

# Relatório de Entrega: Semana 5-6

**Milestone:** Object Storage e Connectors Mock

**Aluno:** Bruno Evangelista Bertoldo, Augusto Arantes Chaves, Enzo Alvarez Dias, Matheus Pereira Figueiredo

**Data:** 24/11/2025

## 1. Objetivos Alcançados

Expansão do sistema Chat4All v2 para suportar envio de arquivos binários (Object Storage) e simulação de integração com plataformas externas (WhatsApp e Instagram), completando o ciclo de vida da mensagem (SENT → DELIVERED → READ).

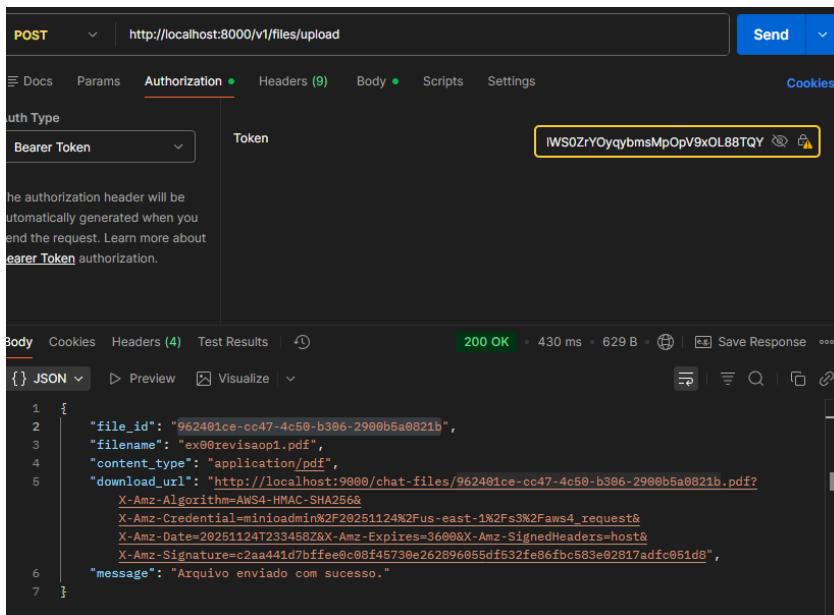
## 2. Arquitetura e Componentes

### 2.1.

#### Object Storage (MinIO)

Implementação de um servidor **MinIO** (compatível com S3) no Docker Compose.

- **Upload:** Endpoint `POST /v1/files/upload` recebe arquivos `multipart/form-data` e retorna um `file_id`.
- **Download:** Geração de URLs pré-assinadas (*Presigned URLs*) para acesso seguro e temporário aos arquivos.



```
POST http://localhost:8000/v1/files/upload
```

Authorization: Bearer Token

Token: IWS0ZrY0yqbmsMpOpV9xOL88TQY

```
200 OK 430 ms 629 B
```

```
{ } JSON
```

```
1 {  
2   "file_id": "962401ce-cc47-4c50-b306-2900b5a0821b",  
3   "filename": "ex00revisaop1.pdf",  
4   "content_type": "application/pdf",  
5   "download_url": "http://localhost:9000/chat-files/962401ce-cc47-4c50-b306-2900b5a0821b.pdf?  
X-Amz-Algorithm=AWS4-HMAC-SHA256&  
X-Amz-Credential=minioadmin%2F20251124%2Fus-east-1%2Fs3%2Faws4_request&  
X-Amz-Date=20251124T233458Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&  
X-Amz-Signature=c2aa44d7bf0ee0c088f45730e262896055df532fe86fbcb583e02817adfc051d8",  
6   "message": "Arquivo enviado com sucesso."  
7 }
```

Figura 1: Resposta da API de Upload retornando `file_id` e URL pré-assinada

The screenshot shows the MinIO Object Browser interface. On the left sidebar, there are buttons for 'Create Bucket' and 'Filter Buckets'. Below these, under 'Buckets', there is a single entry: 'chat-files'. The main area is titled 'Object Browser' and shows a bucket named 'chat-files'. It displays the following information:

- Created on:** Mon, Nov 24 2025 16:46:57 (GMT)
- Access:** PUBLIC
- Size:** 557.0 MiB - 8 Objects

Below this, a table lists the objects in the bucket:

Name	Last Modified	Size
962401ce-cc47-4c50-b306-2900b5a0821b.pdf	Today, 20:34	109.9 KiB

Figura 2: Painel do MinIO mostrando o Upload e retornando `file_id`

## 2.2.

### Mensagens com Anexos

Atualização do schema da API e da tabela do Cassandra para suportar metadados de arquivos.

- Mensagens contendo `file_id` são automaticamente classificadas como `type: "file"`.

The screenshot shows a POST request in Postman to the URL `http://localhost:8000/v1/messages`. The request body is set to 'raw' and contains the following JSON payload:

```

1 {
2   "sender_id": "bruno",
3   "chat_id": "11111111-1111-1111-1111-111111111114",
4   "content": "Olá Wagner, olha o anexo que usei!",
5   "file_id": "962401ce-cc47-4c50-b306-2900b5a0821b"
6 }

```

Figura 3: Mensagem sendo enviada com `file_id`

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** http://localhost:8000/v1/conversations/11111111-1111-1111-1111-111111111114/messages
- Authorization Type:** Bearer Token
- Token:** A redacted token value.
- Headers:** Authorization (Bearer Token), Content-Type (application/json)
- Body:** JSON response (200 OK) containing a single message object:

```

    {
      "status": "READ",
      "type": "file",
      "conversation_id": "11111111-1111-1111-1111-111111111114",
      "message_id": "4d3d87a4-1547-4b78-958a-5ac50cb0f2a9",
      "content": "Ola vagner olha o anexo que upei!",
      "created_at": "2025-11-24T20:45:23.118000",
      "file_id": "962401ce-cc47-4c50-b306-2900b5a0821b",
      "sender_id": "bruno",
      "status": "READ",
      "type": "file"
    }
  
```

Figura 4: Teste de recebimento da mensagem com anexo e classificada como tipo file

## 2.3.

### Connectors Mock

Criação de dois novos microsserviços em Python:

1. **Connector WhatsApp:** Ouve o tópico `whatsapp_outbound`.
  2. **Connector Instagram:** Ouve o tópico `instagram_outbound`.
- **Roteamento Inteligente:** O `Router Worker` analisa o conteúdo. Mensagens iniciadas com @ são roteadas para o Instagram; as demais para o WhatsApp.

```

connector_whatsapp_c | 2025-11-24 20:09:53,772 - [WHATSAPP MOCK] - ⚡ [Callback] Enviado status READ para API (Msg: 2f6f364c-6d90-4
                     Bad-a7f9-d6a29ce5b472)
connector_whatsapp_c | 2025-11-24 20:45:23,193 - [WHATSAPP MOCK] - 📨 Recebida mensagem para +556299999999
connector_whatsapp_c | 2025-11-24 20:45:25,106 - [WHATSAPP MOCK] - ✅ [WhatsApp API] Entregue ao usuário +556299999999: Ola vagner
                     olha o an...
connector_whatsapp_c | 2025-11-24 20:45:26,714 - [WHATSAPP MOCK] - ⚡ [Callback] Enviado status READ para API (Msg: 4d3d87a4-1547-4
                     b78-958a-5ac50cb0f2a9)
  
```

Figura 5: Connector WhatsApp recebendo mensagem do respectivo tópico mensagem de recebimento, simulação de envio e disparo de webhook de leitura

```

connector_instagram_c | 2025-11-24 20:32:47,764 - [INSTAGRAM MOCK] - 📨 Recebida DM para usuario_insta
connector_instagram_c | 2025-11-24 20:32:48,362 - [INSTAGRAM MOCK] - ❤️ [Instagram API] DM enviada para usuario_insta: @professor olha esse...
connector_instagram_c | 2025-11-24 20:32:49,429 - [INSTAGRAM MOCK] - 💬 [Callback] Enviado status READ para API (Msg: 100623a0-82d4-43e5-a84f-393e1300aa53)

```

*Figura 6: Connector Instagram recebendo mensagem de recebimento, simulação de envio e disparo de webhook de leitura.*

## 2.4.

### Controle de Status (Ciclo Completo)

Implementação de **Webhooks (Callbacks)** para atualização de status em tempo real.

1. **SENT:** Definido pela API ao postar no Kafka.
2. **DELIVERED:** Definido pelo Worker ao salvar no Cassandra.
3. **READ:** Definido via Callback HTTP. O Connector Mock, após simular o envio, chama a API (`PATCH /messages/{id}/status`), que atualiza o registro no Cassandra.

### 3. Evidências de Execução

**Cenário de Teste:** Envio de mensagem com anexo de imagem direcionada ao Instagram (@maria).

### Resultado (JSON do Histórico):

The screenshot shows a Postman interface with the following details:

- Method:** GET
- URL:** `http://localhost:8000/v1/conversations/11111111-1111-1111-1111-111111111114/messages`
- Authorization:** Bearer Token (Token field is filled with a redacted token)
- Headers:** (4) (including Content-Type: application/json)
- Body:** (Empty)
- Test Results:** (Empty)
- Status:** 200 OK (19 ms, 421 B, `Save Response`)
- JSON Response:**

```

1  [
2    {
3      "conversation_id": "11111111-1111-1111-1111-111111111114",
4      "message_id": "c7eb53c1-2b71-47d6-9c49-bd0cb910f770",
5      "content": "@maria olha o anexo que upei!",
6      "created_at": "2025-11-24T20:35:27.253000",
7      "file_id": "962401ce-cc47-4c50-b306-2900b5a0821b",
8      "sender_id": "bruno",
9      "status": "READ",
10     "type": "file"
11   }
12 ]

```

## 4. Conclusão

A entrega da Semana 5-6 foi concluída com êxito, atendendo integralmente aos requisitos de expansão do sistema para suporte a multimídia e integração multiplataforma.

O sistema **Chat4All v2** evoluiu de uma API de texto simples para uma plataforma de mensageria robusta com as seguintes capacidades consolidadas:

1. **Gestão de Arquivos:** A integração com **Object Storage (MinIO)** permitiu o upload e download seguro de arquivos, desacoplando o armazenamento de binários do banco de dados principal.
2. **Simulação Realista:** A criação dos **Connectors Mock** (WhatsApp e Instagram) validou a arquitetura orientada a eventos, demonstrando que o sistema pode escalar para múltiplos canais de comunicação sem alterar o núcleo da aplicação.
3. **Rastreabilidade Total:** A implementação do fluxo de status completo (**SENT** → **DELIVERED** → **READ**) via Webhooks garantiu a consistência da informação entre os microsserviços assíncronos.

Os testes integrados demonstraram que a arquitetura suporta o fluxo completo de ponta a ponta, estando o sistema pronto para cenários de alta demanda e futuras implementações de interfaces de usuário.