INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE PERNAMBUCO

Campus Igarassu

Tecnologia em Sistemas para Internet

MIGUEL GABRIEL BARBOSA DOS SANTOS

PROPOSTA DE INFRAESTRUTURA DE SOFTWARE PARA SERVIÇO DE PROCESSAMENTO DE CLICKSTREAM

Igarassu, Pernambuco

2026

MIGUEL GABRIEL BARBOSA DOS SANTOS

PROPOSTA DE INFRAESTRUTURA DE SOFTWARE PARA SERVIÇO DE PROCESSAMENTO DE CLICKSTREAM

Trabalho de conclusão de curso apresentado à Coordenação do curso de TSI do Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, como requisito para obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Dr. Alexandre Strapação Guedes Vianna

Igarassu, Pernambuco

2026

S237p Santos, Miguel Gabriel Barbosa dos

Proposta de infraestrutura de software para serviço de processamento de Clickstream. / Miguel Gabriel Barbosa dos Santos. — Igarassu, o autor,2026.
53f. : il.

Orientador: Prof. Dr. Alexandre Strapação Guedes Vianna

Trabalho de conclusão de curso (Tecnólogo em Sistemas para Internet). Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco. - Campus Igarassu. Coordenação do Curso de Tecnologia em Sistemas para Internet, 2026.

1. clickstream.. 2. Fluxos de dados. 3. Processamento de dados. I. Título. II. Vianna, Alexandre Strapação Guedes. III. Instituto Federal de Educação.

CDD 004

Catalogação na fonte: Bibliotecária : Maria Amanda Cabral CRB4 /1442

PROPOSTA DE INFRAESTRUTURA DE SOFTWARE PARA SERVIÇO DE PROCESSAMENTO DE CLICKSTREAM

Trabalho aprovado. Igarassu, 7 de maio de 2026.

_____

Professor Orientador
Alexandre Strapação Guedes Vianna

_____

Convidado 1

Liliane Alves do Nascimento Sales

_____

Convidado 2

Djalma Araújo Rangel

Igarassu, Pernambuco

2026

RESUMO

No cenário digital atual, a análise do comportamento do usuário tornou-se essencial para a otimização de produtos, mas a implementação de uma infraestrutura de *clickstream* para coletar e processar dados em tempo real é um desafio técnico relevante. O problema se encontra na complexidade e no alto custo, o que limita o acesso a essa tecnologia, especialmente para pequenas e médias empresas. Diante disso, o objetivo deste trabalho é propor o desenvolvimento de uma plataforma de serviço de registros e análise das interações dos usuários em ambientes digitais, conhecido como *clickstream*, que seja acessível e fácil de integrar, eliminando a necessidade de as empresas construírem uma infraestrutura do zero. Para atingir esse objetivo, a metodologia se baseou em três componentes principais: uma Infraestrutura para Gerenciamento de Tópicos Kafka, um Template para Integração com Aplicações *front-end* e uma Documentação Detalhada. Como resultado, a plataforma, denominada MG-ClickStream-Manager, foi desenvolvida para possibilitar a coleta, o processamento e a análise de eventos de clique, facilitando a tomada de decisões baseadas em dados. Conclui-se que o projeto indica potencial para simplificar a implementação e democratizar o acesso à análise de *clickstream*, permitindo que desenvolvedores se concentrem na análise e na tomada de decisão, sem a preocupação com a complexidade da infraestrutura, promovendo um ecossistema de desenvolvimento mais eficiente.

Palavras-chave: *fluxos de dados*. *clickstream*. Processamento em Tempo Real.

ABSTRACT

In today's digital landscape, analyzing user behavior has become essential for product optimization, but implementing a *clickstream* infrastructure to collect and process data in real time is a significant technical challenge. The problem lies in the complexity and high cost, which limits access to this technology, especially for small and medium-sized enterprises. Therefore, the objective of this work is to propose the development of a service platform for recording and analyzing user interactions in digital environments, known as *clickstream*, that is accessible and easy to integrate, eliminating the need for companies to build an infrastructure from scratch. To achieve this objective, the methodology was based on three main components: a Kafka Topic Management Infrastructure, a Template for Integration with *front-end* Applications, and Detailed Documentation. As a result, the platform, named MG-ClickStream-Manager, was developed to enable the collection, processing, and analysis of click events, facilitating data-driven decision-making. In conclusion, the project shows potential to simplify implementation and democratize access to *clickstream* analytics, allowing developers to focus on analysis and decision-making without worrying about infrastructure complexity, thus promoting a more efficient development ecosystem.

Keywords: *data stream processing*. *clickstream*. Real-Time Processing.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

API	Application Programming Interface
IoT	Internet das Coisas
JSON	JavaScript Object Notation
JWT	JSON Web Token
RAM	Memória de Acesso Aleatório
KPIs	Indicadores-Chave de Desempenho
Web	World Wide Web
UX	Experiência do Usuário
AWS	Amazon Web Services
CSV	Comma-Separated Values
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
SQL	Structured Query Language

SUMÁRIO

1 INTRODUÇÃO

No cenário atual, onde a análise de comportamento do usuário se tornou essencial para a otimização de produtos digitais, a implementação de *clickstream* surge como uma solução poderosa para coletar e visualizar interações em tempo real. Esse tipo de tecnologia permite que empresas acompanhem padrões de navegação, identifiquem pontos de melhoria e otimizem a experiência do usuário com base em dados concretos (LOSHIN, 2013).

No entanto, configurar uma infraestrutura eficiente para processar, armazenar e interpretar esses dados pode ser um desafio técnico significativo, exigindo conhecimentos avançados em arquitetura de sistemas distribuídos, processamento de eventos em tempo real e ferramentas de mensageria, além de um ambiente robusto para garantir a escalabilidade e confiabilidade da solução (KREPS, 2011).

Além disso, nem todas as equipes de desenvolvimento dispõem de recursos financeiros, tempo ou pessoal qualificado suficientes para projetar, implementar e manter esse tipo de infraestrutura, o que limita a adoção dessas soluções, especialmente em pequenas e médias organizações.

Nesse contexto, torna-se relevante investigar alternativas que reduzam a complexidade de implementação e facilitem a integração de soluções de *clickstream* em aplicações *web*, permitindo que desenvolvedores se concentrem na análise dos dados e na tomada de decisão, em vez de lidar com os desafios da infraestrutura subjacente.

1.1 Objetivo do Trabalho

Diante desse contexto, este trabalho propõe o desenvolvimento de uma plataforma de serviço de *clickstream*, com o objetivo de tornar essa tecnologia mais acessível e fácil de integrar a qualquer aplicação, sem a necessidade de montar um ambiente complexo do zero. A proposta busca oferecer uma infraestrutura pronta para uso em formato de serviço, que permita a qualquer desenvolvedor integrar a solução às suas aplicações um sistema de rastreamento de cliques de maneira padronizada.

1.2 Benefícios e Impacto do Projeto

Além de simplificar o processo técnico, este projeto também busca contribuir para a democratização do acesso à análise de *clickstream*, permitindo que pequenas e médias empresas possam usufruir dessa tecnologia sem um alto investimento inicial (GARTNER, 2022).

Atualmente, muitas organizações deixam de implementar rastreamento de eventos por conta da complexidade envolvida, o que impacta negativamente sua capacidade de tomar decisões baseadas em dados. Com essa solução, desenvolvedores poderão adicionar funcionalidades de *clickstream* a seus produtos com o mínimo esforço, focando apenas na análise e na tomada de decisão, sem se preocupar com a infraestrutura subjacente.

1.3 Estrutura do Trabalho

Este trabalho explora as tecnologias utilizadas, como Apache Kafka, NestJs, Vue. js e MongoDB, e discute os desafios e benefícios de implementar um sistema centralizado e padronizado para o processamento de eventos em tempo real (KREPS et al., 2011; RICHARDSON, 2018).

A proposta não apenas fornece um conjunto de ferramentas, mas também promove boas práticas de arquitetura, contribuindo para um ecossistema de desenvolvimento mais eficiente e acessível (FOWLER, 2012).

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, são explorados os conceitos e tecnologias essenciais para entender o processamento de fluxos de dados, com foco no contexto de *clickstream*. O processamento de fluxos *de dados* é cada vez mais importante para lidar com grandes volumes de dados em tempo real. Abordamos as definições e características desse paradigma, as arquiteturas utilizadas e as principais ferramentas, como Apache Kafka, Flink e Spark, que permitem a ingestão, análise e transformação de dados de forma eficiente e em tempo real.

2.1 Processamento de Fluxo de Dados

O processamento de fluxos de dados é uma abordagem computacional que visa tratar dados gerados em tempo real. Diferente da abordagem tradicional de batch processing, onde os dados são coletados, armazenados e processados posteriormente, o processamento de fluxos de dados permite a análise e o processamento das informações à medida que elas chegam. Isso possibilita decisões rápidas e reativas, fundamentais em áreas como saúde, logística, finanças e Internet das Coisas (IoT) (MOE; FADER, 2004).Nesse contexto, os sistemas de processamento de fluxo tratam os dados como um fluxo infinito de eventos, onde a computação deve ser realizada de forma incremental à medida que novos eventos surgem (KLEPPMANN, 2017).

O paradigma reflete a dinâmica do mundo digital moderno, no qual eventos ocorrem continuamente. Exemplos incluem cliques de usuários em sites, leituras de sensores, transações financeiras e interações em redes sociais. O processamento em tempo real agrega valor ao transformar informações brutas em insights acionáveis quase instantaneamente.

2.1.1 Conceitos Fundamentais

Aqui são expostos alguns conceitos fundamentais para entender o processamento de fluxos de dados.

Fluxo contínuo de dados: Os dados chegam de forma contínua, frequentemente em alta velocidade e em grandes volumes, exigindo arquiteturas escaláveis para processamento eficiente. Cada evento representa uma unidade de dados que descreve uma ação ou ocorrência, como cliques de usuários, leituras de sensores *IoT* ou logs de sistemas distribuídos. Esses eventos podem conter informações estruturadas ou semiestruturadas, necessitando de técnicas avançadas de ingestão, armazenamento e análise. Essa característica é essencial para aplicações que precisam de monitoramento em tempo real, como sistemas de tráfego, mercados financeiros, detecção de fraudes e análise de comportamento de usuários em plataformas digitais (MOE; FADER, 2004).

Latência de dados: Representa o intervalo entre a chegada do dado e a finalização de seu processamento, sendo um fator crucial para sistemas que exigem respostas em tempo real. Aplicações críticas, como detecção de fraudes, resposta a emergências médicas e monitoramento de infraestrutura, demandam latência mínima para garantir decisões rápidas e eficazes. Para otimizar esse tempo de resposta, ferramentas como Apache Flink e Spark Streaming empregam processamento em memória, paralelismo e particionamento inteligente, permitindo a análise e a tomada de decisão quase instantânea, mesmo em cenários de alto volume de dados (APACHE FLINK DOCUMENTATION, 2024).

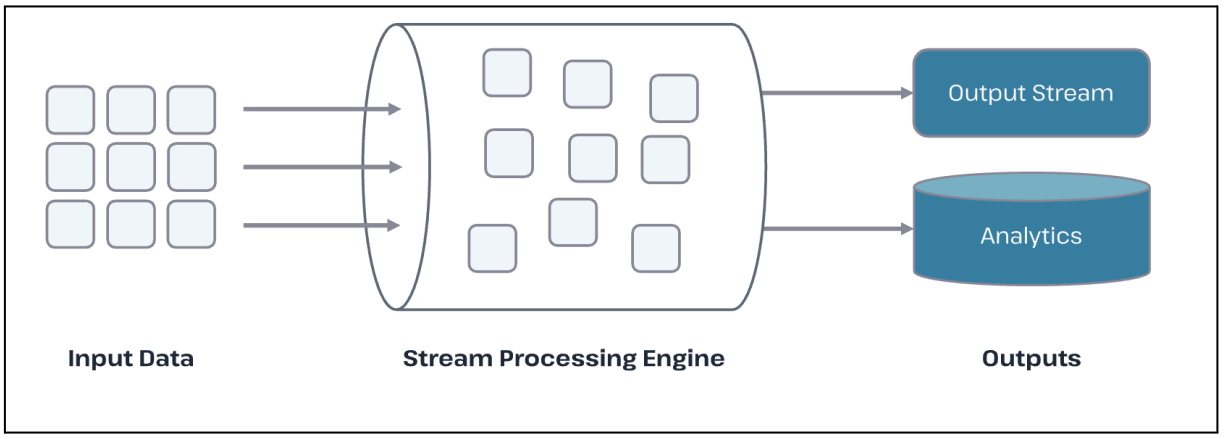
Estado contínuo dos dados: Sistemas de processamento de fluxos *de dados* mantêm um estado interno, acumulando informações ao longo do fluxo. Isso é crucial para cálculos complexos, como agregações e análises temporais, e para a detecção de anomalias com base em históricos. Técnicas como *checkpoints* garantem a consistência do estado, mesmo em sistemas distribuídos (Apache Kafka Documentation, 2024).

Escalabilidade de processamento: A capacidade de lidar com volumes crescentes de dados é alcançada por meio da escalabilidade horizontal, na qual novas máquinas ou clusters são adicionados ao sistema para distribuir a carga de processamento. Esse modelo permite que o sistema mantenha um alto desempenho sem comprometer a latência, mesmo diante de um aumento significativo no volume de dados.

2.1.2 Arquitetura de processamento de fluxos de dados

O processamento de fluxos de dados adota uma arquitetura modular com três componentes principais, tal arquitetura está ilustrada no Figura 1 e a seguir cada componente é detalhado individualmente. Essas estruturas evoluíram para suportar o processamento escalável e tolerante a falhas, muitas vezes fundamentadas em conceitos de arquiteturas como Lambda e Kappa (MARZ; WARREN, 2015).

Figura 1 - Exemplo de processamento de fluxo de dados



Fonte: *Hazelcast* (2024).

Provedores de Dados: As fontes de dados são responsáveis por gerar informações brutas que alimentam sistemas de processamento. Entre as principais estão sensores *IoT*, sistemas de logs, redes sociais, transações financeiras e aplicações web, cada uma com características específicas em termos de volume, frequência e formato dos dados gerados.

Essas fontes influenciam diretamente a taxa de ingestão e a necessidade de escalabilidade e processamento em tempo real. Por exemplo, sensores *IoT* podem enviar pequenos pacotes de dados em intervalos regulares, enquanto redes sociais geram grandes volumes de informações de forma imprevisível. Para lidar com essa diversidade, sistemas de processamento de fluxos utilizam técnicas como *buffering*, compactação e filtragem para otimizar a ingestão e análise dos dados de forma eficiente (MOE; FADER, 2004).

Pipeline de Processamento: O núcleo do sistema é a camada responsável pela ingestão, transformação e saída de dados, e desempenha um papel fundamental no processamento eficiente das informações. Nesta camada, ferramentas como Apache Kafka são utilizadas para gerenciar a ingestão de dados em grande escala, garantindo que as informações sejam recebidas de forma contínua e com alta disponibilidade, mesmo em ambientes distribuídos.

Após a ingestão, os dados são transformados e analisados por ferramentas como Apache Flink e Spark Streaming, que permitem realizar operações em tempo real, como agregações, filtragem, análises complexas e detecção de padrões. Essas ferramentas utilizam processamento paralelo e em memória, permitindo que grandes volumes de dados sejam processados rapidamente, mantendo a latência baixa. Além disso, elas oferecem suporte para técnicas avançadas de janelas de tempo (*windowing*) e tolerância a falhas, garantindo a consistência e a precisão das transformações e análises realizadas (Apache Flink Documentation, 2024).

Consumidores: São os sistemas finais que utilizam os dados processados para gerar informações úteis e permitir a tomada de decisões. Exemplos incluem *dashboards* de monitoramento, sistemas de alerta e modelos de aprendizado de máquina.

Dashboards de monitoramento visualizam métricas e *KPIs* em tempo real, ajudando equipes a acompanhar o desempenho de sistemas e a identificar problemas rapidamente. Sistemas de alerta detectam condições anormais, como falhas ou eventos críticos, e notificam os responsáveis para que possam tomar ações corretivas de maneira ágil. Já os modelos de aprendizado de máquina utilizam os dados processados para realizar previsões, classificações ou recomendações, muitas vezes aplicados em áreas como análise de fraudes, personalização de conteúdo e otimização de processos. Esses sistemas dependem de dados precisos e processados em tempo real para garantir a eficácia das decisões e ações automáticas.

2.1.3 Aplicações Práticas e Tecnologias Populares

O processamento de fluxos de dados é amplamente utilizado em cenários que exigem análise contínua de dados em tempo real, como monitoramento de sistemas, *IoT*, personalização de experiências de usuários e sistemas de recomendação. Essa abordagem permite processar e analisar grandes volumes de dados à medida que eles são gerados, proporcionando *insights* imediatos e a capacidade de responder rapidamente a eventos.

Tecnologias como Apache Kafka, Apache Flink e Google Dataflow são frequentemente adotadas nessas soluções devido à sua escalabilidade, flexibilidade e eficiência em ambientes distribuídos. Elas são capazes de lidar com grandes fluxos de dados em tempo real, distribuindo as cargas de processamento entre múltiplos nós e garantindo alta disponibilidade e resiliência. Essas ferramentas oferecem recursos avançados, como processamento em memória, paralelismo e tolerância a falhas, permitindo que aplicações complexas, como análises de dados em tempo real e aprendizado de máquina, sejam realizadas de forma eficaz e com baixa latência de dados (Apache Kafka Documentation, 2024; Apache Flink Documentation, 2024).

2.2 Clickstream

Clickstream refere-se ao registro e à análise detalhada dos cliques e interações dos usuários enquanto navegam por páginas da web ou utilizam aplicativos. Cada clique, rolagem ou interação realizada pelo usuário é registrado em tempo real, formando um fluxo contínuo de dados que mapeia o comportamento e as ações dentro do ambiente digital. Esses dados podem incluir não apenas cliques, mas também movimentos do mouse, tempo de permanência em páginas e sequências de navegação, fornecendo uma visão detalhada sobre como o usuário interage com o conteúdo (LOSHIN, 2013). Historicamente, o *clickstream* evoluiu de simples contadores de acessos para modelos complexos de análise de probabilidade de compra e navegação sequencial (BUCKLIN; SISMEIRO, 2009). Um

exemplo de ferramentas utilizadas para coleta e análise de clickstream é o uso conjunto do Google Analytics e do Adobe Analytics.

Esses dados são analisados para identificar preferências de navegação, padrões de comportamento e tendências de interação, permitindo que empresas e plataformas otimizem a experiência do usuário (UX). Além disso, a análise do *clickstream* é essencial para personalizar conteúdos, melhorar a eficácia de campanhas de marketing e fornecer recomendações mais precisas. A partir dessas informações, é possível também realizar segmentações mais assertivas e criar estratégias de engajamento, aumentando a retenção de usuários e a conversão em plataformas digitais (MOE; FADER, 2004).

2.2.1 Aplicações do clickstream

A seguir apresentamos algumas aplicações recorrentes de *clickstream*:

E-commerce: O *clickstream* permite compreender o comportamento do usuário desde a primeira interação até a finalização ou abandono de um pedido. Com isso, é possível identificar produtos mais populares, ajustar a organização do site e criar experiências de compra personalizadas, aumentando a conversão e reduzindo o abandono de carrinho. Além disso, é possível analisar a jornada do usuário para otimizar cada etapa do processo de compra.

Marketing Digital: Através da análise de dados de interação, o *clickstream* monitora campanhas publicitárias em tempo real, permitindo avaliar a eficácia de anúncios, segmentar públicos de forma mais precisa e ajustar estratégias de *marketing* com base nos resultados obtidos. Isso possibilita uma alocação mais eficiente do orçamento publicitário e uma abordagem mais assertiva em campanhas futuras.

Otimização de Websites (SEO/UX): O *clickstream* oferece insights valiosos sobre a experiência do usuário (UX), identificando pontos de frustração e interações problemáticas no site. Com base nesses dados, é possível ajustar interfaces e melhorar a navegação para aumentar a retenção de usuários e melhorar o desempenho do site em termos de tempo de permanência, taxas de conversão e satisfação geral.

Business Intelligence e Data Analytics: A análise do *clickstream* fornece padrões de navegação e comportamento do usuário, fundamentais para a tomada de decisões estratégicas e operacionais. Esses dados são utilizados para ajustar produtos, prever tendências e otimizar processos, apoiando áreas como desenvolvimento de produtos, atendimento ao cliente e gestão de operações, garantindo uma atuação mais eficiente e alinhada às necessidades dos consumidores.

2.2.2 Tecnologias e Ferramentas

Existem diversas ferramentas e voltadas para a construção de infraestruturas de processamento de fluxos de dados, a seguir apresentamos as mais populares no mercado:

O Apache Kafka é uma plataforma de mensageria distribuída projetada para a ingestão, armazenamento e transmissão de grandes volumes de eventos de *clickstream* com baixa latência de dados e alta disponibilidade. Ele permite a comunicação assíncrona entre produtores e consumidores de dados, garantindo escalabilidade e resiliência na entrega de eventos. Kafka pode ser integrado a diversas ferramentas de análise e processamento, como Flink e Spark, para análise avançada de dados (APACHE KAFKA, 2025).

O Apache Flink é um framework de processamento de fluxo de dados em tempo real, altamente eficiente e com suporte nativo a estados distribuídos. Ele permite a detecção de padrões complexos, execução de agregações contínuas e modelagem preditiva sobre fluxos de eventos. Diferente de abordagens baseadas em micro-batches, o Flink opera com um modelo de processamento contínuo e de baixa latência de dados, sendo ideal para aplicações que exigem respostas instantâneas, como recomendações personalizadas e monitoramento de fraudes (APACHE FLINK, 2026).

O Apache Spark Streaming complementa essas soluções ao permitir o processamento de fluxos de dados em *micro-batches*. Embora não seja um processamento estritamente em tempo real como o Flink, ele oferece alta performance para aplicações que podem tolerar pequenas latências. Com

integração nativa ao ecossistema Spark, ele facilita análises avançadas por meio de *machine learning* e computação distribuída (APACHE SPARK, 2026).

Além dessas tecnologias *open-source*, diversas plataformas em nuvem oferecem serviços otimizados para processamento de fluxos de dados. A AWS (*Amazon Web Services*) disponibiliza o AWS Lambda (AMAZON WEB SERVICES, 2026a), que permite a execução de funções *serverless* para processamento de eventos em tempo real, eliminando a necessidade de gerenciar servidores, e o Amazon Kinesis (AMAZON WEB SERVICES, 2026b), uma plataforma completa para ingestão, processamento e análise de fluxos de dados em tempo real, suportando integração com diversas fontes de dados e ferramentas analíticas. No ecossistema da Microsoft, o *Azure Stream Analytics* (MICROSOFT, 2026) é um serviço gerenciado para análise contínua de streams de dados provenientes de diversas fontes, como *IoT*, logs de aplicações e eventos de *clickstream*, utilizando SQL-like queries para facilitar a implementação de análises em tempo real. Já no Google Cloud, o Google Cloud Pub/Sub (GOOGLE CLOUD, 2026a) funciona como um sistema de mensageria distribuída projetado para transmissão assíncrona e escalável de eventos, permitindo integração entre serviços, enquanto o *Google Cloud Dataflow* (GOOGLE CLOUD, 2026b), baseado no Apache Beam, possibilita processamento distribuído de fluxos de dados em tempo real e em lote, com suporte a escalabilidade automática e integração com outras ferramentas do Google Cloud.

Essas tecnologias e serviços permitem que empresas coletem, processem e analisem dados de *clickstream* de maneira eficiente, possibilitando a geração de insights em tempo real e aprimorando a experiência do usuário.

2.3 Tecnologias Utilizadas

Nesta seção são descritas as principais tecnologias utilizadas no desenvolvimento deste trabalho. A escolha dessas tecnologias foi fundamentada em critérios como escalabilidade, eficiência, segurança, licença e adequação ao contexto da aplicação.

Apache Kafka: Para a pipeline de processamento, foi adotado o Apache Kafka, pois é uma ferramenta que contempla as necessidades do projeto para as tarefas de

ingestão de eventos em tempo real. Além disso, trata-se de uma solução popular, consolidada no mercado e *open source*, o que garante a robustez necessária para a escalabilidade e a confiabilidade do sistema proposto sem elevar os custos de licenciamento (APACHE KAFKA, 2025).

NestJS (Back-end): O NestJS é um framework progressivo para desenvolvimento de aplicações back-end com Node.js, baseado em TypeScript. Ele adota princípios como modularização, injeção de dependências e arquitetura baseada em camadas, tornando a manutenção e escalabilidade do código mais eficientes. Além disso, o NestJS facilita a implementação de práticas como programação orientada a eventos e arquitetura baseada em microsserviços, o que o torna ideal para aplicações distribuídas e de alta disponibilidade (NESTJS, 2025). Essa modularidade é essencial para o isolamento de domínios em sistemas distribuídos, permitindo que cada serviço evolua de forma independente (NEWMAN, 2021).

MongoDB (Banco de Dados NoSQL): O MongoDB é um banco de dados NoSQL orientado a documentos, projetado para armazenar dados de forma flexível e escalável. Diferente dos bancos relacionais tradicionais, o MongoDB utiliza um modelo baseado em *JSON*, permitindo armazenamento dinâmico de dados, consultas ágeis e indexação avançada. Além disso, o banco oferece suporte a *sharding* e replicação, garantindo maior escalabilidade e disponibilidade para aplicações que precisam lidar com grandes volumes de dados (MONGODB, 2025). O uso de bancos NoSQL como o MongoDB justifica-se pela necessidade de persistência poliglota, onde a flexibilidade do esquema permite evoluções rápidas no modelo de dados (SADALAGE; FOWLER, 2012).

JWT (*JSON Web Token*): O *JSON Web Token* (JWT) é um mecanismo de autenticação baseado em tokens assinados digitalmente. Ele é utilizado para garantir a segurança das APIs, permitindo que apenas usuários autenticados acessem *endpoints* sensíveis. O JWT funciona através de um *token* criptografado, que armazena informações como ID do usuário e permissões de acesso, reduzindo a necessidade de consultas constantes ao banco de dados. Seu uso é amplamente adotado em arquiteturas serverless e microsserviços devido à sua leveza e eficiência (JWT, 2025).

Docker e Rancher Desktop (Gerenciamento de Containers): O Docker é uma plataforma de virtualização baseada em *containers,* que permite empacotar aplicações e suas dependências de forma isolada do sistema operacional. Essa abordagem facilita a portabilidade, escalabilidade e gerenciamento de ambientes entre desenvolvimento e produção. O Rancher Desktop, por sua vez, é uma ferramenta que auxilia na execução e gerenciamento de *containers* Docker localmente, simplificando a administração dos serviços de *back-end* e mensageria (DOCKER, 2025; RANCHER, 2025).

JSON-Server (Simulação de API): O *JSON-Server* é uma ferramenta leve que permite simular *APIs REST* a partir de um arquivo *JSON*. Ele é amplamente utilizado para testes de integração e desenvolvimento de *front-end* desacoplados, pois permite que as requisições *HTTP* sejam manipuladas sem a necessidade de um *back-end* real. Essa abordagem facilita o desenvolvimento de aplicações *front-end* enquanto a API principal ainda está em construção (*JSON-SERVER*, 2025).

TypeScript (Linguagem de Programação): O *TypeScript* é uma linguagem de programação que estende o JavaScript adicionando tipagem estática e recursos avançados como interfaces, decorators e módulos. Essa abordagem melhora a segurança do código, reduzindo erros em tempo de execução e aumentando a produtividade no desenvolvimento. Além disso, o TypeScript oferece suporte avançado para autocompletar, refatoração e verificação de tipos, tornando-se a escolha ideal para projetos de grande escala (TYPESCRIPT, 2025).

3 METODOLOGIA

Este capítulo apresenta o percurso metodológico adotado para o desenvolvimento e a validação do MG-ClickStream-Manager. A pesquisa estrutura-se desde a definição da abordagem experimental até o detalhamento das estratégias de desenvolvimento e avaliação da solução. São descritos o ambiente controlado utilizado para os testes, os métodos de geração de dados para simulação de carga e as abordagens de validação manual e automatizada. Ademais, detalha-se a realização de um estudo de caso, método escolhido para avaliar se a infraestrutura proposta é capaz de fornecer o serviço de *clickstream* conforme os objetivos estabelecidos.

3.1 Tipo de Pesquisa

Este trabalho caracteriza-se como uma pesquisa aplicada, com abordagem experimental, uma vez que propõe o desenvolvimento de uma solução computacional e sua avaliação em um ambiente controlado. O objetivo é validar, na prática, a viabilidade de uma infraestrutura de processamento de *clickstream* baseada em tecnologias de processamento de dados em tempo real de forma acessível, buscando o uso de tecnologias de código aberto e com simples implementação por parte do usuário.

3.2 Estratégia de Desenvolvimento

A estratégia adotada consiste no desenvolvimento de um protótipo funcional denominado MG-ClickStream-Manager, estruturado com base em uma arquitetura de microsserviços. A solução foi projetada com foco em escalabilidade, modularidade e processamento em tempo real, utilizando tecnologias como Apache Kafka, NestJS e MongoDB.

Além disso, a estratégia contemplou a criação de um *middleware* específico para a captura de eventos no lado do cliente e um Gerador de Dados para simular

cargas de trabalho, garantindo que o protótipo não apenas processe os dados, mas ofereça uma solução de ponta a ponta para o monitoramento de interações.

O desenvolvimento foi conduzido de forma incremental, contemplando as seguintes etapas:

(i) levantamento dos requisitos da solução;

(ii) definição da arquitetura do sistema;

(iii) implementação dos componentes principais;

(iv) integração entre os módulos;

(v) realização de testes e validação da solução.

3.3 Ambiente Experimental

A avaliação da solução foi realizada em um ambiente controlado, utilizando uma aplicação simulada capaz de reproduzir interações típicas de usuários em sistemas web. Esse ambiente permite a geração e o monitoramento de eventos de clique, possibilitando a análise do comportamento do sistema em condições próximas às de uso real.

3.4 Avaliação da Solução

A avaliação da solução foi conduzida por meio de um estudo de caso, no qual foi verificada a capacidade de integração do MG-ClickStream-Manager a uma aplicação web. Para isso, foi seguido um tutorial estruturado de integração, permitindo validar se o processo pode ser realizado de forma funcional por um desenvolvedor. Adicionalmente, foi realizado um teste de carga utilizando dados simulados, com o objetivo de avaliar o comportamento da solução sob volume elevado de eventos, bem como sua capacidade de processamento e armazenamento. Por fim, foi analisada a consistência dos dados armazenados, verificando se os eventos capturados foram corretamente persistidos e disponibilizados para consulta.

3.5 Geração de Dados de Teste

Para viabilizar uma avaliação robusta da infraestrutura, foi desenvolvido um método de geração de dados capaz de simular a inserção de grandes volumes de eventos diretamente nos tópicos do Apache Kafka, seguindo rigorosamente os padrões de mensagens pré-estabelecidos pela arquitetura do sistema. Esta abordagem permite a criação de uma massa de dados controlada, essencial para validar o comportamento do ecossistema sob diferentes níveis de estresse. Essa abordagem permite identificar possíveis limitações, gargalos e pontos de melhoria antes de uma eventual aplicação em ambiente real.

3.6 Metodologia de Testes

A validação do sistema foi realizada por meio de duas abordagens complementares:

Testes Manuais: Esta etapa focou na avaliação da usabilidade e no comportamento funcional da interface. Para além da verificação dos fluxos de navegação e da integridade visual, os testes manuais foram fundamentais para auditar a sequência de armazenamento das informações, garantindo que os eventos seguissem o padrão de dados estabelecido na arquitetura. Adicionalmente, realizou-se a validação dos mecanismos de autenticação e autorização, com o objetivo de verificar se o isolamento de dados entre diferentes contas estava operando corretamente, impedindo que informações de um usuário fossem visualizadas ou sobrescritas por outro, reforçando a segurança e a privacidade da solução.

Testes Automatizados: Foram implementados para verificar a estabilidade do sistema e a corretude das funcionalidades sob condições controladas. Para isso, foram desenvolvidos algoritmos *de* automação capazes de simular cenários de maior volume de dados. Em um dos testes realizados, aproximadamente 17.000 eventos de clique foram enviados para a infraestrutura, permitindo avaliar o comportamento do sistema em termos de processamento, armazenamento e estabilidade.

Esses testes também permitiram validar a integração entre as camadas de *front-end* e *back-end*, garantindo que o fluxo de dados — desde o disparo do evento no navegador até o seu processamento e armazenamento — ocorresse sem perdas de informação. Dessa forma, foi possível observar o funcionamento da solução em um cenário de carga simulada, verificando sua consistência e comportamento geral.

3.7 Critérios de Avaliação

A avaliação da solução considerou os seguintes critérios, tais critérios foram elaborados com base em análise do autor a respeito de características que são relevantes para este tipo de serviço:

C1. Capacidade de processamento de eventos em tempo real;
C2. Consistência no armazenamento dos dados;
C3. Comportamento da solução sob carga de dados simulados;
C4. Execução bem-sucedida do processo de integração.

Com base nesses critérios, foi possível analisar a confiabilidade e a aplicabilidade da solução proposta em cenários práticos.

4 DESENVOLVIMENTO DO SISTEMA

O MG-ClickStream-Manager é um sistema desenvolvido para gerenciar eventos de clique de usuários em interfaces web, utilizando uma arquitetura baseada em microsserviços. Este sistema foi projetado para buscar escalabilidade e eficiência, com a capacidade de processar grandes volumes de dados gerados pelas interações dos usuários em tempo real. Para facilitar a integração e a comunicação assíncrona entre os diversos componentes do sistema, foi utilizado o Apache Kafka, uma plataforma de mensageria altamente eficiente que permite o envio e o consumo de eventos em tempo real (KREPS et al., 2011).

Além do MG-ClickStream-Manager, o sistema conta com a API MG-ClickStream-Middleware, que atua como uma camada intermediária entre o *front-end* da aplicação e o sistema de tópicos do Kafka. Esta camada de *middleware* é essencial para garantir a segurança, modularização e escalabilidade da comunicação de eventos (IBM, 2024). O principal objetivo desta abordagem é permitir que as informações de interação do usuário sejam processadas de forma rápida e segura, garantindo que os dados possam ser armazenados e analisados para a melhoria contínua da experiência do usuário.

A necessidade de gerenciar e analisar eventos de clique de forma eficiente é cada vez mais importante para aplicações que dependem de uma análise detalhada do comportamento do usuário. Ao possibilitar a coleta, processamento e análise desses dados, o sistema MG-ClickStream-Manager permite que decisões mais assertivas sejam tomadas para otimizar a experiência do usuário, desde melhorias em interfaces até a personalização de conteúdo.

A arquitetura emprega tecnologias como Kafka e MongoDB que são ferramentas apropriadas para lidar com o crescente volume de dados e as demandas de uma aplicação dinâmica e interativa. A utilização de *JWT* (*JSON Web Token*) para autenticação oferece uma camada adicional de segurança, assegurando a integridade e a validade das requisições (IBM, 2024; RICHARDSON, 2018). Além disso, a implementação de containers via Docker e Rancher Desktop facilita o desenvolvimento, testes e *deployment* do sistema, promovendo a

portabilidade e a padronização do ambiente em qualquer estágio do ciclo de vida do software (RANCHER DESKTOP, 2024).

4.1 Resumo da Solução proposta

A solução desenvolvida é composta por três principais componentes:

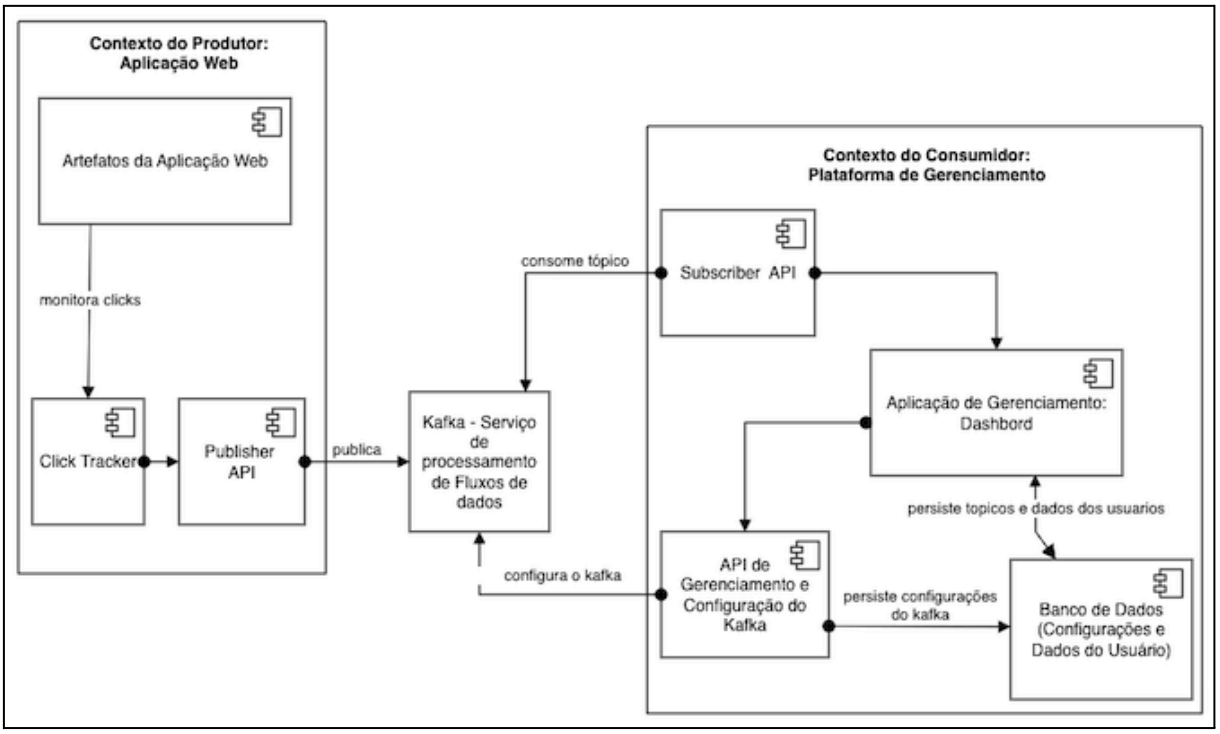
- **Infraestrutura Automatizada para Gerenciamento de Tópicos Kafka:** O sistema permite a criação, atualização, exclusão e monitoramento de eventos de clique de maneira automatizada, eliminando a necessidade de configuração manual complexa. O uso do Apache Kafka garante alta escalabilidade e eficiência no processamento de eventos, possibilitando a ingestão e transmissão de grandes volumes de dados em tempo real.
- **Template para Integração com Aplicações *Front-end*:** Para facilitar a implementação do *clickstream*, o projeto inclui um modelo pré-definido com regras e diretrizes para capturar e enviar eventos de clique diretamente de aplicações web ou móveis. Esse template reduz o tempo necessário para a adoção da solução e padroniza a coleta de dados, garantindo uma estrutura organizada e de fácil uso.
- **Documentação Padronizada e Detalhada:** Um dos desafios ao implementar tecnologias avançadas é a falta de guias claros para sua adoção. Para acabar com esse problema, o sistema conta com uma documentação completa disponível no Github (MIGUELGABRIEL01, 2026d), explicando passo a passo como configurar a infraestrutura, padronizar os eventos, consumir os dados coletados e integrar a ferramenta a diferentes tipos de aplicações.

O sistema MG-ClickStream-Manager, juntamente com a API MG-ClickStream-Middleware, oferece uma solução segura, escalável e eficiente para a gestão de eventos de clique, com o objetivo de melhorar a análise de comportamento dos usuários e a personalização da experiência em aplicações web.

4.2 Arquitetura do Sistema

A arquitetura dos sistemas MG-ClickStream-Manager, MG-ClickStream-Middleware e MG-ClickStream-Interface, representada na Figura 2, foi projetada para seguir o padrão de microsserviços, utilizando Apache Kafka como o serviço de processamento de fluxos de dados e MongoDB como banco de dados para o armazenamento dos dados gerados pelos eventos de clique. Essa abordagem oferece várias vantagens, incluindo maior escalabilidade, desacoplamento entre os componentes e flexibilidade para o crescimento e adaptação do sistema conforme novas necessidades de negócios surgem.

Figura 2 - Arquitetura do projeto



Fonte: Elaborado pelo autor, 2025.

A seguir, descrevemos brevemente os principais módulos do projeto:

- **Aplicações Web:** Uma aplicação web é um sistema computacional cujas interfaces são acessadas via navegador e executadas em servidores *HTTP*. No contexto desta arquitetura, elas representam a origem dos dados, possuindo as configurações necessárias para capturar eventos de interação do usuário e enviá-los de forma transparente para o ecossistema de

monitoramento, usando de componentes e classes que processam o dado recolhido da interface e prepara para o envio ao middleware.

- **API de Mensageria Pub/Sub:** Estes componentes são responsáveis por operacionalizar o modelo de publicação e assinatura (*Publisher/Subscriber)*. Sua função é receber os dados brutos enviados pelas aplicações web, validar a autenticidade da requisição e realizar a ingestão desses eventos nos tópicos específicos do Apache Kafka, garantindo o desacoplamento entre a origem do clique e o processamento final.
- **Apache Kafka:** Atua como o motor de mensageria distribuída da plataforma. Sua função principal é servir como um *buffer* de alta disponibilidade e baixa latência de dados, organizando os fluxos de dados recebidos em tópicos. O Kafka garante que grandes volumes de eventos sejam retidos e disponibilizados para consumo de maneira ordenada e resiliente a falhas.
- **Serviço de Processamento de Stream:** Constitui o núcleo administrativo da solução. Este serviço gerencia o ciclo de vida do cliente na plataforma, controlando desde o cadastro de usuários e autenticação até a orquestração técnica dos tópicos no Kafka. É responsável por garantir que cada desenvolvedor possua um ambiente isolado para o recebimento de seus dados.
- **App de Gerenciamento de Tópicos:** Representa a camada de visualização e controle oferecida ao usuário final. Trata-se de uma interface gráfica que permite listar os tópicos associados a uma conta, monitorar a volumetria de dados gerados e visualizar métricas através de dashboards intuitivos, além de oferecer a funcionalidade de exportação dos registros em formato CSV para análises externas.
- **Banco de Dados:** Utilizado para o armazenamento persistente de metadados do sistema e registros de eventos processados. Como um banco de dados NoSQL orientado a documentos, sua importância reside na flexibilidade para armazenar as estruturas variáveis das mensagens *JSON* de cliques, permitindo consultas performáticas e escalabilidade horizontal para suportar o crescimento da massa de dados.

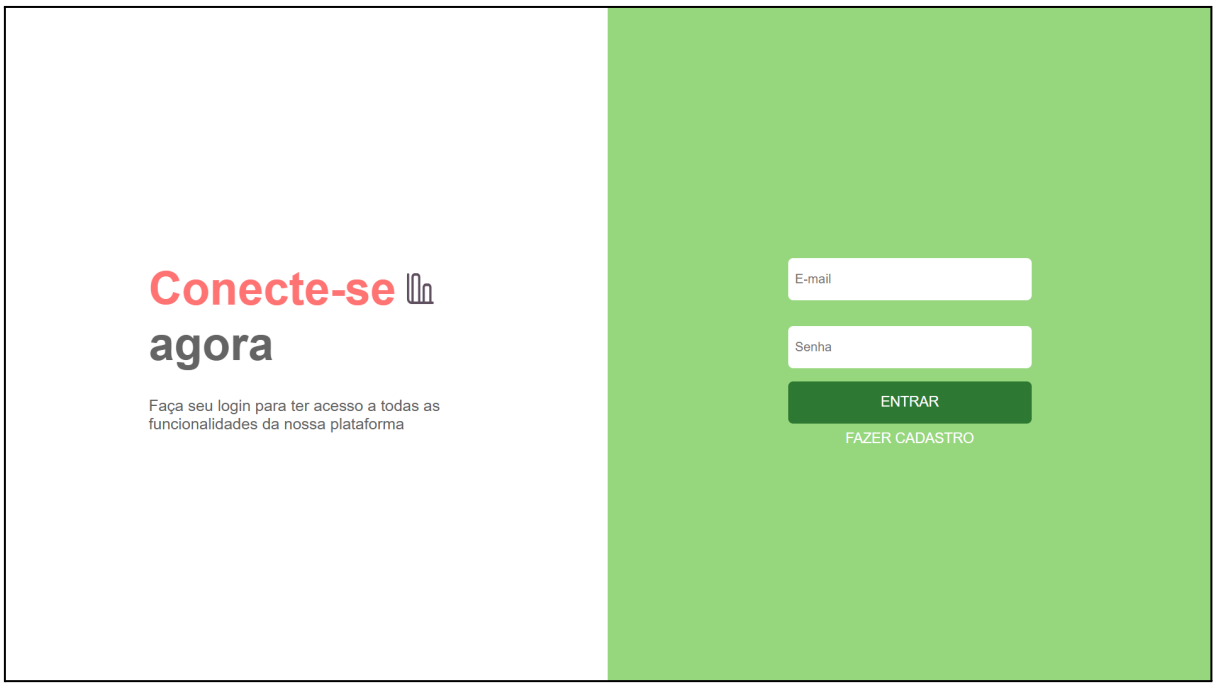
4.3 Fluxo de Funcionalidades

Esta seção detalha o caminho dos dados de interação, desde a origem até o processamento. O Fluxo de Funcionalidades demonstra como os principais componentes da solução – o Template de Integração *front-end* e a Infraestrutura Automatizada Kafka – trabalham em conjunto. O objetivo é mapear a jornada do evento de clique: sua captura padronizada na aplicação, a ingestão na camada de mensageria em tempo real e a consequente disponibilização para a análise, transformando dados brutos em insights acionáveis.

4.3.1 Autenticação de Usuário

A tela inicial da aplicação MG-ClickStream-Manager é o ponto de entrada para novos e antigos usuários. Nela, o usuário tem a possibilidade de realizar o *login* utilizando seu e-mail e senha cadastrados previamente, garantindo o acesso seguro às funcionalidades da plataforma, como apresentado na Figura 3.

Figura 3 - Tela de login da aplicação *front-end*



Fonte: Elaborado pelo autor, 2025.

4.3.2 Cadastro de Novo Usuário

Na página de cadastro, o novo usuário é convidado a preencher um formulário com informações essenciais para criação de sua conta. Os campos obrigatórios incluem o nome completo, um endereço de e-mail válido e uma senha de acesso, como apresentado na Figura 4.

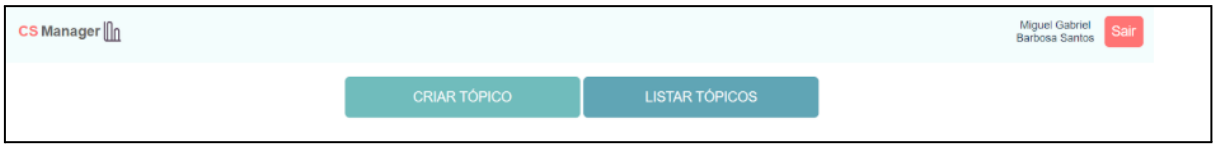
Figura 4 - Tela de cadastro da aplicação *front-end*

Fonte: Elaborado pelo autor, 2025.

4.3.3 Página Principal e Criação de Tópicos Kafka

Assim que a autenticação é realizada com sucesso, o usuário é automaticamente redirecionado para a página principal da aplicação, que serve como o centro de controle das principais funcionalidades oferecidas pela plataforma MG-ClickStream-Manager, como apresentada na Figura 5. Nessa interface, o usuário encontra um *layout* projetado para facilitar a navegação e o acesso às funcionalidades disponíveis. No topo da tela, dois botões se destacam, representando as ações para criar e listar tópicos. A página principal também serve como ponto de partida para outras seções, reforçando a proposta de uma experiência de usuário fluida, objetiva e centrada no usuário:

Figura 5 - Botões de seleção de opções iniciais após autenticação



Fonte: Elaborado pelo autor, 2025.

Caso este seja o primeiro acesso do usuário à plataforma, ele será apresentado à opção de criar um novo tópico, etapa essencial para iniciar a utilização dos recursos oferecidos pelo MG-ClickStream-Manager. Essa funcionalidade permite que o usuário configure um ambiente personalizado para o monitoramento e gerenciamento dos dados de cliques provenientes da aplicação web. A criação de um tópico envolve o preenchimento de um formulário, exibido na Figura 6, com informações básicas, como o nome desejado, que será utilizado como identificador no sistema. Esse processo é rápido, intuitivo e foi projetado com foco em usabilidade inicial ao usuário, incentivando o engajamento e a exploração das demais ferramentas da plataforma. Após a criação do tópico, o sistema já estará pronto para começar a receber e exibir os dados capturados em tempo real.

Figura 6 - Formulário de criação de tópico

Fonte: Elaborado pelo autor, 2025.

4.3.4 Gerenciamento de Tópicos Kafka

O usuário autenticado pode criar novos tópicos Kafka para armazenar eventos de clique. Cada tópico será associado ao ID do usuário e servirá como canal para a comunicação com os serviços de *back-end*. Além disso, os tópicos podem ser listados, removidos ou exportados em formato CSV conforme a necessidade do usuário.

4.3.5 Recebimento e Armazenamento de Eventos de Clique

Os eventos de clique gerados pelos usuários são capturados e enviados para o Kafka, onde são processados em tempo real. Após o processamento, os eventos são armazenados no MongoDB, garantindo que as informações estejam disponíveis para análise futura.

Para a integração com o serviço, a aplicação cliente deve utilizar o paradigma da programação reativa no *front-end*. Isso permite que a interface monitore as interações do usuário em tempo real e despache os eventos de forma assíncrona, sem bloquear a experiência de navegação.

No Vue.js, essa reatividade é implementada instanciando a classe de monitoramento e vinculando-a ao ciclo de vida do componente. O desenvolvedor deve incluir o identificador padronizado diretamente nos elementos *HTML* (DOM) para que o serviço capture a origem exata do clique. O Código Fonte 1 exemplifica a criação de um componente Vue.js com três botões, cada um com um ID seguindo a convenção de nomenclatura, e os configura para enviar dados de cliques para a API.

Código Fonte 1 - Exemplo de código para rastreamento de clicks no *front-end*.

```
<template>
  <div>
    <h1>Exemplo de Rastreamento de Cliques</h1>

    <button :id="'CS-simple-action-Button1'" @click="handleClick('CS-simple-action-Button1')">Botão
1</button>
    <button :id="'CS-simple-action-Button2'" @click="handleClick('CS-simple-action-Button2')">Botão
2</button>
    <button :id="'CS-simple-action-Button3'" @click="handleClick('CS-simple-action-Button3')">Botão
3</button>
  </div>
```

```
</template>

<script>
import ClickTracker from '../ClickTracker';

export default {
  name: 'ClickTrackerComponent',
  data() {
    return {
      clickTracker: new ClickTracker('http://localhost:3030/save') // URL da API NestJS
    };
  },
  methods: {
    handleClick(buttonId) {
      this.clickTracker.trackButtonClick(buttonId);
    }
  }
}
</script>

<style scoped>
button {
  background: rgb(7, 141, 47);
  color: aliceblue;
  font-size: 15px;
  width: 180px;
  height: 40px;
  text-decoration: none;
  border: none;
  border-radius: 2px;
  margin: 10px;
}
</style>
```

Fonte: Elaborado pelo autor, 2025.

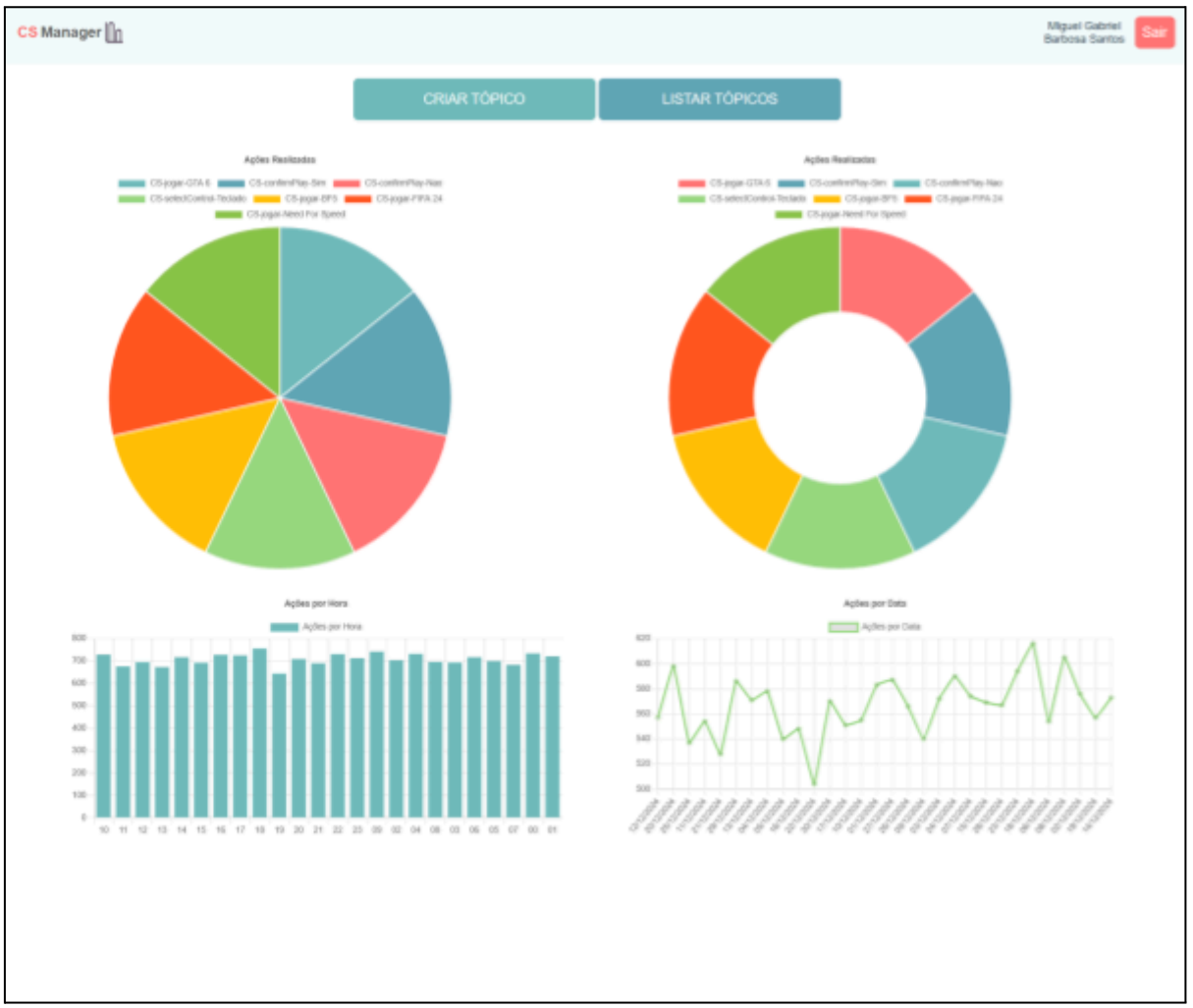
4.3.6 Exibição de Estatísticas de Cliques

A *API* oferece endpoints para consultar e visualizar as estatísticas dos eventos de clique armazenados no banco de dados. Essas informações são utilizadas para analisar o comportamento dos usuários e fornecer *insights* valiosos para melhorar a experiência do usuário. A *interface* permite listar esses dados de forma organizada em dashboards, bem como deletar ou baixar os registros em formato CSV, como apresentado nas figuras 7 e 8.

Figura 7 - Modal de listagem de tópicos criados com opções de leitura, download e exclusão de dados.

Fonte: Elaborado pelo autor, 2025.

Figura 8 - Dashboard com os dados coletados



Fonte: Elaborado pelo autor, 2025.

4.4 Endpoints e Funcionalidades da API

A API fornece os seguintes endpoints principais para interagir com o sistema:

- Cadastro de Usuário: POST /users/create – Registra um novo usuário na plataforma.

- Autenticação de Usuário: POST /users/auth – Realiza o login de um usuário e retorna um token JWT.

- Criar Tópico Kafka: POST /users/topics – Cria um novo tópico Kafka associado ao usuário autenticado.

- Listar Tópicos do Usuário: GET /users/listTopics – Retorna todos os tópicos Kafka criados pelo usuário autenticado.

- Deletar Tópico Kafka: DELETE /users/deleteTopic/:topicId – Exclui um tópico Kafka específico associado ao usuário autenticado.

- Salvar Evento de Clique: POST /save – Recebe e envia os eventos de clique do *front-end* para o Kafka.

4.5 Segurança e Autenticação

A segurança do sistema é garantida através de várias camadas de proteção:

- ***JWT***: A autenticação baseada em *JSON Web Tokens* (JWT) garante que apenas usuários autenticados possam acessar os endpoints protegidos da API.

- **Validação de Payloads**: Todas as requisições enviadas à *API* passam por um processo de validação rigorosa, o que ajuda a prevenir ataques maliciosos, como injeção de código ou valores inválidos.

- **Controle de Acesso**: A aplicação implementa um controle de acesso robusto, garantindo que apenas usuários com permissões adequadas possam executar ações administrativas, como a criação e exclusão de tópicos Kafka.

5 RESULTADOS DA AVALIAÇÃO

Nesta seção, apresenta-se em detalhes o estudo de caso conduzido para avaliar se a infraestrutura proposta é capaz de fornecer o serviço de *clickstream* conforme os objetivos deste trabalho. Por meio de uma simulação controlada, foi possível mensurar o tempo de resposta do processamento assíncrono e identificar o limite máximo de registros suportados por tópico (17 mil registros) sem detectar degradação de performance.

Além da validação técnica do *back-end*, os dados gerados foram fundamentais para testar a integridade e a atualização em tempo real dos *dashboards*, garantindo que as visualizações estatísticas refletissem fielmente os eventos processados pelo MG-ClickStream-Manager.

Cabe ressaltar que as avaliações de desempenho foram realizadas em ambiente de laboratório com os serviços executados localmente. Embora os resultados validem a funcionalidade da *pipeline*, essa configuração não reflete perfeitamente um cenário de produção distribuído em nuvem, onde variáveis como latência de comunicação e overhead de rede podem impactar a performance global do sistema.

5.1 Estudo de Caso

Conforme proposto na metodologia, para validar a viabilidade da infraestrutura, foi desenvolvido um estudo de caso focado na simulação de um cenário real de navegação e coleta de eventos. Para isso, foi construída do zero uma aplicação web específica utilizando o *framework* Vue.js, que simula uma interface de jogos online (MIGUELGABRIEL01, 2026e). O objetivo principal foi testar a capacidade de um desenvolvedor integrar o MG-ClickStream, seguindo o tutorial disponibilizado (Apêndice A), para implementar um monitoramento de cliques completo sem a necessidade de gerenciar os detalhes da infraestrutura de mensageria e processamento.

Além de validar a facilidade de integração, o experimento testou a capacidade de processamento e armazenamento de dados em situações de carga elevada. O

ambiente de testes foi composto por um notebook Dell, equipado com um processador Intel Core i7 de 11ª geração, 16 GB de memória *RAM*, 500 GB de *SSD* e sistema operacional Windows 11 Pro.

5.2 Procedimento de Integração

Todo o procedimento de integração seguiu o passo a passo descrito no tutorial disponibilizado no repositório oficial da plataforma no GitHub (MIGUELGABRIEL01, 2026d) e no Apêndice A, o que possibilita que qualquer desenvolvedor reproduza o cenário com facilidade. Para simular a navegação de um usuário real, foi criada uma pequena aplicação web utilizando Vue.js, representando uma página de jogos na nuvem. Nessa página, o usuário podia escolher um jogo e definir se gostaria de jogar utilizando o teclado ou um joystick. Através deste estudo de caso, simulou-se o uso do MG-ClickStream para construir um sistema de monitoramento de cliques para a referida plataforma de jogos; apenas com a realização de um cadastro e seguindo o tutorial de configuração , foi possível operacionalizar o sistema, obtendo os dados e utilizando as funcionalidades de monitoramento diretamente no *dashboard* do MG-ClickStream.

5.3 Captura de Eventos

Em conformidade com as boas práticas descritas no tutorial, cada botão que deveria registrar um clique foi identificado com um ID específico, padronizado para garantir a correta rastreabilidade das ações. Em seguida, foi implementada a classe responsável por capturar os eventos de clique do usuário. Essa classe, também descrita no tutorial, foi integrada ao site e configurada para armazenar os eventos localmente e enviá-los periodicamente à *API* intermediária da plataforma, o MG-ClickStream-Middleware. Essa *API*, por sua vez, foi encarregada de direcionar os dados ao tópico Kafka previamente criado pelo próprio usuário dentro da interface da plataforma.

5.4 Validação dos Registros

Após o envio dos dados, a consulta e validação dos registros foram feitas por meio da interface **MG-ClickStream-Manager-Interface**, onde foi possível acompanhar o recebimento e o processamento dos eventos. Além do cenário de navegação, foi realizado um segundo experimento com foco em testes de performance. Esse teste teve como finalidade avaliar a capacidade de carga da infraestrutura, especialmente no que diz respeito à quantidade de registros que um tópico é capaz de suportar.

5.5 Teste de Performance

Para isso, foi criada uma função em JavaScript que gerava *payloads* randômicos, simulando interações de clique. A função inseriu esses dados diretamente no tópico selecionado, permitindo medir a robustez e escalabilidade do sistema. Os testes demonstraram que cada tópico é capaz de armazenar aproximadamente 17 mil registros sem perda ou falha de transmissão.

5.6 Resultados Finais

Após a execução dos protocolos de teste e das funções de carga, os dados foram verificados por meio da MG-ClickStream-Manager-Interface, permitindo validar a solução com base nos critérios estabelecidos na metodologia (Seção 3.7). Os resultados obtidos comprovaram o sucesso da aplicação nos seguintes pontos:

- **C1 - Capacidade de processamento de eventos em tempo real:** A comunicação entre o *middleware*, o Apache Kafka e o serviço de processamento demonstrou alta fluidez, garantindo que o fluxo de eventos — desde o disparo no navegador até a disponibilização no *dashboard* — ocorresse sem perdas de informação.
- **C2 - Consistência no armazenamento dos dados:** Os testes manuais e automatizados ratificaram a integridade do armazenamento no MongoDB, mantendo a estrutura dos dados (*JSON*) conforme o padrão definido, mesmo sob condições de execução contínua.
- **C3 - Comportamento da solução sob carga de dados simulados:** No teste de carga realizado, foram enviados aproximadamente 17.000 eventos em 3 minutos de clique para a infraestrutura. Durante esse processo, o sistema

manteve comportamento estável, sem perda de eventos, indicando capacidade adequada de processamento no ambiente experimental.

- **C4 - Execução bem-sucedida do processo de integração:** O estudo de caso realizado com a aplicação de jogos desenvolvida em Vue.js demonstrou que o processo de integração foi realizado com sucesso a partir do tutorial fornecido. Verificou-se, na prática, que a aplicação pôde ser integrada ao serviço de processamento de *clickstream* sem necessidade de ajustes estruturais complexos, utilizando apenas os passos descritos no tutorial fornecido. Esse resultado evidencia a viabilidade da solução para integração com aplicações web, especialmente com o suporte do guia prático e dos recursos disponíveis no repositório oficial.

Em suma, os estudos avaliativos indicaram que a plataforma atendendo os critérios de avaliação pré-estabelecidos na metodologia.

6 CONSIDERAÇÕES FINAIS

Após a implementação de todo o fluxo proposto, que envolve desde a coleta dos dados, seu armazenamento em tópicos do Apache Kafka, até a possibilidade de leitura e visualização desses dados, constata-se que o objetivo inicial foi atingido. A utilização da plataforma e dos serviços disponibilizados aos desenvolvedores permitiu construir um ambiente capaz de capturar, armazenar, tratar e entregar dados referentes às interações dos usuários em um sistema web. Dessa forma, foi possível identificar os cliques realizados pelos usuários do site, possibilitando ao usuário da plataforma proposta a análise dos comportamentos dos usuários dentro do sistema disponibilizado.

Ao final da execução de todo o fluxo, concluiu-se que a plataforma é capaz, dentro dos parâmetros propostos, de realizar todas as etapas de um sistema baseado em *clickstream*. Para validação prática da plataforma, foi desenvolvido um caso de uso utilizando uma página que simulava um ambiente de jogos online (MIGUELGABRIEL01, 2026e). A partir da documentação disponibilizada foi possível realizar a integração desta página web com o serviço de *clickstream* com a inclusão de poucas linhas de código, permitindo que todas as interações dos usuários fossem devidamente registradas e armazenadas no Apache Kafka. Consequentemente, o administrador da aplicação conseguiu acessar e visualizar os dados coletados, concluindo assim todo o ciclo proposto pela plataforma. Considerando os resultados da avaliação do MG-ClickStream, demonstrou-se com sucesso o cumprimento do objetivo proposto: “permitir que desenvolvedores não especialistas em processamento de streams implementem o monitoramento de eventos em qualquer sistema web”. A solução empregou bibliotecas de programação reativa e frameworks modernos, como Vue.js, React, Svelte ou Angular, simplificando a integração com serviço.

Diante dos resultados obtidos, verifica-se que a solução atende ao escopo definido no início do projeto, em propor o desenvolvimento de uma plataforma de serviços de *clickstream*, com o objetivo de tomar essa tecnologia mais acessível, mostrando-se funcional e de fácil integração com aplicações web, sem a necessidade de montar ambiente complexo do zero. O caso de uso, portanto,

validou a proposta da plataforma, demonstrando sua aplicabilidade prática e sua capacidade de fornecer dados para a análise do comportamento dos usuários em ambientes digitais.

Para fins de transparência e fomento à comunidade, toda a estrutura da solução de rastreamento de cliques foi disponibilizada como código aberto no GitHub (MIGUELGABRIEL01, 2026a; MIGUELGABRIEL01, 2026b; MIGUELGABRIEL01, 2026c; MIGUELGABRIEL01, 2026d).

6.1 Ameaças à Validade

Embora os resultados obtidos tenham sido satisfatórios, algumas limitações representam ameaças à validade do trabalho. A principal delas refere-se ao fato de que os testes de performance foram executados em ambiente de laboratório, com todos os serviços rodando localmente em uma única máquina. Esta configuração não é o cenário ideal para uma avaliação definitiva de desempenho, uma vez que, em ambientes de produção reais, os serviços costumam estar distribuídos em nuvem. Essa distribuição introduz variáveis críticas que não puderam ser plenamente mensuradas neste estudo, como a latência de comunicação entre servidores e o overhead de rede. Adicionalmente, a plataforma não foi submetida a uma carga de dados reais provenientes de sistemas em produção no mercado.

O caso de uso desenvolvido, embora tenha cumprido sua função de validar tecnicamente o funcionamento da plataforma, foi elaborado e executado pelo próprio autor do projeto. Dessa forma, não houve uma validação externa, feita por terceiros ou em cenários reais de produção, o que poderia oferecer uma análise mais robusta sobre o desempenho, a escalabilidade e a resiliência do sistema.

Além disso, testes de estresse, simulações de grande volume de dados, possíveis falhas no ambiente e outros desafios comuns em ambientes corporativos não foram contemplados de forma abrangente. Isso representa uma limitação, visto que não é possível garantir, com total segurança, que o sistema mantenha o mesmo desempenho e estabilidade em cenários mais complexos ou sob altas demandas.

Portanto, reconhece-se que futuras validações, realizadas em ambientes de produção, com usuários reais e diferentes cenários de uso, são fundamentais para confirmar a robustez, a eficácia e a escalabilidade da plataforma proposta.

6.2 Trabalhos Futuros

Diante dos resultados alcançados e das limitações identificadas, é possível apontar algumas direções para trabalhos futuros que visam aprimorar e expandir a plataforma desenvolvida. Uma das principais propostas consiste na realização de experimentos em ambientes mais próximos da realidade, envolvendo um grupo de desenvolvedores externos. Essa abordagem permitirá avaliar, de forma mais precisa, tanto o funcionamento técnico da plataforma quanto a experiência de uso por parte de profissionais que não participaram diretamente do seu desenvolvimento.

Além disso, pretende-se também realizar experimentos de uso da plataforma envolvendo um grupo de desenvolvedores, com o objetivo de avaliar com maior precisão o funcionamento e a experiência de uso da ferramenta, identificando possíveis pontos de melhoria a partir do feedback direto desses usuários.

Testes de carga, escalabilidade e resiliência poderiam ser implementados, simulando cenários com grandes volumes de dados, múltiplas conexões simultâneas e situações adversas, a fim de verificar o comportamento do sistema em condições extremas. Isso contribuiria para validar a robustez da solução em ambientes de produção.

Outra possibilidade de melhoria está na expansão das funcionalidades da plataforma, permitindo, por exemplo, a criação de *dashboards* interativos, análises em tempo real e geração de relatórios personalizados, que possam oferecer insights mais completos aos administradores das aplicações web. Da mesma forma, a integração com outras tecnologias e ferramentas de análise de dados, como sistemas de inteligência artificial e machine learning, poderia agregar ainda mais valor à solução, possibilitando análises preditivas e automação de processos.

Por fim, melhorias na documentação, na interface de integração e na usabilidade geral da plataforma também são pontos relevantes, uma vez que facilitariam sua adoção por parte de outros desenvolvedores e empresas interessadas em implementar soluções baseadas em *clickstream*.

REFERÊNCIAS

AMAZON WEB SERVICES. Amazon Kinesis Documentation. 2026a. Disponível em: https://docs.aws.amazon.com/kinesis/. Acesso em: 19 abr. 2026.

AMAZON WEB SERVICES. AWS Lambda Documentation. 2026b. Disponível em: https://docs.aws.amazon.com/lambda/. Acesso em: 19 abr. 2026.

APACHE FLINK. Apache Flink Documentation. 2026. Disponível em: https://nightlies.apache.org/flink/flink-docs-stable/. Acesso em: 19 abr. 2026.

APACHE KAFKA. Apache Kafka Documentation. 2025. Disponível em: https://kafka.apache.org/documentation/. Acesso em: 7 out. 2025.

APACHE SPARK. Apache Spark: Unified Engine for multi-hop data analytics. 2026. Disponível em: https://spark.apache.org/docs/latest/. Acesso em: 19 abr. 2026.

BUCKLIN, Randolph E.; SISMEIRO, Catarina. Clickstream Data: Modeling Consumer Behavior on the Internet. Marketing Science, v. 28, n. 2, p. 249-281, 2009.

FOWLER, Martin. Microservices. 2014. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 7 out. 2025.

FOWLER, Martin. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2012.

GARTNER, Inc. Market Guide for Digital Analytics and Customer Data Platforms. Gartner Research, 2022. Disponível em: https://www.gartner.com/en/documents/market-guide-for-digital-analytics-and-customer-data-platforms. Acesso em: 7 out. 2025.

GOOGLE CLOUD. Cloud Dataflow Documentation. 2026a. Disponível em: https://cloud.google.com/dataflow/docs. Acesso em: 19 abr. 2026.

GOOGLE CLOUD. Cloud Pub/Sub Documentation. 2026b. Disponível em: https://cloud.google.com/pubsub/docs. Acesso em: 19 abr. 2026.

GOOGLE CLOUD. Controle de acesso para APIs Cloud Billing. 2024. Disponível em: https://cloud.google.com/billing/docs/access-control?hl=pt-br. Acesso em: 7 out. 2025.

HARDT, Dick. JSON Web Token (JWT). IETF, RFC 7519, May 2014. Disponível em: https://datatracker.ietf.org/doc/html/rfc7519. Acesso em: 7 out. 2025.

HAZELCAST. Stream Processing. Foundations of Event-Driven Architecture, 2024. Disponível em: https://hazelcast.com/foundations/event-driven-architecture/stream-processing. Acesso em: 15 mai. 2024.

IBM. O que é segurança de API? 2024. Disponível em: https://www.ibm.com/br-pt/topics/api-security. Acesso em: 7 out. 2025.

KAMIŃSKI, Krzysztof; SZYMANOWSKI, Mateusz. NestJS: The Progressive Node.js Framework. 2024. Disponível em: https://nestjs.com/. Acesso em: 7 out. 2025.

KLEPPMANN, Martin. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017.

KREPS, Jay. The log: what every software engineer should know about real-time data's unifying abstraction. LinkedIn Engineering Blog, 16 ago. 2011. Disponível em: https://engineering.linkedin.com/distributed-systems/log. Acesso em: 7 out. 2025.

KREPS, Jay; NARKHEDE, Neha; RAO, Jun. Kafka: a Distributed Messaging System for Log Processing. In: Proceedings of the NetDB, 2011. Disponível em: https://www.cs.cmu.edu/~qifengw/pdf/kafka.pdf. Acesso em: 7 out. 2025.

LOSHIN, David. Big Data Analytics: From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph. Waltham: Morgan Kaufmann, 2013.

MARZ, Nathan; WARREN, James. Big Data: Principles and best practices of scalable realtime data systems. Shelter Island: Manning Publications, 2015.

MICROSOFT. Azure Stream Analytics Documentation. 2026a. Disponível em: https://learn.microsoft.com/en-us/azure/stream-analytics/. Acesso em: 19 abr. 2026.

MICROSOFT. TypeScript: JavaScript with Syntax for Types. 2024. Disponível em: https://www.typescriptlang.org/. Acesso em: 7 out. 2025.

MIGUELGABRIEL01. MG-ClickStream-Manager-API. GitHub, 2026a. Disponível em: https://github.com/miguelgabriel01/MG-ClickStream-Manager-API. Acesso em: 19 abr. 2026.

MIGUELGABRIEL01. MG-ClickStream-Manager-Interface. GitHub, 2026b. Disponível em: https://github.com/miguelgabriel01/MG-ClickStream-Manager-Interface. Acesso em: 19 abr. 2026.

MIGUELGABRIEL01. MG-ClickStream-Middleware. GitHub, 2026c. Disponível em: https://github.com/miguelgabriel01/MG-ClickStream-Middleware. Acesso em: 19 abr. 2026.

MIGUELGABRIEL01. MG-ClickStream-tutorial. GitHub, 2026d. Disponível em: https://github.com/miguelgabriel01/DocUse-managerKlick. Acesso em: 19 abr. 2026.

MIGUELGABRIEL01. Teste-Inserir-Dados-ClickStream. GitHub, 2026e. Disponível em: https://github.com/miguelgabriel01/testeInserirDadosClickStream. Acesso em: 22 abr. 2026.

MOE, Wendy W.; FADER, Peter S. Dynamic conversion behavior at e-commerce sites: Herding the tigers and taming the sharks. Management Science, v. 50, n. 3, p. 326-346, 2004. Disponível em: https://doi.org/10.1287/mnsc.1040.0153. Acesso em: 19 abr. 2026.

MONGODB. Build Event-Driven Applications with MongoDB. 2024. Disponível em: https://www.mongodb.com/resources/solutions/use-cases/building-event-driven-applications-with-mongodb. Acesso em: 7 out. 2025.

NEWMAN, Sam. Building Microservices: Designing Fine-Grained Systems. 2. ed. Sebastopol: O'Reilly Media, 2021.

RANCHER DESKTOP. Why use Rancher Desktop? 2024. Disponível em: https://www.rancher.com/products/rancher-desktop. Acesso em: 7 out. 2025.

RED HAT. O que são microsserviços? 2023. Disponível em: https://www.redhat.com/pt-br/topics/microservices/what-are-microservices. Acesso em: 7 out. 2025.

RICHARDSON, Chris. Microservices Patterns: With examples in Java. Shelter Island: Manning Publications, 2018.

SADALAGE, Pramod J.; FOWLER, Martin. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Upper Saddle River: Addison-Wesley, 2012.

TYPICODE. JSON-Server. 2024. Disponível em: https://github.com/typicode/json-server. Acesso em: 7 out. 2025.

APÊNDICE A

Este guia descreve o passo a passo para integrar o rastreamento de cliques em sua aplicação cliente e conectá-la à infraestrutura do MG-ClickStream.

Passo 1: Preparação do Ambiente (*middleware*)

Antes de configurar o cliente, a *API* de ingestão deve estar ativa e acessível.

1. Realize o clone do repositório MG-ClickStream-Middleware, disponível em: https://github.com/miguelgabriel01/MG-ClickStream-Middleware.
2. No terminal, instale as dependências com o comando npm install.
3. Inicie o servidor com o comando npm run start. Por padrão, a *API* estará disponível na porta 3030.

Passo 2: Configuração do Tópico na Interface

Para que os dados sejam organizados corretamente, você precisa definir o destino das mensagens:

1. Acesse a plataforma de gerenciamento MG-ClickStream-Manager.
2. Crie um novo tópico de coleta (exemplo: fluxo-loja-virtual).
3. Configuração no Código: No arquivo de configuração da sua aplicação cliente(MG-ClickStream-Middleware), insira o nome exato do tópico criado. É este nome que garantirá que seus dados sejam direcionados para o seu *dashboard* exclusivo.

Passo 3: Instalação de Dependências no Cliente

Sua aplicação precisará de uma biblioteca para enviar os dados via protocolo HTTP.

1. No diretório do seu projeto web (Vue.js ou similar), instale o Axios:
    - Com npm: npm install axios
    - Com yarn: yarn add axios

Passo 4: Implementação da Classe ClickTracker (Copy & Paste)

Para facilitar a integração, o código da classe ClickTracker já está totalmente disponível na documentação técnica.

1. Acesse o diretório src/ no repositório oficial (https://github.com/miguelgabriel01/MG-ClickStream-Middleware).
2. Copie e cole o conteúdo da classe(código fonte 2) no seu projeto cliente. Ela já vem configurada para capturar o ID do botão, gerenciar o "caminho percorrido" e realizar o envio automático para o *middleware* após o intervalo programado.

Código Fonte 2 - Exemplo de código para rastreamento de clicks no front-end.

```
<template>
  <div>
    <h1>Exemplo de Rastreamento de Cliques</h1>

    <button :id="'CS-simple-action-Button1'" @click="handleClick('CS-simple-action-Button1')">Botão
1</button>
    <button :id="'CS-simple-action-Button2'" @click="handleClick('CS-simple-action-Button2')">Botão
2</button>
    <button :id="'CS-simple-action-Button3'" @click="handleClick('CS-simple-action-Button3')">Botão
3</button>
  </div>
</template>

<script>
import ClickTracker from '../ClickTracker';

export default {
  name: 'ClickTrackerComponent',
  data() {
    return {
      clickTracker: new ClickTracker('http://localhost:3030/save') // URL da API NestJS
    };
  },
  methods: {
    handleClick(buttonId) {
      this.clickTracker.trackButtonClick(buttonId);
    }
  }
}
</script>

<style scoped>
button {
  background: rgb(7, 141, 47);
  color: aliceblue;
  font-size: 15px;
  width: 180px;
  height: 40px;
  text-decoration: none;
  border: none;
  border-radius: 2px;
  margin: 10px;
}
</style>
```

Passo 5: Padronização de Nomenclatura (IDs)

Para que os relatórios e *dashboards* reflitam a realidade, os elementos de interface (botões e links) devem seguir a regra:

- Estrutura: CS-[Seção]-[Ação]
- Exemplos práticos:
    - CS-nav-home: Botão de início na barra de navegação.
    - CS-form-enviar: Botão de submissão de formulário.
    - CS-produto-comprar: Botão de compra em uma vitrine.

Passo 6: Verificação da Persistência e Fluxo de Dados

Após realizar os cliques na interface, valide se o fluxo foi concluído com sucesso:

1. **Processamento Assíncrono:** Os dados passam pela fila do Kafka antes do armazenamento final.
2. **Validação Final:** A confirmação do fluxo completo — desde o clique até a persistência no MongoDB — pode ser verificada através dos logs de consumo do serviço MG-ClickStream-Manager. Uma vez logados, os dados aparecerão automaticamente nos seus *dashboards* para análise.