

DevLog

Comecei aqui!

1. Início do projeto e instalação das primeiras dependências.

Primeiro iniciei o projeto criando o repositório local + `npm init -y` para iniciar o projeto em NodeJS

Criei o arquivo do `.gitignore` para não commitar determinados arquivos com extensões

Instalei as seguintes dependências no meu projeto via `npm i`:

- express
- mongoose
- dotenv
- cors
- morgan
- prisma
- @prisma/client

Depois Instalei o `nodemon` para seu uso em desenvolvimento apenas.

1.1 Configuração básica para o NPM

Dentro do `package.json` criei os seguintes scripts para serem utilizados quando usar o `npm start` ou `npm dev`, dos quais iniciam o `index.js` do projeto

2. Estrutura base do projeto

Depois estruturei as pastas do projeto para ficar organizado e estruturado utilizando “organização por camada técnica”:

```
1  aeolus-day1/
2  |  node_modules/
3  |  src/                                # Estrutura do projeto
4  |  |  models/                         # Pasta para comportar os modelos do projeto
5  |  |  |  Camera.js
6  |  |  routes/                        # Pasta para comportar as rotas
7  |  |  |  cameras.js
8  |  |  services/                     # Pasta paa comportar os serviços
9  |  |  |  index.js                   # Index do Projeto
10 |  |  .env.example                   # Variaveis do sistema (Guia)
11 |  |  package.json                   # Configurações do Projeto
12 |  |  docker-compose.yml             # Docker Compose
13 |  |  README.md
14
```

3. Criação do primeiro Docker Compose

Altere o arquivo do `Docker Compose`, criando um serviço para o `Mongo`, usando as seguintes configurações:

Serviços que adicionei agora

- Versão: `mongo:noble`
 - Usei a versão segura mais recente, porque considere ser mais estável
- Nome do Container: `aeolus_mongo_server`
- Porta: `27017`
 - Liberei a porta pro sistema sendo a mesma que o mongo disponibiliza para a API
- Volume: `mongo-data:/data/db`

Volumes Gerais:

- `mongo-data:`

Subi o `docker-compose up -d` para poder trabalhar com um servidor mongo por enquanto

4. Criação do `.env` (Variáveis do Sistema)

Adicionei a porta e a uri do mongo padrão (porque não escolhi alterar nada, deixei tudo no default) dentro do `.env` para ter como variáveis.

Criei um arquivo `.env` deixando o projeto assim:

```
1  aeolus-day1/
2  |  node_modules/
3  |  src/                                # Estrutura do projeto
4  |  |  models/                         # Pasta para comportar os modelos do projeto
5  |  |  |  Camera.js
6  |  |  routes/                        # Pasta para comportar as rotas
7  |  |  |  cameras.js
8  |  |  services/                     # Pasta para comportar os serviços
9  |  |  |  index.js                  # Index do Projeto
10 |  |  .env                          # Variaveis reais do sistema
11 |  |  |  .env.example              # Variaveis do sistema (Guia)
12 |  |  package.json                 # Configurações do Projeto
13 |  |  docker-compose.yml          # Docker Compose
14 |  |  README.md
15
```

Descartado: Estrutura do Schema das Cameras usando mongoose

Criei um modulo para criar o Schema das Cameras usando o mongoose no script de `models Camera.js` e fiz o seguinte:

- Criei o Schema que os arquivos das cameras vão ter de acordo com o desafio
 - Nome
 - ID da camera
 - Zona
 - Endereço RTSP
 - Habilitei o timestamp.

- Exportei como um Módulo Chamado: "Camera"

Depois disso criei as rotas do CRUD no scrip cameras.js da seguinte forma:

- Importei o express, o modulo da Camera e o Router do express
- Criei o module export no fim do código pra não esquecer:

Isso aqui se manteve

Alterei no `.package` o `typer` de `CommonJS` para `module`, porque vi no video que era mais moderno utilizar desta forma e mais fácil para utilizar os pacotes e módulos.

5. Adição do Prisma

Importei o Prisma para o meu projeto via `npm install`:

- `prisma`
- `@prisma/client`

Iniciei o prisma

Com o Prisma instalado ele gerou um arquivo chamado `schema.prisma`.

Alterei ele adicionando que ao invés de usar o `postgree` por padrão ele vai usar o `Mongo`

Além disso dentro do arquivo criei o modelo do schema da camera.

Utilizando o Prisma temos a facilidade de criar os nossos modelos dentro deste Schema, gerarmos as variáveis pelo `prisma generate`, e conseguir utilizar toda a estrutura dos modelos como objetos para poder assim fazer consultas e manipulações de forma muito mais dinâmica e simples. Além disso o podemos através do prisma conseguir ler toda uma estrutura de BD já existente e

Criei o modelo da Camera no prisma:

```
13 // Schema - Camera
14 model Camera {
15   id          String    @id @default(auto()) @map("_id") @db.ObjectId
16   name        String
17   cameraID    String    @unique
18   zona        String
19   enderecoRTSP String?
20   createdAt   DateTime @default(now())
21   updatedAt   DateTime @updatedAt
22 }
```

6. Alterações do camera.js para usar o Prisma

Fiz a importação dos seguintes components:

- Router
- Prisma

Desenvolvi os seguintes endpoints para a API com relação as Cameras atendendo os requisitos do REST, que vão fazer seus registros e queries no MongoDB:

POST:

- Cadastro de 1 camera por vez

GET:

- Buscar todas as cameras
- Buscar camera pelo ID

PUT:

- Atualiza o registro de uma camera com base no ID para fazer a sua consulta

DELETE:

- Deleta o registro de uma camera com base no ID para fazer a sua consulta

No fim dele exportei o router como default

7. Criação do Index.js

Estruturei o index.js da seguinte forma.

Importei:

- Express
- dotenv
- Router da Camera

Carreeguei as variaveis do sistema com o DotEnv.

Criei um objeto para o Express

- Usei um middleware para interpretar o JSON

Instanciei a primeira rota sendo a rota de Cameras para fazermos as manipulações e os registros no mongo.

Defini a porta do projeto com uma variavel do .env

Criei a função para iniciar o servidor junto com a porta.

8. Alteração do Docker Compose

Derrubei o container antigo do mongo

Serviço Mongo DB:

- Ativei o modo de replicação no MongoDB, porque o prisma só permite ser utilizado se o mongo tiver essa função ativada.

PARTE IMPORTANTE AGORA:

TODAS AS VEZES QUE EU FOR CRIAR UMA NOVA INSTANCIA DO MONGODB ASSIM QUE FOR CRIADO O CONTAINER EU TENHO QUE RODAR O SEGUINTE COMANDO PARA ATIVAR O REPLICASET:

```
docker exec aeolus_mongo_server mongosh --eval "rs.initiate()"
```

8.1 Porque eu tenho que usar o replica set?*

Resposta do Copilot:

```
1  Ótima pergunta!
2  Você precisa usar o **replica set** porque o Prisma depende de recursos do MongoDB que só
   estão disponíveis quando o banco está rodando nesse modo – mesmo que seja um único nó.
3
4  ### Por quê?
5
6  - O Prisma usa **transações** para garantir consistência nas operações (por exemplo,
   criar e atualizar dados juntos).
7  - O MongoDB só permite transações (inclusive as mais simples) quando está em modo
   **replica set**.
8  - No modo "standalone" (padrão), o MongoDB não suporta transações multi-documento, e o
   Prisma exige isso para funcionar corretamente.
9
10 ### Resumindo:
11 - **Replica set** é o modo recomendado para produção e testes modernos.
12 - Mesmo com um só container, você pode ativar o replica set (não precisa de múltiplos
   servidores).
13 - Isso não deixa o banco mais pesado, só habilita recursos avançados.
14
15 **Fontes:**
16 - [Documentação oficial do Prisma sobre MongoDB]
   (https://www.prisma.io/docs/orm/reference/database-reference/supported-
   databases/mongodb#prerequisites)
17 - [Por que o MongoDB exige replica set para transações?]
   (https://www.mongodb.com/docs/manual/core/transactions/#replica-set-transactions)
18
```

9. Teste API via Postman

Testei a todos os endpo^{int} [...] da APU com postman, tudo ta funcionando normal!

Limpei o BD

Subi no ^{github}

Fui dormir

10. Restante das Tecnologias do projeto, no docker compose

Adicionei dentro do ^{Docker} compose:

- Clickhouse
- Zookeeper
- Kafka
- Minio

Deu um bug gigantesco porque o bitnami não presta, usei as imagens oficiais pro Kafka e pro Zookeeper, em latest, porque eram as unicas que estavam funcionando.

Subi o container atualizado e agora rodou!

Tive que fazer uns ajustes no zookeeper e no kafka, porque a bitnami não disponibiliza mais suas versões de forma gratuita, ai como tive que usar as imagens oficiais fiz o seguinte:

- alterei os environment do kafka

```
1      environment:
2
3      - KAFKA_BROKER_ID=1
4
5      - KAFKA_ZOOKEEPER_CONNECT=aeolus_zookeeper_server:2181
6
7      - KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://aeolus_kafka_server:9092
8
9      - KAFKA_LISTENERS=PLAINTEXT://0.0.0.0:9092
10
11     - KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
```

Tive que usar com = para referenciar as variaveis do sistema porque com dois pontinhos (:) não estava indo.

- Removi o `ALLOW_ANONYMOUS_LOGIN=yes` do zookeeper

Subi tudo no docker ai sim funcionou!

11. Adições no .env

Adicionei as variáveis no ponto env:

- URL do ClickHouse
- Broker do Kafka
- EndPoint do MinIO
- Chave de acesso e chave secreta do MinIO
- Bucket do MinIO

12. Instalação de dependencias

Instalei as dependencias via NPM

- kafkajs
- aws-client-s3
- axios

13. Conexão com o Clickhouse

Fui tentar fazer a conexão com o clickhouse, para ver se conseguia prosseguir, mas nada estava funcionando, dai eu baixei uma extensão chamada de `DBeaver Simulation`, tentei conectar e nada, quando eu fiz um curl no terminal, o clickhouse avisou que

tinha uma senha para eu colocar para poder entrar no servidor, daí então eu voltei lá no docker compose, adicionei os environment de:

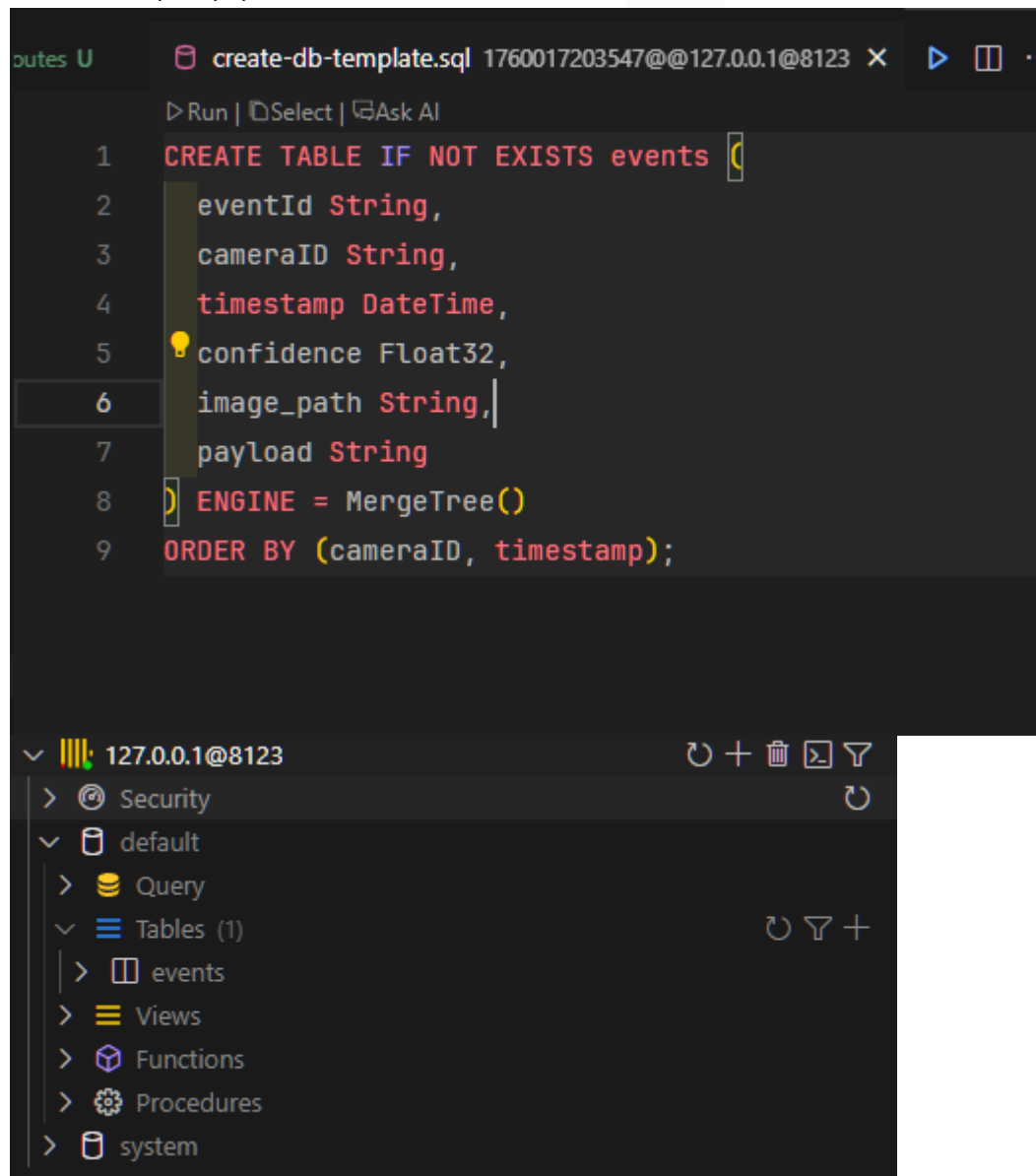
- Usuario: default
- Senha: admin

Subi o compose modificado

Fiz a conexão pela extensão para ter algo visual, e pronto consegui estabelecer conexão com o DB.

14. Criação da Tabela de eventos no clickhouse

Abri uma query para criar uma tabela de `events` dentro da extensão

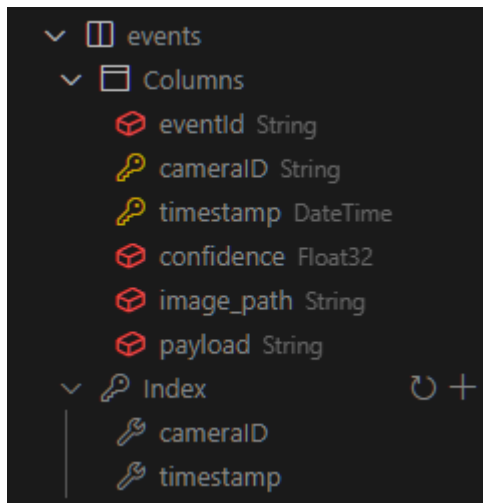


The screenshot shows a SQL editor interface for ClickHouse. The title bar indicates the file is 'create-db-template.sql' and the connection is '1760017203547@@127.0.0.1@8123'. The editor contains the following SQL code:

```
1 CREATE TABLE IF NOT EXISTS events (  
2     eventId String,  
3     cameraID String,  
4     timestamp DateTime,  
5     confidence Float32,  
6     image_path String,  
7     payload String  
8 ) ENGINE = MergeTree()  
9 ORDER BY (cameraID, timestamp);
```

Below the editor, a sidebar shows the database structure. The 'default' database is selected, and the 'events' table is listed under 'Tables (1)'.

Estrutura da tabela:



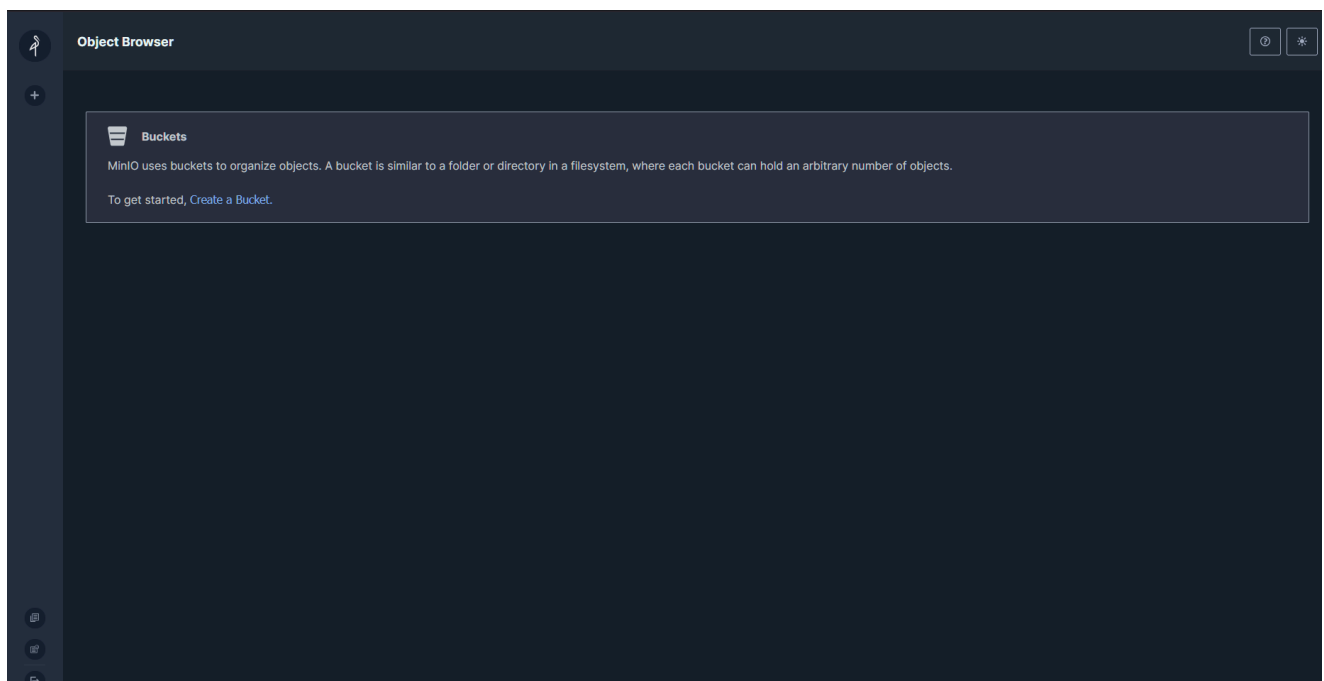
15. Configurações do MinIO

Agora depois de muito apanhar pra conseguir entende o do porque o MinIO não estava querendo redirecionar a porta do seu WebUI pra porta 9002 da minha maquina eu tive que fazer as seguintes alterações.

Por padrão a porta 9000 do minIO disponibiliza a API, so que o clickhouse tambem esta disponibilizando um serviço para a porta 9000, então deu um conflito enorme, e oque eu fiz:

- Redirecionei a porta 9000 do container para a minha porta 9002 onde esta vai ser a API (Back)
- Redirecionei a porta 9001 do container para minha porta 9001 tambem, onde esta vai ser o meu console na Web (Front)
- Como o WebUI não estava aparecendo de forma alguma eu coloquei dentro do command do serviço os parametros `-console-address' ":9001"` habilitando assim o webUi nesta porta

Subi o container atualizado, e funcionou de cara, pediu as chaves de acesso e funcionou



16. Criação do kafkaConsumer.js

Criei um arquivo novo na pasta de services chamado de kafkaConsumer.js

- Importei o KafkaJS
- Importei o S3Client e o PutObjectCommand
- Importei o Axios
- Comecei sua configuração
- Primeiro importei as seguintes dependências:
 - KafkaJS: Para consumir as mensagens do Kafka
 - Aws: Para salvar as imagens das cameras no MinIO
 - S3Client: Faz a conexão com um serviço S3, no caso o MinIO
 - PutObjectCommand: Faz o Upload de um arquivo para o Bucket.
 - Axios: Faz os envios das requisições HTTP
- Configurei o Kafka:
 - Passei o clientID como 'aeolus-app', para identificar
 - Passei o brokers como a variável do sistema que ta no .env KAFKA_BROKER
- Criei o consumidor do Kafka
 - Ele vai ser o responsável por receber as mensagens do tópico
 - Passei o groupID como aeolus-group
 - Vai identificar o grupo de consumidores (útil para balancear mensagens entre múltiplos consumers).

Depois fui fazer a função que vai rodar o RunConsumer

- Criei uma função assíncrona chamada de runConsumer
 - Fiz a Conexão do consumer com o consumer.connect
 - Fiz a inscrição do consumer com o consumer.subscribe
 - Topico inscrito: device-events
 - Defini o fromBeginning como falso, para que ele pegue apenas as mensagens novas
 - Fiz um loop de consumo com o consumer.run, com o eachMessage a cada mensagem recebida pelo consumer como um objeto, da qual ela sera desestruturada da seguinte:
 - topic : Nome do tópico de onde veio a mensagem
 - partition : Numero da partição

Descobri que o Kafka pode separar os tópicos em partições para escalar melhor

 - message : A mensagem recebida em si porem ela vira em formato de um objeto como um buffer (dado temporário para levar informações de um canto para outro)
- Logo apos foi desenvolvido um tratamento de erro;
- Onde o Try vai processar todo o evento e o catch vai só nos retornar uma mensagem de erro ao processar a imagem.

Abaixo vou destrinchar [...]  melhor oque o try esta fazendo:

- Criei um objeto para guardar os valores que peguei do `Buffer` (o atributo `message` que recebi), converti seus valores de forma padrão para `string` [...], pelo `ToString()`, converter seus valores para string, e desses valores para string os armazenei no objeto convertendo-os diretamente para um JSON.
- Defini um objeto para guardar a imagem de base64 para um buffer
- Defini a key do objeto no MinIO
- Salvei a imagem no MinIO passando as seguintes configurações para o `PutObjectCommand`:
 - O Bucket referenciei a variável do `.env` que eu já tinha
 - Passei a Key que eu tinha definido antes
 - No corpo do envio passei o objeto da imagem
 - No tipo do conteúdo foi definido como jpeg

O `PutObjectCommand` é um comando da `biblioteca` AWS SDK (usada também pelo MinIO) que serve para enviar (fazer upload) de um arquivo/objeto para um bucket S3.

No projeto estamos usando ele para salvar a imagem dentro do Bucket do MinIO.

- Após isso salvei os metadados do objeto que registrei as informações do buffer, em um outro objeto, passando sua estrutura:
 - `eventID`
 - `CameraID`
 - `Timestamp`
 - `confidence` = Nível de confiança do evento, normalmente um número `decimal` [...] entre 0 e 1, para indicar o quanto o sistema acredita que o evento detectado é verdadeiro
 - `image_path` = Caminho da imagem no MinIO
 - `payload` = Aqui é salvo um objeto JSON de todo o objeto do buffer para ter uma cópia de toda a estrutura original, mesmo que não tenhamos delegado campos para tais informações.
- Depois peguei este objeto do evento e enviei ele para o ClickHouse, via HTTP Post pelo axios:
 - Passei o endpoint (url do Clickhouse que esta no `.env`) + (Query para inserir na tabela events e formatar cada linha do body para o formato JSON)
 - No body da requisição passei o objeto que tinha salvo os dados do evento como um JSON
- No fim de tudo se ocorrer sem erros, ele confirma na tela que a mensagem foi processada e retorna o ID do evento (mensagem) registrado.

E no fim desse arquivo, eu executo esta função do `runConsumer`, e capturo os erros.

17. Mais dependências:

Instalei mais dependências via `npm install`

- `@aws-sdk/client-s3`
- `@aws-sdk/s3-request-presigner`

18. Bug no mongo que não estava funcionando

Logo após eu terminar o controller dos registros das cameras, eu tentei rodar algumas consultas no postman. so que eu vi que o banco de dados não estava

funcionando, “ele estava meio que fazendo uma consulta que não tinha fim”

Isso porque eu tinha que ligar o Mongo em replicantSet, porem todas as vezes que eu fosse fazer um down de tudo e subir novas imagens para testar o ambiente do zero eu teria que entrar dentro do CLI do container e ligar de forma manual este modo.

Dai eu pensei em elaborar algo que de ligasse o replicantSet forma automática sem ter que entrar no shell dele para fazer isso!

Porque não da pra passar o `rs.initiate()` no compose da image

Dai fiz o seguinte:

- Criei uma pasta chamada `config-services-docker` para conseguir adicionar configurações de init para algumas imagens no meu Docker Compose.
- Criei este arquivo chamado `mongo-init.js`

```
1 rs.initiate({ _id: "rs0", members: [ { _id: 0, host: "localhost:27017" } ] });
```

- Adicionei essa linha no volume do service do mongo no docker compose

```
12 - ./config-services-docker/mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js
```

TESTEI TODOS OS ENDPOINT NO POSTMAN AGORA E RODOU TUDO!

19. Criação do controller dos eventos, event.js

Comecei o desenvolvimento do events.js

Criei um arquivo chamado `/src/routes/events.js`, criei os seguintes endpoints:

- Get all
- Get event by id
- Get id da camera associada ao evento
- Get path_url pelo id do evento
- Get informações de uma camera pelo id da camera

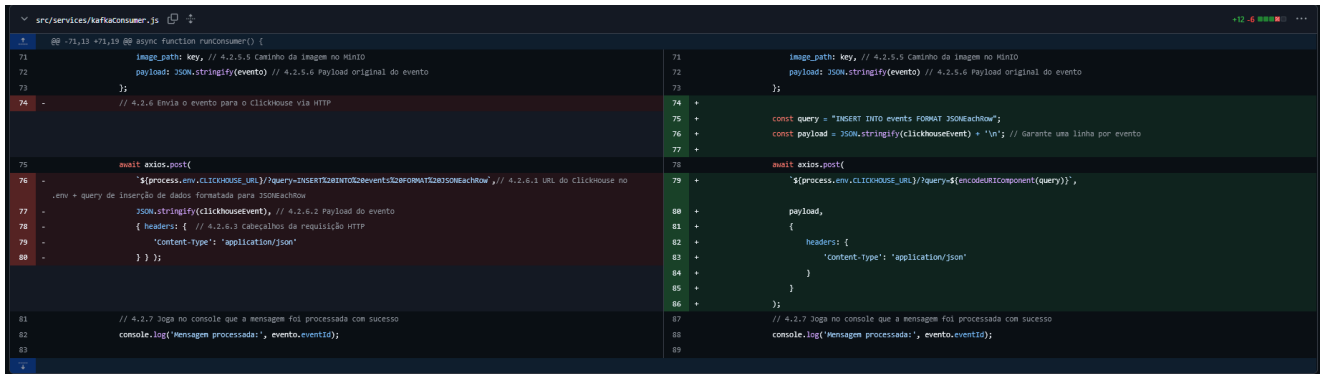
importação dos endpoints para o index

Nessa parte como eu só fui correndo por conta do tempo eu não tive tempo de salvar um registro mas, eu tenho o URL do commit que eu fiz:

<https://github.com/GeorgesBallister/Desafio-Aeolus/commit/7f41d1a14d7a90bce4d0fc7d6ab3be361decb539>

A estrutura inicial que eu tinha feito estava assim, isso depois me deu uma dor de cabeça tão desnecessaria.

20. Alteração da query do kafkaConsumer para um padrão mais organizado com o axios



21. Tive que trocar a imagem do kafka para uma mais antiga

Tive que fazer esta alteração por conta do consumer que não estava pegando eu acho
Versão do projeto:

- confluentinc/cp-kafka:3.3.1

Aqui eu também não lembro com exatidão porque já fazem varias horas, mas foi por conta que eu teria que fazer o que eu fiz de outra forma por conta das atualizações que o kafka teve, ai tive que catar essa versão antes da versão 3.5 do Kafka para fazer funcionar.

22. Alterei a .env do Url do Click house:

Tinha escrito da forma errada, mas agora ta certo:

- CLICKHOUSE_URL="http://localhost:8123/"

23. Fiz com que a tabela de events iniciasse automaticamente quando a imagem do ClickHouse subisse

Basicamente eu queria subir a imagem do ClickHouse com a tabela dos events de forma automatica, mas não tem como fazer isso diretamente pelo command do compose, dai eu descobri que tenho como sobrescrever um arquivo sql que faz o ClickHouse executa-lo assim que ligar, dando run, na query que estiver dentro dele.

Dai criei um arquivo chamado `/config-services-docker/init-table-clickhouse.sql`, dentro da mesma pasta que iniciava o script do mongo.

```

1 CREATE TABLE events (
2     eventId String,
3     cameraID String,
4     timestamp DateTime,
5     confidence Float32,
6     image_path String,
7     payload String
8 ) ENGINE = MergeTree()
9 ORDER BY eventId;

```

Depois alterei a parte de volume para puxar este arquivo no serviço do clickhouse:

```

24 volumes:
25     - clickhouse-data:/var/lib/clickhouse
26     -
      ./config-services-docker/init-events-table-clickhouse.sql:/docker-
      entrypoint-initdb.d/init-events-table.sql

```

Funcionou

24. Alterações no consumer do Kafka para referenciar .env

Fiz o import das variaveis do sistema:

```
import 'dotenv/config';
```

Correção na estrutura da post dentro do KafkaConsumer.js para registro do evento no ClickHouse

```

1  await axios.post(
2
3      `${process.env.CLICKHOUSE_URL}?query=${encodeURIComponent(query)}`,
4
5      payload,
6
7      {
8
9          headers: {
10
11              'Content-Type': 'application/json'
12
13          },
14
15      auth: {

```

```

16
17         username: 'default',
18
19         password: 'admin'
20
21     }
22
23 }
24
25 );

```

Alteração da estrutura de tratamento de erros:

```

1     } catch (error) {
2
3         console.error('Erro ao processar mensagem:', error);
4
5         if (error.response) {
6
7             console.error('Erro ClickHouse:', error.response.data);
8
9         }
10
11     }

```

25. Alteração do Package.json para adicionar o kafka junto ao start

O Kafka é um sistema que pelo menos aqui deve ser executado de forma independente via terminal, daí para facilitar a vida eu fui catar uma forma de conseguir iniciar o Kafka junto com o backend que eu desenvolvi.

Então encontrei o `concurrently` que é usada para executar múltiplos comandos ao mesmo tempo no terminal de forma paralela:

- Instalei: `npm install concurrently --save-dev`
- Adicionei o comando no `package.json`

```

1     "scripts": {
2
3         "start": "node index.js",
4
5         "kafka": "node ./src/services/kafkaConsumer.js",
6
7         "dev:all": "concurrently \"npm:start\" \"npm:kafka\""
8
9     },

```

26. Testando fluxo do sistema de novo

Exclui todos os containers e subi tudo de novo pra ver se tudo inicia sem problema

Iniciei o `npm dev:all`

Rodei o `image-generation-test.js`

Nada do Kafka receber as mensagens ainda

27. Mexendo no simulador de imagens para ir gerando imagens e eventos pro kafka

Mexendo na aplicação do simulador descobri que a sua porta para a API é a 3030. Não devo confundir com a no nosso backend que é 3000.

Altere o compose do compose do simulado adicionando o:

- `name: backend-simulador-aeolus`

28. Testando API do simulado com a minha API Logada

Testei denovo mas não foi, mas tamo com progresso.

ai descobri que as os envios das imagens que estava fazendo estava fazendo o seguinte:

- O meu Kafka não tinha o campo `imageBase64` (ou ele estava como `undefined`). Por isso, ao tentar fazer `Buffer.from(evento.imageBase64, 'base64')`, o kafka lançava um erro de `argumento inválido` no terminal

Dai eu Adicionei uma validação no consumer

```
1 // 4.2.2 Validação do campo imageBase64
2 if (!evento.imageBase64) {
3   console.error('Evento recebido sem imageBase64:', evento);
4   return;
5 }
6 // 4.2.2 Salvar imagem no MinIO convertendo de base64 para buffer
7 const imageBuffer = Buffer.from(evento.imageBase64, 'base64');
```

Rodei os 2 BackEnd de novo

Testei o post do simulador

E FOIIIIIIII O Meu backend ta pegando tudo!!!!

Mas tem um problema o endpoint `/eventos` da nossa API não ta pegando, continua só retornando um array vazio:

- `[]`

29. Testando mais uma vez com os containers do zero

Zerei os containers de novo e re-subi do zero

Continua o mesmo problema, os eventos não estão sendo inseridos no ClickHouse

30. Corrigindo:

Pelo que eu percebi apos tentar inserir manualmente um registro no ClickHouse, ele não aceita o formato do ISO para horário no campo do timestamp do registro do evento.

```

1  INSERT INTO events FORMAT JSONEachRow
2  {"eventId":"evt-teste-001","cameraID":"cam-teste-01","timestamp":"2025-10-13T22:00:00Z","confidence":0.97,"image_path":"cam-teste-01/evt-teste-001.jpg","payload":{"...}}"}

```

Ele me retornou o seguinte:

```

1  INSERT INTO events FORMAT JSONEachRow
2
3  Query id: e221e6fd-10a5-4d49-9b94-a69990153195
4
5  Ok.
6  Error on processing query: Code: 27. DB::Exception: Cannot parse input: expected '"' before: 'Z',"confidence":0.97,"image_path":"cam-teste-01/evt-teste-001.jpg","payload":{"...}}"}': (while reading the value of key timestamp): (at row 1)
7  : While executing ParallelParsingBlockInputFormat: data for INSERT was parsed from query. (CANNOT_PARSE_INPUT_ASSERTION_FAILED) (version 25.9.3.48 (official build))

```

Ele só aceita aquele tipo classico de data e hora: `YYYY-MM-DD HH:MM:SS`

Criei uma função básica para fazer a conversão dentro do KafkaConsumer

Não consegui salvar o registro porque estava com pressa, mas mais para a frente eu substituo essa estrutura por uma mais robusta.

A função basicamente só pegava o campo de Timestamp que ela recebia do evento que o simulador enviava, tratava ela para tirar o formato que vinha (`2025-10-13T22:00:00Z`) e transformava no novo

Rodei tudo e nada de registro aparecer dentro do ClickHouse

31. Correção pt2:

Adicionei uma linha de debug depois do post do axios no consumer:

```

1  console.log('Insert realizado com sucesso no ClickHouse'); //! DEBUG

```

Fiz uma inserção dentro do events manualmente e foi registrado:

```

4  INSERT INTO events FORMAT JSONEachRow
5  {"eventId":"teste-123","cameraID":"camera-001","timestamp":"2025-10-13 22:40:21","confidence":0.99,"image_path":"camera-001/teste-123.jpg","payload":{"foo":"bar"}}

```

events x

Search Results

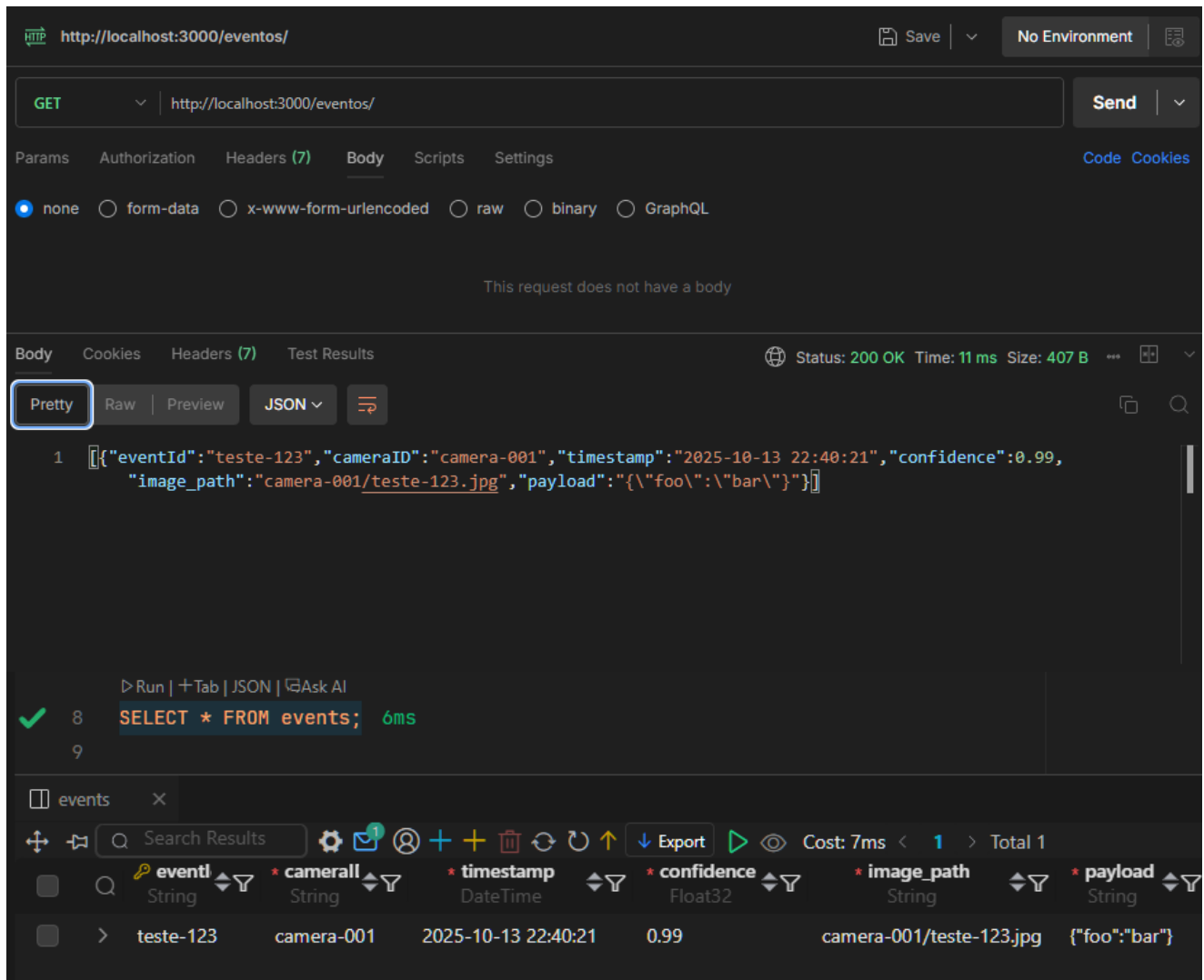
Cost: 6ms

```

INSERT INTO events FORMAT JSONEachRow {"eventId":"teste-123","cameraID":"camera-001","timestamp":"2025-10-13 22:40:21","confidence":0.99,"image_path":"camera-001/teste-123.jpg","payload":{"foo":"bar"}}

```


fiz um get na API e foi:



BELEZA, tentei agora rodar os dois sistemas juntos, com o simulador mandando registros via post para o backend. e agora o backend não ta mais pegando

32. Correção pt3:

Pelo que eu li o erro esta em que o simulador está enviando o campo como `image.base64` (dentro de um objeto `image`), mas o consumer espera `imageBase64` na raiz do evento.

Alterei o seguinte:

- Um tratamento de erro caso não receba a imagem em base64:

```
1 // 4.2.1 Converte os valores do buffer para string e depois para JSON e armazena na
  constante evento
2
3 const evento = JSON.parse(message.value.toString());
4
5 // 4.2.2 Validação do campo imageBase64 (aceita tanto na raiz quanto aninhado)
6
7 const imageBase64 = evento.imageBase64 || (evento.image && evento.image.base64);
8
9 if (!imageBase64) {
10
11     console.error('Evento recebido sem imageBase64:', evento);
```

```

12
13     return;
14
15 }
16
17 // 4.2.2 Salvar imagem no MinIO convertendo de base64 para buffer
18
19 const imageBuffer = Buffer.from(imageBase64, 'base64');
20
21 // 4.2.3 Define a chave do objeto no MinIO
22
23 const key = `${evento.cameraID}/${evento.eventId}.jpg`;

```

- Uma alteração no catch

```

1  } catch (error) {
2
3      console.error('Erro ao processar mensagem:', error);
4
5  }

```

Vi que as variáveis do sistema estavam as mesmas com a mesma chave (Sendo a chave errada) daí o :

- MINIO_ACCESS_KEY
 - MINIO_SECRET_KEY
- Ambas estavam com o valor de admin

Dai eu troquei seus valores para o mesmo que estava lá na construção do MinIO no docker compose:

- MINIO_ACCESS_KEY="minio"
- MINIO_SECRET_KEY="minio123"

33. Alteração na imagem do MinIO

Dai também vi que o bucket `events` não existia dentro do MinIO, ai para corrigir isso, alterei seu docker compose para iniciar com o bucket `events` já por padrão:

```

1  minio:
2      image: minio/minio:latest
3      container_name: aeolus_minio_server
4      command: server /data --console-address ":9001" # Habilita a interface web na porta
5      9001
6
7      environment:
8          - MINIO_ACCESS_KEY=minio
9          - MINIO_SECRET_KEY=minio123
10
11      ports:
12          - "9002:9000"    # S3 API
13          - "9001:9001"    # Console Web UI
14
15      volumes:
16          - minio-data:/data

```

```

17     entrypoint: >
18         sh -c "
19             minio server /data --console-address ':9001' &
20             sleep 5 &&
21             wget https://dl.min.io/client/mc/release/linux-amd64/mc -O /usr/bin/mc &&
22             chmod +x /usr/bin/mc &&
23             mc alias set myminio http://localhost:9000 minio minio123 &&
24             mc mb myminio/events || true
25             wait
26         "

```

Derrubei todos os containers, exclui as imagens e subi tudo de novo.

Quando subi os containers, vi que apareceu a seguinte mensagem:

```

1  INFO: WARNING: MINIO_ACCESS_KEY and MINIO_SECRET_KEY are deprecated.
2  Please use MINIO_ROOT_USER and MINIO_ROOT_PASSWORD

```

e também:

```

1  server: line 4: wget: command not found

```

Dai percebi que meus environment estava escrito errado e a imagem do MinIO não tinha o wget instalado, fiz o seguinte:

- Troquei o environment por:
 - MINIO_ROOT_USER=minio
 - MINIO_ROOT_PASSWORD=minio123
- e também troquei a linha do Wget por:
 - `curl -o /usr/bin/mc https://dl.min.io/client/mc/release/linux-amd64/mc &&`

Porem quando subi a imagem ele fazia que o bucket não existia

Dai tentei criar um “container auxiliar” para criar o bucket:

```

1  minio:
2      image: minio/minio:latest
3      container_name: aeolus_minio_server
4      command: server /data --console-address ":9001"
5
6      environment:
7          - MINIO_ROOT_USER=minio
8          - MINIO_ROOT_PASSWORD=minio123
9
10     ports:
11         - "9002:9000"
12         - "9001:9001"
13
14     volumes:
15         - minio-data:/data
16
17     minio-create-bucket:
18         image: minio/mc
19
20     depends_on:
21         - minio

```

```

22
23     entrypoint: >
24         sh -c "
25             for i in {1..10}; do
26                 /usr/bin/mc alias set myminio http://minio:9000 minio minio123 && break ||
sleep 3;
27             done &&
28             /usr/bin/mc mb myminio/events || true
29         "

```

E até que funcionou abri o ui e vi lá o bucket events dentro do MinIO, pulei um pouco de alegria, mas pensei comigo mesmo que era meio inútil criar um container inteiro pra criar o bucket no outro container, daí procurei na internet e vi que dá pra criar um simples script em JS mesmo inserindo o bucket já com as variáveis do sistema e ser muito mais fácil.

34. Criação do createBucket.js

Fiz o seguinte:

- Removi a estrutura toda do outro container e deixei o MinIO assim:

```

1  minio:
2      image: minio/minio:latest
3
4      container_name: aeolus_minio_server
5
6      command: server /data --console-address ":9001"
7
8      environment:
9          - MINIO_ROOT_USER=minio
10         - MINIO_ROOT_PASSWORD=minio123
11
12      ports:
13          - "9002:9000"
14          - "9001:9001"
15
16      volumes:
17          - minio-data:/data

```

- Criei o script `./src/services/createBucket.js` onde basicamente:
 - Ele inicia um loop de 10 tentativas de criar um bucket.
 - Ele tenta usar o `CreateBucketCommand` dentro do `s3.send` para criar o bucket de `events`.
 - A captura de falhas se dá por 2 formas:
 - Ele roda um `trycatch` tentando criar, se o código do erro que voltar for `BucketAlreadyOwnedByYou`, significa que o Bucket já foi criado e não precisa criar de novo.
 - Se depois das 10 tentativas nada for retornado, simplesmente ele retorna uma mensagem de erro

Adicionei para ele rodar junto com o npm do kafka e do `npm start` para começar tudo de uma vez, logo após subir os containers:

```

6   "scripts": {
7     "start": "node index.js",
8     "kafka": "node ./src/services/kafkaConsumer.js",
9     "minio": "node ./src/services/createBucket.js",
10    "dev:all": "concurrently \"npm:start\" \"npm:kafka\" \"npm:minio\""
11  },

```

Derrubei e excluí todos os containers e imagens, e subi tudo do zero! pra garantir que meu Código vai rodar.

Me esqueci de trocar os environment `.env` para o user e a senha do MinIO, troquei agora.

ACHO QUE NEM FAZ SENTIDO DEVO TA FICANDO LOUCO
realmente não tinha nada a ver

Alterei o `MINIO_ENDPOINT` para a porta 9002, porque vi que estava em 9000, mas a porta que eu estava redirecionando era a 9002 esse tempo todo.

35. Status de Agora

- Os containers sobem corretamente
- O npm dev:all inicia corretamente a API, o Kafka e cria/verifica bucket no Clickhouse.

36: Mais problema mas finalmente a solução de tudo + teste

Simplesmente a API esta dando código 400 para todas as mensagens que o simulador de cameras envia a ele, eles conseguem se interagir, mas a API não consegue registrar a imagem no minIO.

Calma ainda tem umas paradinhas que eu fiz, eu tentei testar novamente tudo e o kafka tava me retornando que os campos de eventId e o campo de cameraID estava como undefined, dai alterei o objeto do clickhouseEvent no Consumer para:

```

1  const clickhouseEvent = {
2
3    eventId: evento.eventId || evento.deviceId, // aceita ambos
4
5    cameraID: evento.cameraID || evento.deviceId, // aceita ambos
6
7    timestamp: evento.timestamp,
8
9    confidence: evento.confidence,
10
11   image_path: key,
12
13   payload: JSON.stringify(evento)
14
15 };

```

Em resumo depois de quase 2 horas testando teorias, eu finalmente consegui refatorar o kafkaConsumer, o que tive que fazer:

Refatorei partes do meu script do kafkaConsumer.

- Linhas de Debug
- Log do registro de forma melhor organizada e apresentada
- Import do prisma para validação de aprovação apenas de cameras registradas pelo /cameras
- Validação do campo imageBase64
- Alteração na estrutura da Key
- Alteração na estrutura dos metadados que serão salvos no ClickHouse
- Adição de uma query mais moderna para inserção do evento no ClickHouse pelo axios.
- Alteração da estrutura do post do Axios
 - Adção de campo de Autenticação
- Alteração no bloco de tratamento de erro `catch`
- Alteração da função de conversão do timestamp para o padrão aceito pelo ClickHouse

Refatorei o router do /eventos/:

- Criei uma função para evitar erro quando o nome da câmera ou do evento tem aspas.
- Agora toda busca por câmera ou evento usa essa função de segurança.
- Corrigi o nome do campo da imagem para ficar igual ao do banco: [image_path](#).
- Coloquei comentários para mostrar onde mudei o código.
- Troquei a ordem das rotas para não dar conflito quando buscar por ID ou por câmera.

37. Rezetei tudo

Zerei todos os containers e subi tudo de novo do zero

E finalmente tudo esta funcionando

Testei todos os EndPoints e esta funcionando

38. Documentando tudo que estava faltando?

Terminei a documentação dos arquivos faltantes

Testei tudo de novo, com um ambiente depois do `docker system prune -a`

Ta funcionando perfeitamente.

39 Estrutura atual da pasta do projeto!

```
1  desafio-Aeolus/
2  |  src/                                # Código-fonte principal do projeto
3  |  |  routes/                          # Rotas da aplicação
4  |  |  |  cameras.js
5  |  |  |  events.js
6  |  |  services/                       # Serviços e integrações externas
7  |  |  |  createBucket.js
8  |  |  |  kafkaConsumer.js
```

```
9 | L # (outros arquivos podem ser adicionados aqui)
10 | - prisma/ # Configuração do Prisma ORM
11 | | - schema.prisma
12 | - config-services-docker/ # Scripts de inicialização para serviços Docker
13 | | - init-events-table-clickhouse.sql
14 | | - mongo-init.js
15 | - Documentacao/ # Documentação do projeto
16 | | - DevLog/ # Dev logs e anotações de desenvolvimento
17 | - .env # Variáveis de ambiente reais do sistema
18 | - .env.example # Exemplo/guia de variáveis de ambiente
19 | - .gitignore # Arquivos e pastas ignorados pelo Git
20 | - docker-compose.yml # Orquestração dos serviços Docker
21 | - index.js # Ponto de entrada da aplicação
22 | - package.json # Configurações e dependências do Node.js
23 | - README.md # Documentação principal do projeto
```

40 Ações Finais

Dependências instaladas:

- inflight
- glob

Testes Adicionados no `__test__`