Chat4All - Plataforma de Comunicação Ubíqua

1. Visão geral do projeto — Chat4All v2

Objetivo: desenvolver uma **plataforma de comunicação ubíqua** (API) capaz de rotear mensagens e arquivos entre usuários em múltiplas plataformas (ex.: WhatsApp, Instagram Direct, Messenger, Telegram) e entre clientes internos (web/mobile/CLI). Suporta comunicação privada e em grupo, persistência no servidor, controle de envio/recebimento/leitura, entrega de arquivos até **2 GB** e operação em escala (milhões de usuários). Arquitetura desenhada para **escalabilidade horizontal** e **alta disponibilidade**.

2. Requisitos Funcionais (detalhados)

2.1 Mensageria básica

- Criar/entrar em conversas privadas (1:1) e grupos (n membros).
- Enviar mensagem de texto entre usuários (instância local → outro usuário em qualquer plataforma suportada).
- Enviar arquivos de até **2 GB** (ver seção técnica para chunking/resume).
- Recepção em tempo real para usuários conectados; entrega retardada para usuários offline (persistência).

2.2 Controle de envio / entrega / leitura

- Estados de mensagem: SENT (aceito pelo servidor), DELIVERED (entregue ao dispositivo alvo), READ (lido no dispositivo).
- O remetente pode pedir confirmação de entrega/ leitura; o sistema deve fornecer histórico de estados.
- Mensagens idempotentes (cada mensagem tem um message_id universal) para evitar duplicação.

2.3 Multiplataforma e roteamento por canal

- Ao enviar, o usuário escolhe os **canais** de entrega (ex.: ["whatsapp", "instagram", "telegram"]) ou all.
- Plataforma atua como broker/unificador: recebe uma mensagem e chama os **connectors/adapters** apropriados para cada canal.
- Suporte a mapear usuários entre plataformas (ex.: usuário interno X vinculado a WhatsApp + Instagram).

- Permitir que um usuário WhatsApp envie mensagem que chegue ao Direct do Instagram de outro usuário, quando esse destino estiver configurado.

2.4 Persistência

- Mensagens (metadados e payloads menores) persistidos em banco distribuído; arquivos grandes armazenados em armazenamento de objetos com referência no banco.
- Entrega "store-and-forward" quando destinatário offline.

2.5 API pública e SDKs

- Expor API REST (ou gRPC) para envio/recebimento de mensagens, criação de conversas, anexação de arquivos, consulta de histórico e webhooks para callbacks de eventos (delivery/read).

2.6 Extensibilidade de canais

- Plugin architecture para adicionar novos canais (adapters) sem alterar núcleo.
- Interface padronizada para adapters: connect(), sendMessage(), sendFile(), webhookHandler().

3. Requisitos Não-Funcionais (NFR)

3.1 Escalabilidade

- Suportar **milhões de usuários ativos** e picos de tráfego (ex.: 10^6 a 10^7 mensagens por minuto, dependendo do escopo do curso).
- Arquitetura **stateless** nas front-APIs; estado persistido em stores escaláveis.
- Auto-scale horizontal (novos nós podem ser adicionados sem downtime).

3.2 Alta disponibilidade / Tolerância a falhas

- SLA objetivo: **≥99.95%**
- Failover automático de componentes críticos (middleware, gateways).
- Detectar falhas via heartbeats e substituição automática de nós.

3.3 Consistência & Garantias de entrega

- Garantia: **At-least-once** com deduplicação; oferecer caminho para **effectively-once** via idempotência e dedupe pelo message_id.

- Ordem: garantir **ordem causal por conversa** (sequenciamento por conversation_id + per-conversation sequence number). Consistência eventual entre réplicas.

3.4 Latência

- Latência direcionada para comunicações em tempo real: **<200ms** para caminhos internos (cliente → front → enqueue), dependendo da localização geográfica; entrega a endpoints externos sujeita a latência do provider de cada canal.

3.5 Throughput

- Por nó: projetar para milhares de mensagens/s; escalar horizontalmente via particionamento.

3.6 Armazenamento de arquivos

- Aceitar uploads até **2 GB** por arquivo. Técnica: chunked upload com resumable protocol (ex.: similar ao tus, ou multipart upload S3).
- Arquivos armazenados em Object Storage (S3-compatible). Podem usar o mini-io como sistema de armazenamento. Metadados em DB.

3.7 Operacional / Observabilidade

- Monitoramento (Prometheus), alertas (Alertmanager), dashboards (Grafana).
- Tracing distribuído (OpenTelemetry) para investigar latência/falhas.
- Logs estruturados e centralizados (ELK/EFK).
- Métricas chave: mensagens/s, latência de entrega, taxa de erro connectors, utilização de disco, throughput de object storage.

3.9 Extensibilidade/Manutenibilidade

- Clean interface para adapters; documentação para desenvolver novos connectors.
- Versionamento da API (v1/v2).
- Gerar documentação da api usando Swagger/OpenAPI

4. Arquitetura proposta (alto nível)

Componentes principais:

1. **API Gateway / Ingress** (stateless): autenticação, rate limiting, TLS termination.
2. **Frontend Service (Stateless)**: endpoints REST/gRPC, validações, enfileira eventos.
3. **Message Broker / Stream (durável)**: ex.: Kafka — armazena eventos de mensageria para processamento assíncrono e replay.
4. **Workers / Router Services**: consomem o broker e executam lógica de roteamento e entrega aos connectors; garantem retry/dedup.
5. **Connectors / Channel Adapters** (pluggable): integradores com WhatsApp Business API, Instagram Graph API (Direct), Telegram Bot API, Messenger, etc.
6. **Metadata DB** (consenso/coordenação): etcd / CockroachDB para configurações, mapa de usuários → canais.
7. **Message Store (persistência)**: banco distribuído otimizado para escrita e leitura (ex.: mongoDB, Cassandra/ScyllaDB) — guarda metadados e texto.
8. **Object Storage**: S3- mini-io, compatible para arquivos grandes.
9. **Notification / Push Service**: para push mobile / websockets.
10. **Presence Service**: rastreia quem está online; usado por Router para decidir push vs persistência.
11. **Admin & Monitoring**: dashboards, logs, tracing.

Fluxo simplificado de envio:

- Cliente → API Gateway → Frontend Service valida e grava evento no Kafka → Router Worker consome → persiste metadados no Message Store e faz put do arquivo no Object Storage (se aplicável) → envia ao destinatário local (push/websocket) ou chama Connector para plataformas externas → atualiza status (DELIVERED, READ) via callbacks/webhooks.

5. Decisões técnicas fundamentais (sugestões)

5.1 Mensageria & Buffering

- Use **Apache Kafka** (ou equivalente) como backbone de eventos: alta taxa, durabilidade, replay.
- Particionamento por conversation_id para preservar ordem causal dentro de uma conversa.

5.2 Armazenamento de mensagens

- **MongoDB / CassandraDB** para mensagens: escrita rápida, escala horizontal, replicação eventual.
- Usar TTL configurável para mensagens se desejado.

5.3 Arquivos grandes

- Implementar a sua estratégia de replicação e balanceamento de carga.
- Metadados (URL, checksum, size, chunk-manifest) persistidos no DB.

5.4 Connectors para plataformas externas

- Cada connector roda como serviço independente (autoscalável).
- Padronizar interface: sendText(dest, payload), sendFile(dest, fileReference), onWebhookEvent().
- Retries exponenciais e circuit-breaker por connector.

5.5 Consistência e idempotência

- message_id UUIDv4 fornecido pelo remetente ou gerado pela API; workers deduplicam.
- Para exactly-once sem E2E: dedupe + idempotent writes; aceitar at-least-once entrega para canais externos.

5.6 Observability

- Insira spans OpenTelemetry em todo o caminho; logs correlacionados por trace_id e message_id.

6. API pública (exemplos de endpoints REST / gRPC)

6.1 Autenticação

POST /auth/token
Body: { "client_id": "...", "client_secret": "..." }
Resposta: { "access_token":"...", "expires_in":3600 }

6.2 Conversas

POST /v1/conversations
Body: { "type": "private"|"group", "members": ["userA","userB",...], "metadata": {} }
GET /v1/conversations/{conversation_id}/messages?since=...

6.3 Enviar mensagem

POST /v1/messages
Body:

{

"message_id":"<uuid>",

"conversation_id":"<conv>",

"from":"userA",

"to":["userB"],

"channels":["whatsapp","instagram"], // or ["all"]

"payload":{"type":"text","text":"Olá!"},

"metadata": {"priority":"normal"}

}

Resposta: { "status":"accepted", "message_id":"<uuid>" }

6.4 Upload de arquivo (resumable)

POST /v1/files/initiate -> retorna upload_url (presigned) e file_id
PATCH/PUT upload chunks to presigned URL
POST /v1/files/complete { "file_id": "...", "checksum": "..." }

6.5 Delivery / Read callbacks (webhooks)

Registrar webhook via: POST /v1/webhooks
Callback payloads deverão incluir message_id e status.

7. Requisitos de teste / validação (e demonstrações obrigatórias)

- **Carga**: demonstração de throughput (ex.: 100k msgs/min — ajustar para escala do curso) usando ferramentas (k6, Gatling).
- **Falhas controladas**: derrubar nós de workers / Kafka / DB e demonstrar failover / rebalancing sem perda de mensagens (ou perda aceitável documentada).
- **Testes de entrega cross-channel**: ex.: enviar de WhatsApp para Instagram Direct e demonstrar a transição e callbacks.
- **Upload/download de arquivos**: enviar arquivo de ~1.8 GB e demonstrar chunking/resume.
- **Escalabilidade horizontal**: adicionar nós em tempo de execução e mostrar aumento de capacidade.
- **Observabilidade**: dashboards mostrando métricas e tracing de fluxo de mensagens.

9. Entregáveis, milestones e formato de entrega

Entregáveis finais (por grupo):

1. Código-fonte (repositório) com README e scripts de deploy (K8s manifests ou docker-compose para POC).
2. Documentação da API (OpenAPI / proto).
3. Relatório técnico (máx 15 páginas) cobrindo arquitetura, decisões, testes e resultados.
4. Script / instruções para demo: 1 cenário básico (chat interno), 1 cenário cross-platform (WhatsApp→Instagram), e 1 cenário de stress.
5. Dashboards e logs de execução das cargas (links ou screenshots).
6. Vídeo curto da demonstração opcional (5–10 min).

Milestones sugeridos:

- Semana 1-2: Design da arquitetura, modelagem de dados, esqueleto do projeto.
- Semana 3-4: Implementar API básica + Kafka + worker simples + persistência de texto.
- Semana 5-6: Implementar object storage upload + connector mock (simula Instagram/WhatsApp).

- Semana 7-8: Testes de carga, monitoramento e relatório.

10. Critérios de Avaliação (sugestão de pesos)

- **Escalabilidade & Tolerância a Falhas (35%)** — arquitetura, provas (testes de carga/failover), sharding/replicação.
- **Corretude Funcional (20%)** — envio/recebimento, estados (SENT/DELIVERED/READ), persistência e arquivos.
- **Qualidade do Código & Design (15%)** — modularidade, documentação, uso de padrões e tests.
- **Integração Multiplataforma (15%)** — presença de connectors (ao menos 2: ex.: mock + Telegram) e demonstração cross-channel.
- **Observabilidade e Operação (10%)** — métricas, tracing, logs e scripts de deploy.
- **Relatório & Demonstração (5%)** — clareza do relatório e demo funcional.

11. Sugestões práticas (implementação para estudantes)

- **Começar com POC mínimo:** message flow local (API → worker → DB), depois adicionar Kafka e object storage. Use mocks para connectors no início.
- **Arquitetura by steps:** construir camadas e testar cada uma.
- **Reutilizar libs/standards:** resumable upload (tus or S3 multipart), OpenAPI para docs.
- **Limitar escopo realista:** exigir integração com *um* canal real (ex.: Telegram) e *outros* simulados para simplificar dependências comerciais (p.ex. WhatsApp Business API tem custos/limitações).

12. Exemplos de desafios técnicos que merecem nota extra

- Implementar **sharding dinâmico** por conversation_id com re-sharding sem downtime.
- Implementar **consensus** leve em metadados críticos (ex.: usar Raft via etcd).

- Implementar **file deduplication** por checksum para reduzir custo de armazenamento.
- Implementar **garantia de ordering causal** cruzando múltiplos servidores.

13. Checklist de avaliação técnica a anexar ao enunciado (para os grupos)

- API implementada com autenticação e docs.
- Mensagens persistidas e estados enviados corretamente.
- Arquivos até 2GB suportados (upload resumable + download).
- Connectors: pelo menos 1 real e 1 mock que demonstre cross-platform.
- Kafka (ou fila durável) + workers para roteamento.
- Banco distribuído para mensagens + object store para arquivos.
- Teste de carga documentado.
- Demonstração de failover e adição de nó sem perda de mensagens.
- Observabilidade (Prometheus + Grafana ou equivalente).