhos de validação JWT (sem Spring Security)
- Decisão 1: Store de estado de chamada em memória (`ConcurrentHashMap<String, CallContext>`)
- Decisão 3: Handshake de SubscriptionValidation síncrono do Event Grid
- Decisão 4: Interfaces seladas + records para esquema de eventos Voice Live
- Decisão 5: RestClient + circuit breaker Resilience4j para despacho de chamadas de ferramentas
- Decisão 6: Hierarquia de exceções selada com tradução específica por superfície
- Decisão 7: Dockerfile multi-estágio com extração de JAR em camadas
- Decisão 9: Externalização de configuração via variáveis de ambiente + relaxed binding Spring
- Decisão 10: Logging JSON estruturado com troca por profile

Da Arquitetura: Regras Obrigatórias (8)

- M-1: Sem `Mono`, `Flux` ou tipos reativos em nenhum lugar
- M-2: Sem segredos, URLs ou strings de conexão hardcoded
- M-3: Sem `Thread.sleep()` para delays
- M-4: Sem blocos `synchronized` (pinning de virtual thread)
- M-5: Sem pools de threads personalizados para trabalho de I/O
- M-6: Sem áudio WebSocket através de middleware HTTP/APIM
- M-7: Todos os endpoints REST públicos devem validar JWT
- M-8: Sem `catch (Exception e) {}` — engolir silenciosamente é proibido

Da Arquitetura: Dependências

- `spring-boot-starter-web` — Tomcat, Spring MVC, Jackson
- `spring-boot-starter-websocket` — Suporte WebSocket
- `spring-boot-starter-actuator` — Health, métricas Micrometer
- `spring-boot-starter-validation` — Validação de beans
- `spring-boot-configuration-processor` — Metadados de `@ConfigurationProperties`
- `azure-communication-callautomation` — SDK ACS Call Automation
- `azure-identity` — `DefaultAzureCredential`
- `azure-sdk-bom` — Gerenciamento de versão do Azure SDK
- `opentelemetry-spring-boot-starter` — Autoconfig OTel
- `resilience4j-spring-boot3` — Circuit breaker

- `logstash-logback-encoder` — Logging JSON estruturado
- `spring-boot-starter-test` — JUnit 5, Mockito, AssertJ

Da Arquitetura: Estrutura do Projeto

- ~60 arquivos em 7 pacotes de funcionalidade + resources
- Diretório de testes completo espelhando pacotes fonte
- Testes de integração: `WebhookIntegrationTest`, `WebSocketIntegrationTest`, `ToolCallIntegrationTest`
- Fixtures de teste: eventos JSON, JWKS de teste, binário de frame de áudio, stubs WireMock
- Definições de ferramentas: `get-user-data.json`, `send-invoice-registered-email.json`, `send-invoice-provided-email.json`

Do PRD: Requisitos de Segurança (18 SEC-REQs)

- SEC-REQ-1 (P0): Validação JWT em todas as conexões de entrada
- SEC-REQ-2 (P0): Auth Entra ID em webhooks do Event Grid
- SEC-REQ-3 (P0): Validação JWT ACS em callbacks
- SEC-REQ-4 (P0): Validação de token na URL de callback
- SEC-REQ-5 (P0): Zero logging / persistência de áudio
- SEC-REQ-9 (P0): Isolamento de endpoints do Actuator
- SEC-REQ-10 (P0): Endurecimento de respostas de erro
- SEC-REQ-11 (P0): Identidade gerenciada em todos os lugares
- SEC-REQ-14 (P0): Proteção contra replay de eventos
- SEC-REQ-15 (P0): Rate limiting de WebSocket
- SEC-REQ-6 (P1): Mascaramento de PII em logs
- SEC-REQ-7 (P1): Trilha de auditoria de consentimento LGPD
- SEC-REQ-8 (P1): Direitos de titulares de dados LGPD
- SEC-REQ-12 (P1): Varredura de vulnerabilidades de dependências
- SEC-REQ-13 (P1): Varredura de imagem de container
- SEC-REQ-16 (P1): Imposição de TLS 1.2+
- SEC-REQ-17 (P1): Segmentação de rede
- SEC-REQ-18 (P0): Auth de chamada de ferramenta via identidade gerenciada

Do PRD: Requisitos de UX (Experiência do Chamador — 11 UX-REQs)

- UX-REQ-1: Tolerância a silêncio ≤ 3s, depois tom de conforto
- UX-REQ-2: Voz de IA consistente durante toda a chamada
- UX-REQ-3: Pedir desculpas antes de desconectar em falhas
- UX-REQ-4: Tempo até primeiro áudio < 3s
- UX-REQ-5: Suporte a barge-in (interromper IA)
- UX-REQ-6: Sem cortes abruptos no meio de frases
- UX-REQ-7: Latência de resposta da IA < 1s após fala
- UX-REQ-8: Barge-in interrompe áudio da IA em 100ms
- UX-REQ-9: Notificação de encerramento por duração máxima
- UX-REQ-10: Mensagem graciosa com circuit-breaker aberto
- UX-REQ-11: Mensagem falada de rejeição de admissão

Do PRD: Tiers de Prioridade

- **P0 (Sprint MVP de 5 Dias):** 30 itens em plano dia a dia — ciclo de vida básico de chamada, ponte de áudio, integração Voice Live, segurança, observabilidade, chamadas de ferramentas
- **P1 (Crescimento):** Mascaramento de PII, conformidade LGPD, transferência de chamada, métricas avançadas
- **P2 (Escala):** Autoscaling KEDA, multi-região, retomada de sessão

Mapa de Cobertura de FRs

- FR1: Épico 1 — Receber IncomingCall do Event Grid, iniciar sequência de atendimento
- FR2: Épico 1 — Atender chamadas recebidas via ACS com streaming de mídia bidirecional
- FR3: Épico 1 — Tratar eventos de callback mid-call do ACS, atualizar estado da chamada
- FR4: Épico 4 — Teardown gracioso de todos os recursos no fim da chamada
- FR5: Épico 7 — Deduplicar eventos de callback do ACS
- FR6: Épico 3 — Notificação de encerramento por duração máxima e desconexão forçada
- FR7: Épico 1 — Aceitar conexões WebSocket do ACS em /ws/v1, fazer parse de frames de áudio
- FR8: Épico 2 — Enviar áudio de saída para ACS via WebSocket
- FR9: Épico 2 — Encaminhar áudio bidirecionalmente entre ACS e Voice Live (zero-copy)
- FR10: Épico 2 — Ciclo de vida vinculado entre WebSockets do ACS e Voice Live
- FR11: Épico 2 — Estabelecer WebSocket de saída para API Voice Live
- FR12: Épico 2 — Autenticar no Voice Live via identidade gerenciada
- FR13: Épico 2 — Enviar session.update com prompt de sistema, modelo de voz, defs de ferramentas, config de VAD
- FR14: Épico 3 — Barge-in: parar encaminhamento de áudio VL + cancelar Play ACS
- FR16: Épico 3 — Injetar mensagens de nível de sistema ao Voice Live durante sessão
- FR17: Épico 4 — Heartbeat/keepalive e detecção de obsolescência em ambos WebSockets
- FR18: Épico 4 — Circuit breaker em conexões Voice Live
- FR19: Épico 4 — Transições de estado do circuit breaker (CLOSED→OPEN→HALF-OPEN→CLOSED)
- FR20: Épico 4 — Janela de reconexão ACS com retomada de sessão Voice Live
- FR21: Épico 4 — Áudio de fallback quando circuit breaker OPEN ou retomada falha
- FR22: Épico 6 — Validação JWT no handshake de upgrade WebSocket
- FR23: Épico 1 — Auth de identidade gerenciada do ACS Call Automation
- FR24: Épico 6 — Auth de identidade gerenciada do Voice Live + pré-requisito RBAC
- FR25: Épico 1 — Validação de token Entra ID do Event Grid + handshake SubscriptionValidation
- FR26: Épico 6 — Imposição de WSS-only
- FR27: Épico 6 — Abstração testável de saúde de credenciais
- FR28: Épico 1 — Logging JSON estruturado com correlationId, traceId, spanId
- FR29: Épico 1 — Spans de rastreamento distribuído para operações de ciclo de vida de chamada
- FR30: Épico 1 — Health probes de três camadas (startup, liveness, readiness)
- FR31: Épico 1 — Externalização de configuração com propriedades Spring + validação
- FR32: Épico 1 — Métricas Micrometer para todos os 14 nomes de métrica
- FR33: Épico 7 — Controle de admissão de chamadas concorrentes com prompt de ocupado
- FR34: Épico 7 — Shutdown gracioso: drenagem SIGTERM com notificação de encerramento
- FR35: Épico 3 — Prompts de áudio pré-gravados para todos os 5 modos de falha
- FR36: Épico 6 — Geração + validação de token criptográfico na URL de callback
- FR37: Épico 6 — Validação de frescor de eventTime do Event Grid

- FR38: Épico 6 — Rate limiting por IP em handshakes WebSocket
- FR39: Épico 1 — Isolamento do Actuator em porta de gerenciamento separada
- FR40: Épico 1 — Corpos JSON genéricos de erro, sem vazamento de informação
- FR41: Épico 4 — Sondagem periódica de conectividade do Voice Live
- FR42: Épico 7 — Varredura de sessões órfãs
- FR43: Épico 7 — Timeout de inatividade de WebSocket após upgrade
- FR44: Épico 7 — Processamento de lote do Event Grid (tratamento individual, 200 para lote)
- FR45: Épico 7 — Deserialização JSON leniente do Voice Live + contador de erro de parse
- FR46: Épico 4 — Detecção de stall de áudio em meio de transmissão
- FR47: Épico 3 — Tratamento de PlayFailed: log ERROR, prosseguir para teardown
- FR48: Épico 5 — Despacho de chamada de ferramenta para APIM via HttpClient em virtual thread
- FR49: Épico 5 — Encaminhar resultado de chamada de ferramenta para Voice Live como tool_call_output
- FR50: Épico 5 — Tratamento de timeout/erro de chamada de ferramenta (payload de erro, sem encerramento de chamada)
- FR51: Épico 5 — Carregar definições de ferramentas de arquivo JSON externo na inicialização de sessão

Lista de Épicos

Épico 1: Fundação do Projeto & Atendimento de Chamadas

O serviço inicia, atende chamadas recebidas, aceita conexões de áudio e fornece visibilidade operacional completa. Um chamador liga → Event Grid entrega `IncomingCall` → o serviço atende via ACS → ACS abre o WebSocket de áudio. Inclui setup do projeto Spring Boot (template inicial, todas as dependências, framework de configuração), logging estruturado, rastreamento distribuído, métricas, health probes e isolamento do actuator.

FRs cobertos: FR1, FR2, FR3, FR7, FR23, FR25, FR28, FR29, FR30, FR31, FR32, FR39, FR40

Notas:

- História 1 deve ser o setup do template inicial (requisito da Arquitetura)
- FR23 (identidade gerenciada ACS) necessário para atender chamadas
- FR25 (auth Event Grid + SubscriptionValidation) necessário para receber chamadas
- FR28/FR29/FR30/FR31/FR32 (logging, rastreamento, probes, config, métricas) são preocupações transversais fundamentais
- FR39/FR40 (isolamento do actuator, erros genéricos) são baseline de segurança

---

Épico 2: Integração Voice Live & Ponte de Áudio

O chamador fala e ouve respostas da IA em tempo real. O serviço conecta-se de saída à API Voice Live, autentica via identidade gerenciada, inicializa a sessão de IA (prompt de sistema, modelo de voz, configurações de VAD) e faz ponte de áudio bidirecionalmente entre ACS e Voice Live com encaminhamento zero-copy.

FRs cobertos: FR8, FR9, FR10, FR11, FR12, FR13

Notas:

- FR9 é a ponte de áudio central — apenas tradução de protocolo, sem buffering
- FR10 estabelece ciclo de vida vinculado: qualquer WebSocket fecha → ambos fazem teardown
- FR12 → auth de identidade gerenciada para Voice Live (sem chaves de API)
- FR13 → session.update com prompt de sistema externalizado, modelo de voz, defs de ferramentas, config de VAD

Épico 3: Inteligência Conversacional & Experiência do Chamador

O chamador tem uma conversa natural e responsiva com suporte a barge-in, tons de conforto e gerenciamento gracioso de tempo. Quando o chamador fala sobre a IA, o barge-in para imediatamente o áudio da IA. Se a conexão levar tempo, o chamador ouve um tom de conforto. Na duração máxima, a IA encerra naturalmente. Falha na reprodução de prompt é tratada sem retentativas.

FRs cobertos: FR6, FR14, FR16, FR35, FR47

Notas:

- FR14 → barge-in: parar encaminhamento de áudio VL + cancelar Play ACS (idempotente)
- FR35 → todos os 5 tipos de prompt: tom de conforto, saudação de fallback, desculpas, ocupado, serviço indisponível
- FR6/FR16 → encerramento por duração máxima + imposição de desconexão forçada
- FR47 → PlayFailed: log em ERROR, prosseguir para teardown, sem retentativa

---

Épico 4: Resiliência de Conexão & Recuperação

Quando a infraestrutura tem problemas, os chamadores experimentam tratamento gracioso em vez de silêncio morto ou desconexão abrupta. Padrão circuit breaker nas conexões Voice Live. Monitoramento de heartbeat detecta obsolescência. Janela de reconexão ACS permite retomada de sessão. Detecção de stall de áudio em meio de transmissão captura respostas congeladas. Sondagem proativa de conectividade aciona o breaker antes que os chamadores sejam afetados.

FRs cobertos: FR4, FR17, FR18, FR19, FR20, FR21, FR41, FR46

Notas:

- FR18/FR19 → circuit breaker: CLOSED → OPEN → HALF-OPEN → CLOSED
- FR17 → heartbeat/keepalive + detecção de obsolescência em ambos WebSockets
- FR20 → reconexão ACS dentro de janela de 5s → tentar retomada de sessão VL
- FR21 → áudio de fallback quando circuit breaker OPEN ou retomada falha
- FR41 → sondagem periódica de conectividade VL, acionamento preventivo do breaker
- FR46 → detecção de stall em meio de transmissão (sem audio delta por C-55ms após início)
- FR4 → teardown gracioso de recursos em todos os caminhos de desconexão

---

Épico 5: Execução de Chamadas de Ferramentas

A IA pode consultar dados de clientes e realizar ações durante a conversa, tornando a chamada acionável. Voice Live envia eventos `tool_call` → o serviço despacha requisições HTTPS para backends via APIM →

encaminha resultados de volta como `tool_call_output`. Definições de ferramentas são carregadas de arquivos JSON externos. Timeouts e erros nunca encerram a chamada.

FRs cobertos: FR48, FR49, FR50, FR51

Notas:

- FR48 → despacho via java.net.http.HttpClient em virtual thread, auth de identidade gerenciada para APIM
- FR50 → timeout/erro → retornar payload de erro para VL, incrementar métrica, chamada continua
- FR51 → carregar defs de ferramentas de arquivo externo na inicialização de sessão, modo degradado se inválido

---

Épico 6: Endurecimento de Segurança

Segurança em profundidade valida cada ponto de entrada, previne ataques de replay, limita abuso e vaza zero informação. Validação JWT nos upgrades WebSocket rejeita antes da escalada de protocolo. Tokens na URL de callback defendem contra callbacks forjados. Validação de timestamp de eventos bloqueia replays. Rate limiting por IP em handshakes WebSocket. Abstração de saúde de credenciais para probes de readiness. Imposição de WSS-only.

FRs cobertos: FR22, FR24, FR26, FR27, FR36, FR37, FR38

Notas:

- FR22 → validação JWT no upgrade de /ws/v1, rejeitar com 401 antes do upgrade
- FR24 → auth de identidade gerenciada VL + pré-requisito RBAC (complementa FR12 do Épico 2)
- FR36 → geração + validação de token criptográfico na URL de callback
- FR37 → validação de frescor de eventTime do Event Grid
- FR38 → rate limiting por IP em handshakes WS (faixa de IP ACS isenta conforme NFR-SEC10)
- FR27 → abstração testável de saúde de credenciais (usada pela probe de readiness do FR30)

Épico 7: Controle de Admissão & Segurança Operacional

O serviço se protege contra sobrecarga e trata cada caso de borda — sem sessões órfãs, sem conexões zumbis, sem eventos perdidos. Controle de admissão de chamadas concorrentes rejeita overflow com mensagem falada. Shutdown gracioso drena chamadas em andamento no SIGTERM. Varredura de sessões órfãs captura sessões vazadas. Timeout de WebSocket inativo limpa conexões mortas. Processamento de lote de eventos e deduplicação de callbacks garantem confiabilidade.

FRs cobertos: FR5, FR33, FR34, FR42, FR43, FR44, FR45

Notas:

- FR33 → controle de admissão local à instância, prompt de ocupado (C-38) quando na capacidade
- FR34 → SIGTERM: marcar unhealthy → notificação de encerramento → drenar dentro de C-34
- FR42 → varredura periódica de sessões órfãs
- FR43 → timeout de inatividade em WebSocket após upgrade (sem áudio dentro de C-54)
- FR5 → deduplicação de callback via cache callId+eventType+correlationId

- FR44 → lote Event Grid: processar individualmente, 200 para lote, log de falhas por evento
- FR45 → deserialização JSON leniente de VL + contador de erro de parse

---

Épico 1: Fundação do Projeto & Atendimento de Chamadas

O serviço inicia, atende chamadas recebidas, aceita conexões de áudio e fornece visibilidade operacional completa.

Story 1.1: Scaffold do Projeto Spring Boot & Framework de Configuração

Como um **desenvolvedor**, Eu quero um projeto Spring Boot 4.x totalmente configurado com todas as dependências, estrutura de pacotes e framework de configuração, Para que todas as histórias subsequentes tenham uma fundação sólida e consistente para construir.

Critérios de Aceite:

Dado um novo projeto Spring Initializr **Quando** gerado com starters (`web`, `websocket`, `actuator`, `validation`, `configuration-processor`) e todas as 12 dependências da arquitetura **Então** o projeto compila com sucesso com `./mvnw clean package` produzindo um JAR executável (NFR-I7)

Dado a configuração da aplicação **Quando** `spring.threads.virtual.enabled=true` está definido **Então** a aplicação usa virtual threads para processamento de requisições (TC-1)

Dado a estrutura do projeto **Quando** examinando os pacotes **Então** todos os 7 pacotes de funcionalidade existem: `config`, `webhook`, `websocket`, `bridge`, `toolcall`, `model`, `exception` (Seção 5 da Arquitetura)

Dado o contexto Spring **Quando** o bean `ServerEndpointExporter` está registrado **Então** anotações JSR 356 `@ServerEndpoint` são detectadas e registradas

Dado classes `@ConfigurationProperties` com prefixo `sofia.*` **Quando** valores de configuração obrigatórios estão ausentes ou inválidos **Então** a aplicação falha ao iniciar dentro de 5s com erro de validação claro (FR31, NFR-R10)

Dado Spring profiles (`test`, `local`, `prod`) **Quando** um profile está ativo **Então** overrides específicos de ambiente se aplicam via Spring relaxed binding (Decisão 9)

Dado o profile de teste **Quando** executando `./mvnw test` **Então** os testes passam com zero variáveis de ambiente Azure e zero chamadas de rede (NFR-I5)

Story 1.2: Fundação de Observabilidade (Logging, Rastreamento, Métricas & Health Probes)

Como um **engenheiro de operações**, Eu quero logging estruturado, rastreamento distribuído, métricas e health probes desde o primeiro dia, Para que eu possa monitorar, diagnosticar e gerenciar o serviço em produção.

Critérios de Aceite:

Dado qualquer evento de log **Quando** emitido **Então** está em formato JSON estruturado com campos `correlationId`, `timestamp`, `level` e `component` (FR28)

Dado um contexto de trace OTel ativo **Quando** um evento de log é emitido **Então** inclui campos `traceId` e `spanId` (FR28, ADR-9)

Dado operações de ciclo de vida de chamada definidas **Quando** processando uma chamada **Então** spans de rastreamento distribuído são emitidos para `ivr.call`, `ivr.call.answer`, `ivr.call.voicelive.connect`, `ivr.call.audio.bridge` (FR29)

Dado métricas Micrometer configuradas **Quando** a aplicação inicia **Então** todos os 14 nomes de métrica (prefixados com `ivr.*`) estão registrados e prontos para emissão (FR32)

Dado a probe de startup **Quando** consultada após primeira aquisição de token de identidade gerenciada **Então** retorna HTTP 200; aquisição de credencial é limitada a 15s (FR30, NFR-P9)

Dado a probe de liveness **Quando** consultada enquanto o contexto Spring está ativo **Então** retorna HTTP 200 (FR30)

Dado a probe de readiness **Quando** consultada **Então** verifica status de credencial em cache e resolução DNS do Voice Live (FR30)

Dado o Spring Boot Actuator **Quando** vinculado à porta de gerenciamento (C-48) **Então** apenas os endpoints `/health`, `/health/liveness`, `/health/readiness`, `/health/startup` são expostos; todos os outros caminhos do actuator retornam 404 (FR39, NFR-SEC6)

Story 1.3: Webhook do Event Grid & Ingestão de Chamadas

Como o **sistema IVR**, Eu quero receber e autenticar eventos de webhook do Event Grid, Para que chamadas recebidas sejam recebidas de forma segura e a sequência de atendimento seja iniciada.

Critérios de Aceite:

Dado um `SubscriptionValidationEvent` do Event Grid **Quando** recebido no endpoint do webhook **Então** o sistema completa o handshake de validação retornando o `validationCode` (FR25, Decisão 3)

Dado um evento `IncomingCall` do Event Grid **Quando** recebido com um token bearer Entra ID válido **Então** o sistema inicia a sequência de atendimento da chamada dentro do timeout de toque configurado C-03 (FR1)

Dado um evento `IncomingCall` **Quando** o cabeçalho `Authorization` contém um token Entra ID inválido, expirado ou ausente **Então** o sistema rejeita a requisição com HTTP 401 (FR25)

Dado qualquer resposta de erro HTTP do controller webhook **Quando** retornada a um cliente **Então** o corpo da resposta é uma estrutura JSON genérica sem stack traces, nomes de classe ou detalhes específicos do framework (FR40, NFR-SEC5)

Dado um evento `IncomingCall` recebido **Quando** processado **Então** o evento é registrado em log em JSON estruturado com `correlationId` em nível INFO

Story 1.4: ACS Call Automation & Atendimento de Chamadas

Como um **chamador**, Eu quero que o sistema atenda minha chamada com streaming de áudio bidirecional, Para que minha voz chegue ao agente de IA e eu possa ouvir suas respostas.

Critérios de Aceite:

Dado um evento `IncomingCall` validado **Quando** a sequência de atendimento inicia **Então** o sistema atende usando `CallAutomationClient` com streaming de mídia bidirecional habilitado em PCM 24K mono (FR2, TC-7)

Dado `CallAutomationClient` **Quando** autenticando no ACS **Então** credenciais de identidade gerenciada são usadas sem strings de conexão ou chaves de API (FR23, NFR-SEC2)

Dado um callback `CallConnected` **Quando** recebido do ACS **Então** um novo `CallContext` é criado com o `callId` do ACS como `correlationId` e o estado da chamada transiciona para `INITIALIZING` (FR3, CC-1, CC-2)

Dado um callback `CallDisconnected` **Quando** recebido do ACS **Então** o estado da chamada transiciona para `TERMINATED` e todos os recursos associados são marcados para limpeza (FR3)

Dado callbacks `PlayFailed` ou `PlayCompleted` **Quando** recebidos do ACS **Então** o estado interno da chamada é atualizado adequadamente (FR3)

Dado a operação de atendimento de chamada **Quando** medida de ponta a ponta (`IncomingCall` recebido até resposta de `answerCall()`) **Então** a latência é < 3s no p95 (NFR-P5)

Story 1.5: Endpoint de Servidor WebSocket de Áudio ACS

Como o **sistema IVR**, Eu quero aceitar e fazer parse de conexões WebSocket de áudio do ACS, Para que o stream de áudio de entrada esteja pronto para ponte com o Voice Live.

Critérios de Aceite:

Dado uma requisição de conexão WebSocket de áudio do ACS **Quando** direcionada ao caminho versionado `/ws/v1` **Então** o sistema aceita a conexão WSS e faz o upgrade do protocolo (FR7)

Dado frames binários WebSocket de entrada do ACS **Quando** recebidos na conexão estabelecida **Então** são parseados em pacotes `AudioMetadata` e `AudioData` usando `StreamingData.parse()` (FR7)

Dado um upgrade WebSocket bem-sucedido **Quando** a conexão é estabelecida **Então** é associada ao `CallContext` correto via o `correlationId` da chamada

Dado o tempo de inicialização da aplicação **Quando** medido do lançamento da JVM até probe de readiness HTTP 200 **Então** completa em ≤ 30s em uma JVM fria (NFR-P9)

Épico 2: Integração Voice Live & Ponte de Áudio

O chamador fala e ouve respostas da IA em tempo real.

Story 2.1: Cliente WebSocket Voice Live & Inicialização de Sessão

Como o **sistema IVR**, Eu quero conectar à API Voice Live via WebSocket, autenticar com identidade gerenciada e inicializar a sessão de IA, Para que a IA conversacional esteja pronta para processar o áudio do chamador.

Critérios de Aceite:

Dado uma chamada no estado `INITIALIZING` **Quando** a conexão Voice Live é iniciada **Então** um WebSocket de saída é estabelecido ao endpoint configurado (C-07) com versão de API (C-08) dentro do timeout de conexão (C-10) usando `java.net.http.WebSocket` (FR11, TC-3)

Dado a conexão WebSocket Voice Live **Quando** autenticando **Então** credenciais de identidade gerenciada (`DefaultAzureCredential`) são usadas sem chaves de API no código ou configuração (FR12, TC-5, TC-6)

Dado uma conexão WebSocket Voice Live bem-sucedida **Quando** `session.created` é recebido **Então** o sistema envia um evento `session.update` configurando: prompt de sistema externalizado (C-12), modelo de voz (C-13), definições de ferramentas de C-60 e configurações de VAD — tipo (C-14), threshold (C-15), duração de silêncio (C-16), supressão de ruído (C-17), cancelamento de eco (C-18) (FR13)

Dado o Voice Live rejeita a sessão **Quando** `session.update` falha **Então** a rejeição é registrada em log em WARN com `correlationId`, código/mensagem de erro e tamanho do prompt de sistema (FR13)

Dado a latência de conexão Voice Live **Quando** medida do handshake WebSocket até `session.created` **Então** é ≤ 2s no p95, ≤ 3s no p99 (NFR-P6)

Dado o WebSocket Voice Live **Quando** a conexão usa o esquema `ws://` **Então** a conexão nunca é iniciada — apenas `wss://` é permitido (NFR-SEC3)

Story 2.2: Ponte de Áudio Bidirecional

Como um **chamador**, Eu quero minha voz encaminhada para a IA e a voz da IA encaminhada de volta para mim em tempo real, Para que eu possa ter uma conversa natural e de baixa latência.

Critérios de Aceite:

Dado frames de áudio chegando do ACS (via Story 1.5) **Quando** a sessão Voice Live está ativa **Então** o áudio é encaminhado ao Voice Live sem buffering, inspeção ou transformação do conteúdo — apenas tradução de protocolo (FR9, NFR-P8, TC-17)

Dado eventos `response.audio.delta` do Voice Live **Quando** recebidos durante uma sessão ativa **Então** o áudio é enviado ao ACS via formato de streaming de saída usando `OutStreamingData.getStreamingDataForOutbound()` (FR8)

Dado um único frame de áudio atravessando a ponte **Quando** medido de ponta a ponta **Então** a latência é < 50ms no p99 (NFR-P4)

Dado o tempo até primeiro áudio **Quando** medido de `CallConnected` até primeiro byte de áudio enviado ao ACS **Então** é < 3s no p95; cache JWKS é pré-aquecido durante a inicialização (NFR-P1)

Dado memória por chamada **Quando** medida em estado estável com ponte de áudio ativa **Então** uso de heap é ≤ 200 KB por chamada (NFR-P7)

Dado nenhum tipo reativo **Quando** examinando o código da ponte de áudio **Então** zero tipos `Mono`, `Flux` ou `Publisher` são usados (NFR-I2, M-1)

Story 2.3: Ciclo de Vida Vinculado de WebSocket

Como o **sistema IVR**, Eu quero que o fechamento de qualquer WebSocket dispare teardown gracioso de ambos, Para que nenhuma conexão ou sessão órfã permaneça após o fim de uma chamada.

Critérios de Aceite:

Dado o WebSocket do ACS fecha (códigos 1000, 1001) **Quando** um fechamento normal é detectado **Então** o WebSocket do Voice Live é fechado e todos os recursos da chamada são liberados em ≤ 3s (FR10, NFR-R2)

Dado o WebSocket do Voice Live fecha (códigos 1000, 1001) **Quando** um fechamento normal é detectado **Então** o WebSocket do ACS é fechado e todos os recursos da chamada são liberados em ≤ 3s (FR10, NFR-R2)

Dado o WebSocket do ACS fecha de forma anormal (código 1006) **Quando** detectado **Então** a sessão Voice Live é mantida por C-28 (janela de reconexão de 5s), depois o teardown prossegue se não houver reconexão (NFR-R12)

Dado transições de estado concorrentes em ambos WebSockets **Quando** ambos os lados tentam teardown simultaneamente **Então** nenhuma condição de corrida, vazamento de recursos ou teardown duplicado ocorre (NFR-R5)

Dado limpeza de recursos **Quando** o teardown é completado **Então** o `CallContext` é removido do store em memória, todas as métricas são atualizadas e o span `ivr.call` é completado

---

Épico 3: Inteligência Conversacional & Experiência do Chamador

O chamador tem uma conversa natural e responsiva com suporte a barge-in, tons de conforto e gerenciamento gracioso de tempo.

Story 3.1: Barge-In & Tratamento de Interrupção de Fala

Como um **chamador**, Eu quero interromper a IA no meio da frase e fazer com que ela pare imediatamente, Para que eu possa redirecionar a conversa naturalmente sem esperar.

Critérios de Aceite:

Dado o Voice Live envia `input_audio_buffer.speech_started` **Quando** a IA está atualmente transmitindo áudio para o chamador **Então** o sistema para imediatamente de encaminhar frames de áudio do Voice Live para o ACS (FR14)

Dado o Voice Live envia `input_audio_buffer.speech_started` **Quando** uma operação Play do ACS está em andamento **Então** o sistema cancela a operação Play; tanto parar-encaminhamento quanto cancelar-play são idempotentes — seguros para invocar redundantemente (FR14)

Dado a latência de barge-in **Quando** medida de `speech_started` recebido até `StopAudio` enviado ao ACS **Então** é < 100ms no p95 (NFR-P3, UX-REQ-8)

Dado a latência de resposta da IA **Quando** medida do fim de fala do VAD até primeiro `response.audio.delta` **Então** é < 1s no p95 (NFR-P2, UX-REQ-7)

Dado o chamador faz barge-in **Quando** a IA estava no meio de uma frase **Então** nenhum corte abrupto no meio de frase ocorre do lado do chamador — o áudio para de forma limpa (UX-REQ-6)

Story 3.2: Prompts de Áudio & Experiências de Fallback

Como um **chamador**, Eu quero ouvir sinais de áudio apropriados durante atrasos de conexão e falhas, Para que eu nunca fique em silêncio ou confuso com comportamento abrupto.

Critérios de Aceite:

Dado a conexão Voice Live excede C-33 (threshold de tom de conforto) **Quando** nenhum áudio de IA foi enviado ao chamador ainda **Então** o sistema reproduz um tom de conforto via ACS Play action (FR35a, UX-REQ-1)

Dado a sessão Voice Live estabelecida mas sem áudio de IA dentro de C-32 **Quando** o timeout de saudação de fallback expira **Então** o sistema reproduz o prompt de saudação de fallback (FR35b, UX-REQ-4)

Dado uma desconexão inesperada do Voice Live **Quando** o chamador ainda está conectado **Então** o sistema reproduz o prompt de desculpas antes de desconectar graciosamente (FR35c, UX-REQ-3)

Dado rejeição de admissão (do Épico 7) **Quando** o limite de chamadas por instância é excedido **Então** o sistema reproduz o prompt de ocupado (C-38) para o chamador (FR35d, UX-REQ-11)

Dado o circuit breaker está OPEN (do Épico 4) **Quando** uma nova chamada chega **Então** o sistema reproduz o prompt de serviço indisponível antes de desconectar (FR35e, UX-REQ-10)

Dado todos os 5 prompts de modo de falha **Quando** avaliados quanto à cobertura **Então** 100% dos modos de falha definidos têm um prompt de áudio correspondente (NFR-I1)

Story 3.3: Gerenciamento de Duração Máxima & Tratamento de PlayFailed

Como o **sistema IVR**, Eu quero impor limites de tempo de chamada graciosamente e tratar falhas de reprodução de prompt com segurança, Para que chamadas terminem cooperativamente e falhas não cascateiem.

Critérios de Aceite:

Dado uma chamada ativa se aproximando da duração máxima **Quando** restam C-29 menos C-30 segundos **Então** o sistema injeta uma mensagem de contexto ao Voice Live solicitando que a IA encerre naturalmente (FR6, FR16, UX-REQ-9)

Dado a chamada atinge a duração máxima **Quando** a chamada não terminou voluntariamente após a notificação de encerramento **Então** o sistema impõe uma desconexão forçada (FR6)

Dado um callback `PlayFailed` do ACS **Quando** recebido para qualquer reprodução de prompt **Então** o sistema registra em log em ERROR com `correlationId`, identificador do prompt e razão de falha do ACS — NÃO tenta novamente ou reproduz prompt substituto, prossegue diretamente para encerramento gracioso (FR47)

Dado mensagens de sistema durante sessão **Quando** injetadas ao Voice Live (ex.: notificação de encerramento) **Então** não interrompem ou resetam o contexto da conversa ativa (FR16)

Épico 4: Resiliência de Conexão & Recuperação

Quando a infraestrutura tem problemas, os chamadores experimentam tratamento gracioso em vez de silêncio morto ou desconexão abrupta.

Story 4.1: Teardown Gracioso de Recursos

Como o **sistema IVR**, Eu quero todos os recursos de chamada limpos deterministicamente quando uma chamada termina, Para que nenhuma conexão, memória ou sessão vaze independentemente de como a desconexão ocorre.

Critérios de Aceite:

Dado uma chamada terminando (qualquer gatilho — chamador desliga, IA desconecta, erro, timeout) **Quando** o teardown começa **Então** ambas as conexões WebSocket são fechadas, o `CallContext` é removido do `ConcurrentHashMap`, métricas são atualizadas e o span de rastreamento é completado (FR4)

Dado o teardown é iniciado **Quando** um fechamento de WebSocket ou limpeza de recurso lança uma exceção **Então** a limpeza continua via `try-finally` em todos os caminhos — nenhuma falha única impede outros recursos de serem liberados (FR4, CC-6)

Dado a taxa de conclusão de chamadas **Quando** medida em todas as chamadas atendidas **Então** ≥ 99% terminam graciosamente (NFR-R8)

Dado um teardown completado **Quando** nenhuma sessão Voice Live permanece após fim da chamada + timeout **Então** zero sessões órfãs existem (NFR-R7)

Story 4.2: Monitoramento de Heartbeat & Detecção de Obsolescência

Como o **sistema IVR**, Eu quero detectar quando qualquer conexão WebSocket fica obsoleta, Para que conexões mortas sejam encerradas rapidamente em vez de deixar chamadores em silêncio morto.

Critérios de Aceite:

Dado qualquer uma das conexões WebSocket ACS ou Voice Live **Quando** nenhuma mensagem é recebida dentro do timeout de inatividade configurado (C-05 para ACS, C-11 para Voice Live) **Então** a conexão é detectada como obsoleta (FR17)

Dado um WebSocket obsoleto detectado **Quando** a obsolescência é confirmada **Então** a conexão obsoleta é forçosamente fechada, disparando o teardown de ciclo de vida vinculado conforme FR10 (FR17)

Dado a latência de detecção de heartbeat **Quando** medida do último heartbeat até detecção de obsolescência **Então** é ≤ 3s (NFR-R9)

Dado mecanismos de heartbeat/keepalive **Quando** ativos em ambas as conexões **Então** operam independentemente — falha de heartbeat de uma conexão não interfere na lógica de detecção da outra

---

Story 4.3: Circuit Breaker & Sondagem Proativa de Conectividade

Como o **sistema IVR**, Eu quero um circuit breaker nas conexões Voice Live que acione em falhas repetidas e se recupere automaticamente, Para que novas chamadas não sejam direcionadas a uma dependência quebrada.

Critérios de Aceite:

Dado falhas consecutivas de conexão Voice Live **Quando** a contagem atinge C-22 (threshold) **Então** o circuit breaker aciona para estado OPEN (FR18, NFR-R3)

Dado o circuit breaker está OPEN **Quando** o atraso half-open configurável (C-23) expira **Então** o breaker transiciona para HALF-OPEN, permitindo uma tentativa de conexão de sondagem (FR19)

Dado o circuit breaker está HALF-OPEN **Quando** C-24 conexões sucessivas sucedem **Então** o breaker transiciona de volta para CLOSED (FR19)

Dado o circuit breaker está OPEN **Quando** uma nova chamada chega **Então** o sistema reproduz o prompt de serviço indisponível (FR35e via Épico 3) e desconecta graciosamente — nenhuma tentativa de conexão é feita ao Voice Live (FR21)

Dado a sondagem periódica de conectividade Voice Live (intervalo C-25) **Quando** sondas falham **Então** o circuit breaker é acionado preventivamente para OPEN antes que qualquer chamador seja afetado (FR41)

Dado a métrica de estado do circuit breaker `ivr.circuit_breaker.state` **Quando** o breaker transiciona **Então** a métrica é atualizada e um evento de log estruturado é emitido em nível WARN

Story 4.4: Reconexão ACS, Retomada de Sessão & Detecção de Stall

Como um **chamador**, Eu quero que breves falhas de rede não matem minha conversa, e respostas congeladas da IA sejam capturadas, Para que eu experimente resiliência em vez de silêncio morto.

Critérios de Aceite:

Dado o WebSocket do ACS fecha de forma anormal (código 1006) **Quando** dentro da janela de reconexão C-28 (5s) **Então** o ACS reconecta e o sistema tenta retomar a sessão Voice Live existente (FR20, NFR-R12)

Dado o ACS reconecta dentro da janela **Quando** a retomada de sessão Voice Live falha **Então** o sistema recorre a reproduzir o prompt de desculpas e desconectar graciosamente (FR20, FR21)

Dado a janela de reconexão expira **Quando** nenhuma reconexão ACS ocorre **Então** o teardown prossegue para ambos WebSockets (NFR-R12)

Dado o Voice Live está transmitindo áudio (eventos `response.audio.delta` recebidos) **Quando** nenhum evento de áudio adicional e nenhum `response.done` chegam dentro de C-55 (5000ms) **Então** um stall em meio de transmissão é detectado — o sistema registra em log WARN, reproduz o prompt de desculpas (C-37) e inicia teardown gracioso (FR46)

Dado a precisão de detecção de stall **Quando** medida em conversas normais com pausas naturais **Então** a taxa de falsos positivos é < 1% (NFR-R13)

Épico 5: Execução de Chamadas de Ferramentas

Quando o Voice Live precisa de dados externos durante uma conversa, a ponte despacha chamadas de ferramentas para o gateway APIM e retorna resultados — permitindo que a IA responda perguntas de faturamento e conta em tempo real.

3 Ferramentas Predefinidas (escopo MVP):

Nome da Ferramenta	Endpoint APIM	Propósito
`get-user-data`	`GET {C-57}/api/v1/users/{id}`	Buscar dados de conta/faturamento do chamador
`send-invoice-registered-email`	`POST {C-57}/api/v1/invoices/send`	Enviar cópia da fatura para o email registrado do chamador
`send-invoice-provided-email`	`POST {C-57}/api/v1/invoices/send`	Enviar cópia da fatura para um email ditado pelo chamador

Story 5.1: Carregamento de Definições de Ferramentas & Registro na Sessão

Como o **sistema IVR**, Eu quero carregar as 3 definições de ferramentas de arquivos externos e registrá-las com o Voice Live no início da sessão, Para que o Voice Live saiba quais ferramentas estão disponíveis e possa invocá-las durante a conversa.

Critérios de Aceite:

Dado a aplicação inicia **Quando** o `ToolDefinitionRegistry` inicializa **Então** carrega 3 arquivos de definição de ferramentas do caminho especificado por C-60 (`classpath:tool-definitions.json` por padrão): `get-user-data.json`, `send-invoice-registered-email.json`, `send-invoice-provided-email.json` — cada um seguindo o esquema de function calling da OpenAI (FR51)

Dado uma nova sessão Voice Live é estabelecida **Quando** o evento `session.update` é enviado (FR13) **Então** todas as 3 definições de ferramentas são incluídas no array `tools` da configuração de sessão, habilitando o Voice Live a invocar `get-user-data`, `send-invoice-registered-email` e `send-invoice-provided-email`

Dado um arquivo de definições de ferramentas está ausente ou contém JSON inválido **Quando** o registro carrega na inicialização **Então** o sistema registra em log em nível ERROR com o nome do arquivo e erro de parse, e inicia a sessão sem definições de ferramentas (modo degradado) — a conversa com IA funciona mas não pode invocar ferramentas (FR51)

Dado o conteúdo do registro de ferramentas **Quando** carregado com sucesso **Então** cada `ToolDefinition` carrega seu caminho de endpoint APIM (ex.: `/api/v1/users/{id}`, `/api/v1/invoices/send`), método HTTP, override de timeout e esquema de resposta esperado — correspondendo à Decisão 5 da Arquitetura

---

Story 5.2: Despacho, Resposta & Tratamento de Erros de Chamadas de Ferramentas

Como um **chamador**, Eu quero que a IA busque meus dados de conta e envie faturas durante nossa conversa, Para que eu possa obter respostas de faturamento e receber cópias de fatura sem ser transferido para um atendente humano.

Critérios de Aceite:

Dado o Voice Live envia um evento `tool_call` para `get-user-data` com argumento `{"phone": "<telefone-chamador>"}` **Quando** o `ToolCallDispatcher` recebe o evento **Então** despacha `GET {C-57}/api/v1/users/{id}` para o gateway APIM com `Authorization: Bearer <token>` (DefaultAzureCredential, escopo C-57b), o `correlationId` da chamada e os argumentos da ferramenta (FR48, SEC-REQ-18)

Dado o Voice Live envia um evento `tool_call` para `send-invoice-registered-email` ou `send-invoice-provided-email` **Quando** o `ToolCallDispatcher` recebe o evento **Então** despacha `POST {C-57}/api/v1/invoices/send` para o gateway APIM com o mesmo modelo de auth, `correlationId` e os argumentos específicos da ferramenta (FR48)

Dado o gateway APIM retorna uma resposta HTTP bem-sucedida para qualquer chamada de ferramenta **Quando** a resposta é recebida **Então** o sistema a encapsula como evento `tool_call_output` (contendo `call_id` + payload da resposta) e envia de volta ao Voice Live dentro de 5s p95 (FR49, AC-23, NFR-P10)

Dado o gateway APIM não responde dentro de C-58 (padrão 5000ms) ou retorna 4xx/5xx **Quando** o timeout ou erro ocorre **Então** o sistema retorna um `tool_call_output` de erro ao Voice Live (ex.: `{"error": true, "message": "Não foi possível recuperar os dados neste momento"}`), registra em log WARN com `correlationId`, nome da ferramenta, status HTTP (se disponível) e tempo decorrido, incrementa `ivr.tool_call.errors`, e **NÃO** encerra a chamada — a conversa com IA continua graciosamente (FR50, AC-24, NFR-R14)

Dado o `ToolCallDispatcher` usa `RestClient` (Spring Boot 4.x) **Quando** despachando chamadas de ferramentas em virtual threads **Então** um circuit breaker Resilience4j protege contra acúmulo de threads bloqueadas quando APIM/backend está fora — sem retentativa (fail-fast para UX de voz, Decisão 5 da Arquitetura)

Dado `C-59` (máximo de chamadas de ferramentas concorrentes por chamada, padrão 3) **Quando** uma chamada de ferramenta é despachada **Então** chamadas de ferramentas concorrentes por chamada são limitadas por C-59; chamadas de ferramentas adicionais além do limite são enfileiradas ou rejeitadas

Dado o stub do gateway WireMock (profile de teste) **Quando** testes de integração rodam **Então** WireMock em `localhost:8082` responde a `GET /api/v1/users/{id}` e `POST /api/v1/invoices/send`, incluindo asserções para cabeçalho `Authorization: Bearer` e propagação de `correlationId`; injeção de falhas suporta delay configurável > C-58, erros 4xx/5xx e conexão recusada

> **Nota:** O PRD originalmente especificou `POST /api/v1/users/{id}` para `get-user-data`. Alterado para `GET` pois é uma operação de leitura. Sinalizado para revisão do Arquiteto — a ponte é agnóstica ao verbo (lê método HTTP da config de definição de ferramenta), então isso afeta apenas o arquivo de definição de ferramenta e stubs WireMock.

Épico 6: Endurecimento de Segurança

Todas as conexões de entrada são autenticadas, credenciais são validadas, ataques de replay são bloqueados e rate limiting protege contra abuso — zero trust em cada fronteira.

Story 6.1: Validação JWT em Endpoints WebSocket & Callback

Como o **sistema IVR**, Eu quero validar tokens JWT em cada upgrade WebSocket de entrada e requisição de callback do ACS, Para que apenas conexões ACS autenticadas possam transmitir áudio ou disparar eventos de chamada.

Critérios de Aceite:

Dado uma requisição de upgrade WebSocket do ACS para `/ws/v1` **Quando** o cabeçalho `Authorization` contém um JWT válido e não expirado **Então** o upgrade prossegue e a conexão WebSocket é estabelecida (FR22)

Dado uma requisição de upgrade WebSocket do ACS **Quando** o JWT está ausente, expirado ou inválido **Então** a conexão é rejeitada com HTTP 401 **antes** do upgrade — nenhuma sessão WebSocket é criada (FR22, SEC-REQ-1)

Dado uma requisição de callback mid-call do ACS **Quando** recebida no endpoint de callback **Então** o JWT assinado nos cabeçalhos é validado contra o JWKS do ACS (SEC-REQ-3)

Dado o endpoint JWKS **Quando** o serviço inicia **Então** o conjunto de chaves é pré-buscado e cacheado para evitar latência na primeira requisição (NFR-P1) — refresh do cache segue práticas padrão de rotação JWKS

Dado uma URL de callback registrada com o ACS **Quando** a chamada é atendida via `answerCall()` **Então** a URL contém um token criptograficamente aleatório (comprimento mínimo C-50), e cada callback daquela chamada valida o token — rejeitando requisições com token ausente ou incompatível (FR36, SEC-REQ-4)

Story 6.2: Autenticação do Event Grid & Proteção contra Replay

Como o **sistema IVR**, Eu quero validar eventos do Event Grid e rejeitar eventos obsoletos reproduzidos, Para que apenas eventos legítimos e recentes de chamadas recebidas disparem o processamento de chamadas.

Critérios de Aceite:

Dado um `SubscriptionValidationEvent` do Event Grid **Quando** recebido no endpoint do webhook **Então** o sistema responde com o `validationCode` para completar o handshake (SEC-REQ-2)

Dado um evento `IncomingCall` do Event Grid **Quando** o `eventTime` é mais antigo que C-45 **Então** o evento é rejeitado com HTTP 400 e registrado em log em WARN com `correlationId` e idade do evento (FR37, SEC-REQ-14)

Dado um evento `IncomingCall` **Quando** o `eventTime` está dentro da janela de frescor C-45 **Então** o evento é processado normalmente

Dado o endpoint do webhook Event Grid **Quando** recebendo requisições **Então** o cabeçalho `Authorization` é validado como token bearer Entra ID (SEC-REQ-2)

Story 6.3: Identidade Gerenciada & Saúde de Credenciais

Como o **sistema IVR**, Eu quero toda autenticação de saída tratada via identidade gerenciada sem segredos na config, Para que a superfície de ataque seja minimizada e credenciais sejam rotacionadas automaticamente.

Critérios de Aceite:

Dado a conexão WebSocket Voice Live **Quando** o serviço inicia a conexão de saída **Então** a autenticação usa `DefaultAzureCredential` (identidade gerenciada em prod, `az login` localmente) — zero chaves de API no código ou configuração (FR24, SEC-REQ-11)

Dado a abstração `CredentialHealthIndicator` **Quando** a probe de readiness a consulta **Então** reporta o status do token em cache (válido, expirando em breve, expirado) sem disparar refresh de token ativo (FR27)

Dado a aquisição de token `DefaultAzureCredential` para o gateway APIM **Quando** uma chamada de ferramenta é despachada **Então** o token bearer adquirido é escopado a C-57b e enviado no cabeçalho `Authorization` — se a aquisição falhar, a chamada de ferramenta falha graciosamente (caminho de erro FR50) em vez de enviar uma requisição não autenticada (SEC-REQ-18)

Dado a infraestrutura **Quando** implantando o serviço **Então** a role RBAC `Cognitive Services User` é atribuída no recurso Foundry — este é um pré-requisito de infraestrutura, não código de aplicação (FR24)

Story 6.4: Imposição de WSS & Rate Limiting de WebSocket

Como o **sistema IVR**, Eu quero todas as conexões WebSocket usando `wss://` exclusivamente e tentativas de handshake limitadas por taxa por IP, Para que áudio em texto puro nunca trafegue pela rede e tentativas de conexão por força bruta sejam bloqueadas.

Critérios de Aceite:

Dado o endpoint de servidor WebSocket do lado ACS **Quando** qualquer tentativa de conexão usa `ws://` (texto puro) **Então** a conexão é rejeitada — apenas `wss://` é aceito (FR26)

Dado o cliente WebSocket de saída Voice Live **Quando** iniciando uma conexão **Então** apenas URLs `wss://` são usadas — o código nunca constrói uma URI `ws://` (FR26)

Dado um único endereço IP **Quando** tentativas de handshake WebSocket excedem C-46 dentro da janela de tempo C-47 **Então** conexões adicionais são rejeitadas com HTTP 429 (FR38, SEC-REQ-15)

Dado rejeições de rate limit **Quando** ocorrem **Então** o evento é registrado em log em WARN com o IP de origem e contagem de rejeições — nenhuma PII além do IP é registrada (SEC-REQ-6)

Épico 7: Controle de Admissão & Segurança Operacional

O sistema se protege contra sobrecarga, limpa recursos órfãos e trata eventos malformados graciosamente — para que casos de borda operacionais nunca degradem o serviço para chamadores ativos.

Story 7.1: Admissão de Chamadas Concorrentes & Shutdown Gracioso

Como o **sistema IVR**, Eu quero rejeitar novas chamadas quando o limite por instância é atingido e drenar chamadas existentes durante o shutdown, Para que chamadores ativos nunca experimentem degradação por sobrecarga e deploys sejam zero-downtime.

Critérios de Aceite:

Dado o número de chamadas ativas nesta instância é igual a C-26 (máximo de chamadas concorrentes) **Quando** um novo evento `IncomingCall` chega **Então** o sistema rejeita a chamada com o prompt de

ocupado (C-38), registra em log WARN com `correlationId` e incrementa `ivr.calls.rejected` (FR33, FR34, UX-REQ-11)

Dado C-26 não é atingido **Quando** um novo evento `IncomingCall` chega **Então** a chamada é admitida normalmente

Dado SIGTERM é recebido (shutdown do container) **Quando** o sinal de shutdown é detectado **Então** o sistema para de aceitar novas chamadas, espera até C-34 (timeout de shutdown gracioso) para chamadas ativas completarem, depois encerra forçosamente as sessões restantes (FR34)

Dado chamadas ativas durante shutdown **Quando** o timeout de shutdown gracioso (C-34) é atingido com chamadas ainda ativas **Então** chamadas restantes são forçosamente encerradas — ambos WebSockets fechados, `CallContext` removido, métricas atualizadas (FR34)

Dado o contador de admissão **Quando** uma chamada completa teardown (Épico 4 Story 4.1) **Então** a contagem de chamadas ativas é decrementada atomicamente — sem drift do contador ao longo do tempo

---

Story 7.2: Deduplicação de Callback & Processamento de Lote de Eventos

Como o **sistema IVR**, Eu quero callbacks duplicados do ACS rejeitados e arrays de callback em lote processados corretamente, Para que eventos duplicados não corrompam o estado da chamada e a entrega em lote do ACS funcione perfeitamente.

Critérios de Aceite:

Dado um callback do ACS com uma combinação de `operationContext` + `eventType` já processada para esta chamada **Quando** o duplicado chega **Então** o callback é registrado em log em DEBUG e descartado — sem mutação de estado, sem erro (FR44)

Dado o ACS entrega um payload de callback como array JSON (modo lote) **Quando** o endpoint do webhook o recebe **Então** cada evento no array é individualmente validado e despachado em ordem — uma falha processando um evento não impede o processamento de eventos subsequentes (FR5)

Dado a janela de deduplicação **Quando** dimensionada por chamada **Então** o conjunto de dedup é liberado com o `CallContext` no teardown — sem crescimento ilimitado de memória (C-56)

Story 7.3: Varredura de Sessões Órfãs & Timeout de Conexão Inativa

Como o **sistema IVR**, Eu quero uma varredura periódica que detecte e limpe sessões órfãs, e conexões inativas que escaparam da detecção de heartbeat, Para que recursos vazados não se acumulem e degradem a instância ao longo do tempo.

Critérios de Aceite:

Dado uma entrada `CallContext` no mapa de sessões ativas **Quando** a varredura roda no intervalo C-52 e encontra uma sessão cujo timestamp de última atividade excede C-54 (threshold de órfão) **Então** a sessão é forçosamente encerrada — ambos WebSockets fechados, `CallContext` removido, `ivr.sessions.orphaned` incrementado, registrado em log em WARN com `correlationId` (FR42, FR43)

Dado a varredura roda **Quando** todas as sessões têm atividade recente dentro de C-54 **Então** nenhuma ação é tomada

Dado o threshold de órfão (C-54) **Quando** definido **Então** é estritamente maior que os timeouts de inatividade de heartbeat (C-05, C-11) — a varredura é uma rede de segurança, não o mecanismo primário de detecção

Dado a varredura **Quando** encerra forçosamente uma sessão **Então** usa o mesmo caminho de teardown da Story 4.1 — sem lógica de limpeza separada

---

Story 7.4: Parse Leniente de Eventos & Resiliência a Erros

Como o **sistema IVR**, Eu quero que eventos desconhecidos ou malformados do ACS e Voice Live sejam registrados em log e ignorados sem crash, Para que adições de protocolo ou malformações transitórias não derrubem chamadas ativas.

Critérios de Aceite:

Dado um callback do ACS com um `eventType` não reconhecido **Quando** recebido no endpoint do webhook **Então** o evento é registrado em log em WARN com o tipo desconhecido e `correlationId`, e descartado — sem exceção, sem resposta 5xx (FR45)

Dado um evento do Voice Live com um campo `type` não reconhecido **Quando** recebido no cliente WebSocket **Então** o evento é registrado em log em WARN e ignorado — a ponte de áudio e a conversa ativa continuam inalteradas (FR45)

Dado um evento do Voice Live com `type` válido mas payload malformado (ex.: campos obrigatórios ausentes) **Quando** a deserialização JSON falha parcialmente **Então** o sistema registra em log o erro de parse em ERROR com tipo do evento e `correlationId`, ignora aquele único evento e continua processando — a conexão WebSocket NÃO é fechada (FR45, SEC-REQ-10)

Dado qualquer exceção não tratada no processamento de eventos **Quando** ocorre no handler de callback do ACS ou handler de mensagem do Voice Live **Então** a exceção é capturada na fronteira do handler, registrada em log em ERROR, e NÃO se propaga para crashar a thread ou fechar a conexão (FR45)