LUMEN ET VIRTUS
ISSN: 2177-2789

BOAS PRÁTICAS EM DESENVOLVIMENTO DE APIS PARA INTEGRAÇÃO DE SISTEMAS

https://doi.org/10.56238/levv16n45-069

Data de submissão: 25/01/2025 **Data de publicação:** 25/02/2025

Rodrigo Monteiro Guedes de Almeida

RESUMO

Este artigo tem como objetivo analisar os impactos das boas práticas aplicadas no desenvolvimento de APIs voltadas à integração de sistemas corporativos. A pesquisa foi conduzida por meio de uma revisão bibliográfica com abordagem qualitativa. O objetivo foi identificar estratégias técnicas, modelos arquiteturais e soluções operacionais adotadas em contextos diversos, visando compreender como as interfaces de programação podem promover interoperabilidade, escalabilidade e segurança entre plataformas distintas. A análise evidenciou a prevalência da arquitetura REST como padrão dominante na construção de APIs modernas, destacando seus benefícios quanto à leveza na comunicação, clareza estrutural e compatibilidade com os protocolos da web. Também foram observadas práticas recomendadas como versionamento explícito, autenticação baseada em tokens, testes automatizados e uso de ferramentas de monitoramento em tempo real. A documentação das interfaces foi identificada como fator determinante para facilitar a integração com consumidores externos e internos, reforçando a importância de adotar soluções padronizadas e acessíveis. A separação entre lógica de negócio e de infraestrutura, bem como a organização modular do sistema, surgiram como diferenciais para garantir manutenibilidade e evolução sustentável. Constatou-se que o sucesso na implementação de APIs depende tanto de fatores técnicos quanto de alinhamento com os objetivos organizacionais, tornando essas interfaces ativos estratégicos em ambientes digitais complexos. Ao sistematizar os principais conceitos e práticas descritos na literatura, o estudo oferece um panorama confiável para profissionais e pesquisadores interessados na construção de soluções eficientes para integração de sistemas por meio de APIs.

Palavras-chave: API. Integração de Sistemas. Arquitetura REST. Segurança da Informação. Boas Práticas.

1 INTRODUÇÃO

O avanço da transformação digital impulsionou a necessidade de comunicação contínua e eficiente entre sistemas computacionais, tornando as APIs elementos estratégicos para a integração de soluções tecnológicas heterogêneas, já que essas interfaces permitem que aplicações distintas compartilhem dados e funcionalidades com agilidade, segurança e autonomia, criando ecossistemas interoperáveis que atendem às exigências de escalabilidade e flexibilidade exigidas pelo mercado atual, sendo essa capacidade de conexão o que fundamenta a proposta de APIs REST, baseadas em princípios definidos por Fielding, que estabelecem um estilo arquitetural robusto e amplamente difundido entre empresas e desenvolvedores devido à sua simplicidade na representação de recursos e ao uso de protocolos amplamente conhecidos, como o HTTP, o que permite não apenas integrar sistemas legados e modernos, mas também sustentar a criação de novos modelos de negócios e experiências digitais centradas no usuário (Fielding, 2000).

A adoção de APIs se consolidou como resposta à necessidade de desenvolver interfaces bem definidas e acessíveis a partir de diferentes dispositivos e linguagens de programação, permitindo que empresas consigam manter serviços interligados com alto grau de automação, o que transforma essas interfaces em ferramentas que reduzem custos operacionais, aumentam a escalabilidade dos serviços digitais e ampliam a interoperabilidade entre sistemas de forma segura e consistente, conforme discutido por Jacobson et al., ao indicarem que APIs representam não apenas uma ponte tecnológica, mas um ativo estratégico para a inovação em serviços digitais (Jacobson et al., 2011).

Com o crescimento das soluções mobile e a popularização de serviços em nuvem, as APIs passaram a ser amplamente utilizadas como meio de acesso padronizado a funcionalidades internas de plataformas, o que potencializou o consumo e o fornecimento de dados em tempo real por meio de serviços estruturados, e essa realidade exigiu um novo modelo mental para os desenvolvedores, que precisaram adotar boas práticas de organização de código, separação de responsabilidades e uso de padrões REST para garantir desempenho, segurança e manutenibilidade das aplicações, conforme salientado por Alecrim ao tratar da virtualização e da ubiquidade das aplicações modernas (Alecrim, 2008).

A complexidade dos ambientes corporativos e a heterogeneidade dos sistemas utilizados internamente pressionaram as equipes de desenvolvimento a criarem soluções com base em design modular e padrões arquiteturais que permitam desacoplamento e reusabilidade, sendo esse um dos pilares da chamada Arquitetura Limpa, que se propõe a organizar o sistema de forma que a lógica de negócios permaneça independente dos frameworks, bibliotecas e interfaces externas, favorecendo a testabilidade e o crescimento sustentável do código, abordagem essa reforçada nas boas práticas implementadas por Aguiar em sua API REST educacional (Aguiar, 2024).

Dentro desse cenário, observa-se que as APIs deixam de ser apenas uma interface técnica e passam a ser vistas como contratos formais entre sistemas, exigindo documentação precisa, controle de versões, autenticação robusta e testes automatizados para garantir a continuidade do serviço em ambientes de alta concorrência, e para alcançar esse nível de maturidade técnica, autores como Franklin e Coustan destacam a importância da abstração e da ocultação da lógica interna por trás de endpoints bem definidos, permitindo que o consumidor da API foque apenas na funcionalidade e não na complexidade da implementação (Franklin e Coustan, 2009).

A necessidade de escalabilidade é outro fator determinante no desenvolvimento de APIs modernas, principalmente quando se trata de integrações com grandes volumes de requisições simultâneas, e nesse ponto destaca-se o uso de padrões como o Enterprise Service Bus e os modelos baseados em arquitetura orientada a eventos, os quais promovem maior resiliência e desacoplamento entre os serviços, como abordado por Novais e Stekel ao explorarem uma solução de integração entre sistemas locais e APIs externas, destacando a importância da criptografia RSA e da separação clara entre camadas de lógica e apresentação (Novais e Stekel, 2025).

A segurança, por sua vez, emerge como uma preocupação central em qualquer aplicação baseada em APIs, exigindo desde autenticação forte até políticas de autorização granular que limitem o acesso a recursos de acordo com o perfil do usuário, sendo que as falhas nesse aspecto comprometem não apenas a integridade dos dados, mas também a imagem institucional da organização que fornece a interface, o que levou Campos a ressaltar o uso de mecanismos como API Keys, OAuth e tokens JWT como práticas essenciais para evitar exposições indevidas e garantir a conformidade com legislações como a LGPD (Campos, 2013).

O desempenho das APIs também depende da maneira como os dados são modelados e disponibilizados, sendo essencial pensar na definição dos recursos, no versionamento adequado e na escolha dos formatos de mensagem que priorizem a leveza e a eficiência no tráfego, e nesse sentido, o padrão JSON vem sendo amplamente adotado por sua simplicidade e compatibilidade com diferentes linguagens, sendo essa prática reconhecida como uma das responsáveis pela disseminação do uso de REST em detrimento de abordagens mais pesadas como SOAP, conforme aponta o estudo de Vacari et al. ao abordar a experiência da Embrapa com APIs (Vacari et al., 2016).

Um aspecto recorrente nos projetos bem-sucedidos de integração via API é o foco no consumidor da interface, o que implica em modelar os endpoints não a partir da estrutura interna do sistema, mas da perspectiva de quem vai utilizá-los, o que favorece a clareza, a usabilidade e a reutilização da API por diferentes equipes e aplicações, sendo esse um princípio defendido tanto por Kasthurirathne et al. quanto por Lee et al., ao destacarem o valor de práticas centradas na experiência do desenvolvedor como fator-chave para a adesão e o sucesso da integração (Kasthurirathne et al., 2015; Lee et al., 2014).

A documentação da API também precisa ser pensada como parte do produto, e não como um item acessório, pois é por meio dela que novos desenvolvedores compreenderão os recursos disponíveis, as regras de negócio envolvidas e as restrições técnicas impostas, o que reforça a importância de ferramentas como Swagger e Postman para descrever os contratos e facilitar testes manuais ou automatizados, conforme orienta Moyer ao discutir o ciclo de hype das APIs abertas no setor bancário e a importância da clareza na exposição de funcionalidades (Moyer, 2015).

O monitoramento em tempo real das APIs é outra prática que se torna indispensável para detectar falhas, gargalos e usos indevidos antes que causem impactos graves na operação, e isso inclui a coleta de métricas como tempo de resposta, taxa de erros e consumo por cliente, que permitem ações proativas por parte das equipes técnicas, e esse tipo de solução foi eficazmente implementado no trabalho de Sousa, com uso do Loggly integrado à arquitetura da aplicação, evidenciando o ganho em agilidade e confiabilidade nos processos de suporte e manutenção (Sousa, 2021).

Ainda que os benefícios das APIs sejam amplamente reconhecidos, a sua adoção sem critérios ou o desenvolvimento sem padrões estabelecidos pode resultar em sistemas frágeis, difíceis de manter e propensos a falhas silenciosas que comprometem a experiência do usuário e os objetivos de negócio, e por isso Bacili alerta para a necessidade de planejamento estratégico na exposição de serviços e na gestão do ciclo de vida da API, o que envolve desde a concepção até a sua desativação planejada com notificações aos consumidores (Bacili, 2012).

A escolha entre uma API pública, privada ou parceira depende dos objetivos da organização, do nível de controle desejado e da estratégia de posicionamento no ecossistema digital, sendo comum observar grandes empresas como Google e Facebook optando por expor APIs públicas para fomentar a inovação de terceiros, enquanto outras mantêm interfaces privadas para consumo exclusivo de aplicações internas, abordagem que precisa ser cuidadosamente ponderada com base nos riscos, nas oportunidades e no modelo de governança adotado (Jacobson et al., 2011).

O uso de testes automatizados no desenvolvimento de APIs é outra prática que contribui significativamente para a robustez do sistema, já que permite validar continuamente o comportamento esperado dos endpoints diante de diferentes cenários de entrada, o que reduz o retrabalho, melhora a confiabilidade do código e garante que novas implementações não quebrem funcionalidades anteriores, sendo o TDD uma das abordagens destacadas por Aguiar em sua proposta de API educacional, onde a qualidade interna do software foi assegurada desde os primeiros ciclos de desenvolvimento (Aguiar, 2024).

Sendo assim, observa-se que o desenvolvimento de APIs não é apenas uma tarefa técnica, mas uma atividade estratégica que envolve arquitetura de software, segurança da informação, usabilidade, documentação, testes e governança, e o domínio dessas múltiplas dimensões é o que diferencia uma integração frágil de uma solução robusta e preparada para sustentar o crescimento da organização,

sendo esse o ponto de partida para a reflexão apresentada neste artigo, que busca analisar boas práticas no desenvolvimento de APIs com foco na integração eficiente de sistemas corporativos a partir das contribuições dos principais autores da área.

2 REFERENCIAL TEÓRICO

2.1 INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES (API)

A evolução das tecnologias da informação trouxe consigo a necessidade de desenvolver mecanismos que facilitassem a comunicação entre diferentes sistemas computacionais, e nesse cenário surgem as APIs como elementos centrais na estruturação da interoperabilidade entre softwares, atuando como pontes que conectam plataformas diversas e permitindo que dados e comandos trafeguem com segurança, controle e agilidade, sendo essa funcionalidade essencial para ambientes corporativos que lidam com múltiplos sistemas legados e modernos, o que torna indispensável a padronização da interface e a definição clara de contratos de consumo e entrega de serviços digitais, conforme argumentam Jacobson et al. ao descreverem o papel das APIs como estruturantes da conectividade digital moderna (Jacobson et al., 2011).

Uma API pode ser entendida como uma especificação técnica que define de forma objetiva como um sistema disponibiliza determinadas funcionalidades para serem utilizadas por outras aplicações, abstraindo a complexidade da implementação interna e permitindo que o desenvolvedor utilize esses recursos sem precisar compreender os detalhes do seu funcionamento, conceito amplamente difundido por Franklin e Coustan, que apontam que essa abstração permite foco na experiência de uso e promove rapidez no desenvolvimento de novas soluções, especialmente em contextos empresariais que exigem agilidade na entrega de valor (Franklin e Coustan, 2009).

A utilização de APIs torna-se ainda mais estratégica quando observamos o impacto que elas causam na estrutura organizacional, pois ao serem incorporadas como parte das plataformas digitais, essas interfaces transformam produtos em serviços acessíveis por diferentes canais, o que gera oportunidades para novos modelos de negócio e parcerias tecnológicas, conforme demonstrado por Bacili, que enfatiza a capacidade das APIs de impulsionar a inovação e a expansão da presença digital das organizações, ao permitir que suas funcionalidades sejam reutilizadas por aplicações terceiras de forma escalável e controlada (Bacili, 2012).

Do ponto de vista técnico, uma API bem estruturada deve seguir princípios sólidos de design, contemplando regras de versionamento, organização semântica dos endpoints, padronização dos formatos de entrada e saída e documentação clara, pois esses aspectos garantem não apenas a interoperabilidade entre aplicações, mas também a longevidade da interface diante de evoluções tecnológicas e de mudanças nos requisitos de negócio, como defende Sousa ao destacar que a

padronização das rotas e a separação clara das funcionalidades por meio de módulos favorecem a manutenção e a escalabilidade do sistema (Sousa, 2021).

As APIs são classificadas de acordo com seu nível de exposição e controle, podendo ser públicas, privadas ou parceiras, sendo que as públicas são acessíveis por qualquer desenvolvedor e geralmente utilizadas para fomentar inovação externa, enquanto as privadas são restritas a sistemas internos de uma organização e as parceiras são disponibilizadas para consumidores específicos mediante contrato, sendo esse modelo híbrido o mais adotado por empresas que buscam equilibrar segurança, controle e colaboração, conforme observado por Vacari et al. no estudo sobre a adoção de APIs na Embrapa, onde diferentes níveis de exposição foram utilizados para atender às diversas necessidades da instituição (Vacari et al., 2016).

Um aspecto central no desenvolvimento de APIs é a definição dos recursos que serão disponibilizados, e nesse sentido o modelo REST se destaca por organizar os endpoints em torno de entidades do negócio, utilizando verbos HTTP para representar operações como leitura, criação, atualização e exclusão de dados, estrutura essa que promove simplicidade, clareza e compatibilidade com navegadores, dispositivos móveis e outras plataformas digitais, o que explica sua ampla adoção no mercado, como evidenciado por Campos, ao destacar que REST e JSON compõem a base técnica das APIs modernas devido à sua leveza e facilidade de implementação (Campos, 2013).

Com isso, a escolha do formato de mensagens também influencia diretamente na eficiência da API, sendo o JSON o mais utilizado por sua estrutura leve, legibilidade e fácil manipulação em diferentes linguagens de programação, embora o XML ainda seja aplicado em contextos específicos que demandam maior complexidade estrutural ou aderência a padrões mais antigos, sendo essa decisão impactante tanto no desempenho quanto na experiência de desenvolvimento e manutenção, como explorado por Novais e Stekel ao descreverem a construção de uma API segura e performática baseada em formatos padronizados e criptografia RSA (Novais e Stekel, 2025).

A segurança é outro pilar essencial nas APIs, principalmente diante da crescente exposição de dados sensíveis e da necessidade de autenticação e autorização em ambientes multiclientes, sendo recomendadas práticas como uso de tokens JWT, verificação por OAuth 2.0, aplicação de rotinas de rate limiting e monitoramento de atividades suspeitas, visto que qualquer brecha pode ser explorada para comprometer sistemas inteiros, como apontado por Alecrim ao tratar da responsabilidade dos desenvolvedores na construção de interfaces seguras e alinhadas às normas de proteção de dados pessoais (Alecrim, 2008).

Uma dificuldade recorrente no desenvolvimento de APIs é o versionamento, uma vez que mudanças na estrutura de dados ou nos endpoints podem quebrar integrações existentes e causar indisponibilidades em serviços dependentes, por isso é fundamental definir estratégias claras de versionamento, como incluir a versão no caminho da URL, utilizar cabeçalhos HTTP ou query strings,

de modo a garantir compatibilidade retroativa e permitir a evolução da API sem prejuízo aos consumidores, conforme exemplificado por Campos ao apresentar diferentes formas de versionamento em seus estudos sobre boas práticas em APIs REST (Campos, 2013).

Ressalta-se que a experiência do desenvolvedor que consome a API deve ser considerada como um fator de qualidade do produto, pois APIs com documentação clara, exemplos práticos, sandbox de testes e respostas consistentes facilitam o entendimento e reduzem o tempo de integração, promovendo uma adoção mais rápida e segura, sendo essa uma das recomendações centrais de Lee et al., que reforçam que APIs bem projetadas são aquelas que priorizam a clareza e a previsibilidade no uso, atributos que impactam diretamente na produtividade das equipes de desenvolvimento e no sucesso das integrações (Lee et al., 2014).

Além da clareza técnica, a confiabilidade é construída a partir de práticas como o uso de testes automatizados, que verificam o comportamento esperado dos endpoints diante de diferentes entradas e cenários, garantindo que alterações futuras não comprometam funcionalidades existentes, sendo o TDD uma abordagem amplamente recomendada para esse fim, como demonstrado por Aguiar em seu projeto de API educacional, onde o uso de testes desde a concepção do sistema aumentou a estabilidade e reduziu significativamente os erros em produção (Aguiar, 2024).

A governança de APIs é outro tema relevante, pois envolve o controle de acesso, o monitoramento de uso, a definição de políticas de segurança e a gestão do ciclo de vida da interface, desde a concepção até sua eventual descontinuação, sendo práticas fundamentais para manter a ordem, prevenir abusos e assegurar que as APIs atendam às metas de negócio, como destacado por Moyer ao analisar o uso de APIs abertas no setor bancário e os desafios de mantê-las seguras e eficientes em ambientes altamente regulados e competitivos (Moyer, 2015).

Com a implementação de métricas de consumo e desempenho permite que as equipes técnicas tomem decisões com base em dados concretos, ajustando recursos de infraestrutura, identificando gargalos, priorizando melhorias e antecipando falhas, prática essa evidenciada no trabalho de Sousa ao incorporar soluções de monitoramento como Loggly em sua aplicação, proporcionando uma visão em tempo real do comportamento da API e permitindo intervenções rápidas em caso de falhas ou uso anômalo por parte dos consumidores (Sousa, 2021).

Além disso, a modelagem das APIs deve considerar a necessidade de desacoplamento entre o backend e os consumidores, permitindo que diferentes aplicações utilizem os mesmos serviços sem impactar diretamente o núcleo do sistema, o que requer planejamento arquitetural baseado em princípios como separação de camadas, uso de DTOs e abstração de lógica de negócios, garantindo que mudanças em uma camada não exijam reestruturações em outras, abordagem destacada por Vacari et al. ao relatar a experiência da Embrapa no desenvolvimento de APIs modulares e sustentáveis (Vacari et al., 2016).

Ao se observar o panorama geral, fica evidente que a Interface de Programação de Aplicações deixou de ser um recurso técnico restrito a desenvolvedores e passou a ocupar papel estratégico nas organizações, sendo o elo entre sistemas, plataformas, parceiros e usuários finais, o que reforça a importância de tratá-la com o mesmo rigor e planejamento de qualquer outro ativo de negócio, assumindo que seu bom funcionamento impacta diretamente na experiência do cliente, na performance dos serviços e na reputação da marca, sendo essa a base que sustenta o desenvolvimento responsável e eficiente de APIs voltadas à integração de sistemas complexos e distribuídos.

2.2 ARQUITETURA DE SOFTWARE E MODELOS ARQUITETURAIS

A arquitetura de software corresponde à estrutura fundamental de um sistema, composta por seus componentes, pelas relações entre eles e pelos princípios orientadores que influenciam seu design e sua evolução, sendo essa definição amplamente aceita entre especialistas que defendem que o planejamento arquitetural deve preceder qualquer decisão de codificação, já que estabelece os alicerces sobre os quais todo o sistema será sustentado, e segundo Booch, citado por Aguiar, a função de um bom software é fazer o complexo parecer simples, o que só é possível quando a arquitetura promove clareza, modularidade e coesão entre os elementos do sistema (Booch apud Aguiar, 2024).

Dentre os modelos arquiteturais mais utilizados, destacam-se aqueles que favorecem o desacoplamento, a reutilização de componentes e a manutenção evolutiva da aplicação, como é o caso da arquitetura em camadas, da arquitetura orientada a serviços e da arquitetura orientada a eventos, sendo essas estruturas aplicadas de acordo com a natureza do projeto, as demandas do negócio e os requisitos de escalabilidade e segurança, conforme discutido por Novais e Stekel ao adotarem um modelo baseado em padrões modernos como Factory e Strategy para garantir flexibilidade na integração entre sistemas de gestão e APIs externas (Novais e Stekel, 2025).

A escolha do modelo arquitetural afeta diretamente a performance, a segurança e a manutenibilidade da aplicação, o que exige dos arquitetos de software a habilidade de mapear não apenas as necessidades técnicas, mas também os objetivos estratégicos da organização, garantindo que a solução proposta atenda aos critérios de qualidade exigidos pelo domínio em que será aplicada, conforme ressaltado por Campos, que destaca que muitas falhas em sistemas complexos decorrem de decisões arquiteturais mal fundamentadas, tomadas sem análise adequada do impacto de longo prazo no ciclo de vida da aplicação (Campos, 2013).

A arquitetura limpa, proposta por Robert C. Martin e aplicada por Aguiar em seu trabalho, estabelece uma organização de camadas concêntricas que separa claramente as regras de negócio das implementações técnicas, promovendo independência entre as partes e facilitando testes, manutenção e extensibilidade, sendo essa abordagem ideal para aplicações que precisam evoluir sem comprometer sua integridade interna, e nesse modelo, o núcleo do sistema permanece isolado de detalhes como

frameworks, bancos de dados e interfaces externas, o que confere ao sistema maior resiliência a mudanças tecnológicas (Aguiar, 2024).

Além disso, outro conceito relevante é o de separação de responsabilidades, também conhecido como Separation of Concerns (SoC), que prega que cada módulo do sistema deve ter uma responsabilidade única e bem definida, evitando sobrecarga de funções e reduzindo o acoplamento entre componentes, sendo essa uma das diretrizes centrais da arquitetura orientada a serviços e das APIs REST, pois permite que os serviços sejam desenvolvidos, testados e implantados de forma independente, o que favorece a escalabilidade e a distribuição de carga em ambientes de alta disponibilidade, como demonstra o trabalho desenvolvido na Embrapa com o SIExp (Vacari et al., 2016).

A definição da arquitetura deve considerar ainda os requisitos não funcionais do sistema, como desempenho, disponibilidade, segurança, usabilidade e portabilidade, pois são esses atributos que determinam a qualidade percebida pelo usuário e a viabilidade técnica do projeto, sendo comum que arquiteturas mal dimensionadas comprometam a performance da aplicação ou dificultem a integração com outras plataformas, como alerta Alecrim ao tratar dos desafios enfrentados em ambientes corporativos que precisam suportar múltiplos acessos simultâneos, demandas em tempo real e integração com sistemas legados (Alecrim, 2008).

Modelos como o Client-Server, o MVC (Model-View-Controller) e o Microservices têm sido amplamente utilizados na construção de sistemas distribuídos, cada qual com suas vantagens e limitações, sendo os microsserviços especialmente relevantes no contexto atual por permitirem que diferentes partes da aplicação sejam desenvolvidas e escaladas de forma independente, utilizando tecnologias distintas e garantindo maior flexibilidade na gestão do ciclo de vida dos componentes, característica essa valorizada por Sousa ao implementar uma API modular com painéis de monitoramento e persistência em bancos de dados relacionais e não relacionais (Sousa, 2021).

Por outro lado, a definição do padrão arquitetural também está diretamente ligada à forma como a segurança será tratada, pois sistemas que compartilham dados sensíveis com múltiplos consumidores precisam garantir que a arquitetura permita mecanismos de autenticação, autorização e rastreamento das operações, sendo o uso de camadas intermediárias, gateways e proxies recursos comuns nesse cenário, como citado por Novais e Stekel ao detalharem a aplicação de criptografia RSA e isolamento de camadas lógicas para garantir confidencialidade e integridade das informações trocadas entre sistemas (Novais e Stekel, 2025).

A adoção de boas práticas arquiteturais implica também na formalização de padrões internos, na escolha criteriosa de ferramentas e na definição de convenções que orientem o desenvolvimento das equipes, garantindo que todos os envolvidos tenham clareza sobre as diretrizes do projeto e possam colaborar de maneira alinhada, reduzindo retrabalho, conflitos de versão e erros de integração, e para

que essa governança seja eficaz, Campos sugere a elaboração de guias de estilo e a realização de revisões de arquitetura como parte do processo de validação contínua da aplicação (Campos, 2013).

A documentação da arquitetura deve ser tratada como um artefato vivo, atualizado ao longo da evolução do projeto e acessível a todas as partes interessadas, pois é por meio dela que novos desenvolvedores compreendem a lógica do sistema, identificam pontos de extensão e compreendem os fluxos de dados e controle, o que facilita a manutenção corretiva e evolutiva, como aponta Vacari et al., ao relatar que a ausência de documentação clara dificultava a reutilização de módulos e a interoperabilidade com sistemas externos no início do projeto SIExp, sendo essa lacuna superada com a adoção de padrões arquiteturais e técnicos bem documentados (Vacari et al., 2016).

A integração entre sistemas baseados em arquiteturas distintas exige a existência de pontos de contato bem definidos e protegidos, o que normalmente se concretiza por meio de APIs e middlewares que atuam como tradutores e mediadores entre aplicações, sendo essas soluções essenciais para empresas que utilizam softwares de diferentes fornecedores ou que passaram por processos de fusão e precisam unificar suas operações digitais, realidade que foi enfrentada na Embrapa e que demandou uma reformulação completa da arquitetura de seus sistemas de informação para permitir interoperabilidade plena entre módulos (Vacari et al., 2016).

A arquitetura orientada a eventos é uma alternativa poderosa para aplicações que lidam com fluxos assíncronos e grande volume de mensagens, pois permite que os componentes se comuniquem de forma indireta por meio de filas, tópicos e mecanismos de publish-subscribe, promovendo maior desacoplamento e tolerância a falhas, sendo esse modelo aplicado com sucesso em sistemas bancários, plataformas de e-commerce e aplicações de IoT, como destacam estudos mencionados por Moyer no contexto das APIs abertas e da necessidade de estruturas reativas e resilientes em ambientes regulados e dinâmicos (Moyer, 2015).

Em contextos de alta complexidade, pode ser necessária a combinação de diferentes modelos arquiteturais em um mesmo projeto, como ocorre em arquiteturas híbridas que utilizam microsserviços para funcionalidades críticas e arquitetura monolítica para operações de baixo risco ou que exigem alta coesão, sendo essa flexibilidade fundamental para atender às necessidades específicas de cada domínio, conforme reforça Aguiar ao demonstrar que a aplicação de arquitetura limpa em conjunto com práticas ágeis permitiu a entrega contínua de valor sem comprometer a estrutura central da aplicação (Aguiar, 2024).

A escolha entre centralização e descentralização das responsabilidades arquiteturais também deve ser considerada, pois enquanto estruturas centralizadas promovem uniformidade e controle, arquiteturas descentralizadas permitem maior autonomia das equipes, reduzindo gargalos e acelerando entregas, e o equilíbrio entre esses dois polos depende da maturidade da organização, da complexidade do sistema e da cultura de desenvolvimento adotada, sendo uma decisão estratégica que impacta

diretamente na eficiência e na governança do ambiente de software, como discutido por Lee et al. ao tratarem da gestão de APIs em grandes corporações (Lee et al., 2014).

A arquitetura de software é, portanto, uma disciplina que combina conhecimentos técnicos, visão sistêmica e estratégia de negócio, sendo a base para que aplicações modernas suportem mudanças frequentes, atendam às expectativas dos usuários e se mantenham relevantes em ambientes altamente competitivos, e nesse cenário, a adoção consciente de modelos arquiteturais bem definidos, aliados a boas práticas de documentação, segurança, testes e governança, é o que garante que o sistema não apenas funcione, mas evolua de forma sustentável, segura e alinhada aos objetivos da organização.

2.3 MODELO ARQUITETURAL REST

O modelo arquitetural REST, sigla para Representational State Transfer, foi originalmente proposto por Roy Fielding em sua tese de doutorado como uma alternativa para estruturar aplicações distribuídas com foco na simplicidade, escalabilidade e compatibilidade com a infraestrutura já existente da web, especialmente o protocolo HTTP, e essa proposta se destacou por sua abordagem pragmática que busca representar os recursos do sistema como entidades manipuláveis por meio de operações padronizadas, utilizando verbos como GET, POST, PUT e DELETE, o que viabiliza a construção de interfaces claras, previsíveis e acessíveis a partir de qualquer plataforma que possua conexão com a internet (Fielding, 2000).

Ao contrário de modelos anteriores como o SOAP, que exigem contratos rígidos e uso de protocolos complexos como o XML-RPC, o REST baseia-se na simplicidade da comunicação por meio de URLs e mensagens HTTP estruturadas em formatos leves como o JSON, o que favorece a interoperabilidade entre diferentes linguagens e tecnologias, reduz o tempo de resposta das aplicações e facilita a depuração dos serviços, sendo essa leveza um dos principais fatores que contribuíram para sua rápida difusão em startups, grandes empresas e instituições públicas, como apontado por Campos ao destacar a preferência do mercado por soluções REST em detrimento de arquiteturas mais verborrágicas e difíceis de manter (Campos, 2013).

O princípio fundamental do REST é o conceito de recurso, que representa qualquer entidade significativa para o sistema, sendo acessada por meio de uma URI (Uniform Resource Identifier) única e manipulada por operações HTTP que refletem ações sobre esse recurso, o que promove clareza na organização da interface e facilita a documentação, testes e reutilização de código, permitindo que diferentes consumidores interajam com os mesmos dados de forma padronizada e segura, conforme exemplificado no estudo de Sousa ao descrever a organização dos endpoints da API de e-commerce com base na lógica REST e nos princípios de separação entre responsabilidades (Sousa, 2021).

Outro princípio importante do REST é a ausência de estado entre as requisições, ou seja, cada chamada à API deve conter todas as informações necessárias para seu processamento, sem que o

servidor precise manter histórico de interações anteriores, o que confere escalabilidade à aplicação ao permitir que múltiplas requisições sejam atendidas de forma paralela e independente, característica essa fundamental para sistemas de alta demanda e ambientes distribuídos, sendo esse modelo adotado com sucesso no projeto da Embrapa para viabilizar o consumo simultâneo de serviços por diferentes plataformas de coleta e análise de dados (Vacari et al., 2016).

A uniformidade da interface REST é outro aspecto que contribui para a sua robustez, já que ao restringir o conjunto de operações possíveis e exigir a padronização da estrutura das requisições e respostas, o modelo reduz a complexidade da implementação e da manutenção do sistema, promovendo previsibilidade e facilitando a adoção por novos desenvolvedores, conforme discutido por Franklin e Coustan, que destacam a importância da consistência e da simplicidade como atributos indispensáveis para interfaces que serão consumidas por múltiplas aplicações ao longo do tempo (Franklin e Coustan, 2009).

O uso de hipermídia como motor do estado da aplicação (HATEOAS) é outro princípio do REST que, embora nem sempre implementado, tem como objetivo guiar o consumidor da API por meio de links contidos nas respostas, permitindo que ele descubra dinamicamente os próximos passos possíveis na interação com o sistema, o que adiciona flexibilidade e acoplamento mínimo entre cliente e servidor, sendo essa característica especialmente útil em ambientes com regras de negócio variáveis e integrações entre múltiplos serviços, como sugerido nos estudos apresentados por Jacobson et al. sobre a evolução das APIs públicas em ecossistemas abertos (Jacobson et al., 2011).

O REST também se destaca pela sua compatibilidade com caching, o que significa que determinadas respostas da API podem ser armazenadas localmente para reduzir o número de requisições ao servidor, aumentando a performance do sistema e diminuindo a carga sobre a infraestrutura, e esse mecanismo é viabilizado pela utilização de cabeçalhos HTTP como ETag, Cache-Control e Last-Modified, que controlam a validade e a atualização das respostas, sendo essas práticas amplamente recomendadas para aplicações que lidam com grande volume de acessos e dados não sensíveis, como observado por Moyer em sua análise sobre APIs em setores financeiros e governamentais (Moyer, 2015).

O versionamento de APIs REST é uma necessidade recorrente, já que sistemas evoluem, novas funcionalidades são adicionadas e mudanças estruturais se tornam inevitáveis, sendo boas práticas incluir a versão no caminho da URL ou nos cabeçalhos das requisições, de modo a evitar a quebra de contratos existentes e garantir que consumidores antigos continuem funcionando enquanto novos recursos são disponibilizados, sendo esse controle de versões essencial para manter a estabilidade do ecossistema e facilitar a transição gradual entre versões, como exemplificado por Campos em suas demonstrações sobre versionamento por path e media type (Campos, 2013).

Já a documentação de APIs REST deve ser clara, atualizada e abrangente, contendo informações sobre os recursos disponíveis, os parâmetros esperados, os formatos de resposta e os códigos de status HTTP retornados, pois é por meio dela que os desenvolvedores compreenderão como utilizar corretamente os serviços e evitarão erros que poderiam comprometer o desempenho e a segurança da aplicação, sendo recomendada a utilização de ferramentas como Swagger (OpenAPI) para padronizar e automatizar essa documentação, prática destacada por Vacari et al. ao discutirem os desafios enfrentados na adoção das APIs da Embrapa em múltiplos módulos internos (Vacari et al., 2016).

A utilização de códigos de status HTTP padronizados é outro aspecto crucial no modelo REST, pois permite que o cliente compreenda o resultado da requisição de forma objetiva, facilitando o tratamento de erros e a construção de fluxos de lógica adequados, e entre os códigos mais comuns estão o 200 para sucesso, 201 para criação de recurso, 400 para erro de validação, 401 para autenticação inválida, 403 para acesso negado e 500 para erro interno, sendo fundamental que esses códigos sejam utilizados corretamente, conforme exemplificado por Sousa ao demonstrar a validação de endpoints e o tratamento adequado de falhas no serviço de e-commerce (Sousa, 2021).

A escolha do formato das mensagens trocadas entre cliente e servidor impacta diretamente na legibilidade, no desempenho e na compatibilidade da API, sendo o JSON o formato preferido por sua simplicidade e ampla aceitação, embora em alguns casos o XML ainda seja necessário para atender a padrões específicos de interoperabilidade, e independentemente do formato adotado, é importante definir claramente os campos obrigatórios, os tipos de dados e a estrutura esperada da resposta, como detalhado por Campos ao apresentar exemplos de design de representações em APIs REST bem projetadas (Campos, 2013).

A segurança das APIs REST depende da aplicação de boas práticas como a autenticação via tokens, a utilização de conexões HTTPS, o controle de acesso baseado em papéis e a validação de entradas para evitar ataques como injeção de comandos, e essas medidas são essenciais para proteger os dados em trânsito e garantir que apenas usuários autorizados acessem os recursos, sendo essa abordagem adotada por Novais e Stekel em seu modelo de integração escalável, que utilizou criptografia RSA e camadas intermediárias de segurança para proteger as interfaces críticas da aplicação (Novais e Stekel, 2025).

Com a integração de APIs REST com bancos de dados relacionais ou não relacionais exige atenção à consistência dos dados, à normalização das tabelas e à performance das consultas, pois a forma como os dados são estruturados e acessados impacta diretamente na resposta da API e na experiência do consumidor, e para isso é fundamental planejar a arquitetura da aplicação levando em conta o volume de dados, a frequência de acesso e os padrões de leitura e escrita, prática exemplificada

por Sousa ao descrever a separação entre o banco relacional para produtos e o banco não relacional para autenticação (Sousa, 2021).

A capacidade de monitorar e auditar as chamadas feitas à API é essencial para manter a confiabilidade e a transparência do sistema, permitindo identificar padrões de uso, detectar abusos e diagnosticar falhas em tempo real, e para isso ferramentas como Loggly, Prometheus e ELK Stack são amplamente utilizadas por permitirem o acompanhamento contínuo das operações, o que melhora a tomada de decisão e reduz o tempo de resposta em caso de incidentes, sendo essa abordagem aplicada com êxito por Sousa em seu painel de monitoramento da aplicação (Sousa, 2021).

O modelo arquitetural REST, ao aliar simplicidade, flexibilidade e compatibilidade com os padrões da web, consolidou-se como o paradigma dominante na construção de APIs modernas, tornando-se a escolha natural para projetos que exigem comunicação eficiente, escalabilidade e fácil integração com outras plataformas, e embora nem todas as implementações sigam estritamente todos os seus princípios, o respeito às suas diretrizes fundamentais garante interfaces robustas, compreensíveis e evolutivas, sendo essa a base sobre a qual se sustenta grande parte das soluções digitais contemporâneas.

3 METODOLOGIA

O presente estudo foi elaborado com base em uma abordagem qualitativa, fundamentando-se em revisão bibliográfica e análise de fontes científicas disponíveis em bases de dados eletrônicas nacionais e internacionais, com o objetivo de investigar e discutir as boas práticas no desenvolvimento de APIs voltadas à integração de sistemas, priorizando a identificação de conceitos recorrentes, modelos de arquitetura, desafios técnicos e soluções aplicadas em diferentes contextos corporativos, a fim de consolidar um referencial teórico que respalde decisões técnicas e estratégicas para o uso eficiente dessas interfaces de programação no ambiente organizacional.

A seleção dos documentos foi realizada por meio de busca dirigida nas bases Google Scholar, Scielo, BDTD e ResearchGate, utilizando os descritores combinados "API", "integração de sistemas", "REST", "arquitetura de software" e "boas práticas", de forma a abranger conteúdos que tratassem da implementação de interfaces, do versionamento de serviços, da segurança da comunicação e da organização modular de componentes, além de aspectos como documentação, testes e monitoramento de desempenho, os quais constituem elementos fundamentais para a construção de sistemas interoperáveis.

Foram considerados critérios de inclusão a pertinência direta ao tema proposto, a presença de autores citados dentro dos textos analisados, a clareza metodológica das obras e a credibilidade das instituições vinculadas às produções acadêmicas, enquanto os critérios de exclusão envolveram publicações que tratassem de tópicos tangenciais sem aprofundamento técnico, materiais opinativos

desprovidos de fundamentação teórica e textos que não apresentassem referências documentadas ou que estivessem desatualizados em relação às práticas atuais do desenvolvimento de APIs.

O procedimento metodológico envolveu a leitura exploratória inicial de todo o material encontrado, seguida da leitura seletiva e analítica dos textos que atenderam aos critérios estabelecidos, com extração e organização das informações relevantes por meio de fichamentos temáticos e síntese das contribuições teóricas identificadas, o que possibilitou a construção de um corpo teórico consistente, capaz de embasar a discussão crítica das boas práticas e dos desafios enfrentados na aplicação de APIs como mecanismos de integração entre plataformas heterogêneas.

4 RESULTADOS E DISCUSSÃO

Os resultados do estudo revelou uma convergência significativa entre os autores quanto à importância das boas práticas no desenvolvimento de APIs para integração de sistemas, especialmente quando essas interfaces são concebidas com base nos princípios do modelo REST, que privilegia a simplicidade estrutural, a independência das operações e a compatibilidade com os padrões da web, sendo esse consenso observado tanto em aplicações voltadas a plataformas educacionais, como no trabalho de Aguiar, quanto em sistemas de gestão empresarial, como descrito por Novais e Stekel em seu estudo prático sobre integração segura e escalável com uso de criptografia assimétrica (Aguiar, 2024; Novais e Stekel, 2025).

Os resultados evidenciaram que o uso do padrão REST associado ao formato JSON promove uma experiência mais fluida de consumo e manutenção da API, ao passo que permite maior compatibilidade entre diferentes linguagens de programação e ambientes operacionais, sendo essa estrutura reconhecida como eficaz por Campos, que apresentou uma comparação entre diferentes estilos de desenvolvimento e concluiu que a adoção disciplinada de REST, com versionamento e respostas padronizadas, reduz significativamente os erros de comunicação entre sistemas e melhora a eficiência operacional (Campos, 2013).

Algo comum entre os autores analisados é a valorização do versionamento como estratégia para preservar a integridade das integrações ao longo do tempo, sendo as abordagens que utilizam o caminho da URL, os headers HTTP e os media types apontadas como as mais comuns, com destaque para a clareza proporcionada pela inclusão explícita da versão na URI, como demonstrado por Sousa em seu projeto de e-commerce, onde as rotas foram cuidadosamente organizadas para permitir a coexistência de diferentes versões sem impacto negativo no consumo da aplicação pelos clientes (Sousa, 2021).

A documentação das APIs também foi apontada como um componente indispensável para o sucesso de qualquer integração, especialmente quando o objetivo é permitir que diferentes times, empresas ou sistemas externos acessem os serviços de forma segura e previsível, e nesse aspecto, os

trabalhos analisados reforçam a necessidade de utilizar ferramentas como Swagger para padronizar os contratos de interface e permitir testes rápidos e confiáveis, sendo esse fator determinante na experiência de desenvolvedores terceirizados, como discutido por Vacari et al. no estudo sobre a Embrapa, onde a clareza documental foi um diferencial na adoção das APIs por múltiplos sistemas internos (Vacari et al., 2016).

A segurança foi tratada como uma preocupação central em todos os textos, especialmente nos que lidam com dados sensíveis ou operam em ambientes com múltiplos níveis de acesso, e nesse ponto, as boas práticas recomendadas envolvem autenticação baseada em tokens, autorização por perfis, utilização de HTTPS, aplicação de limites de requisição e registro das atividades por meio de logs auditáveis, sendo essa estrutura evidenciada por Novais e Stekel, que demonstraram o uso de criptografia RSA como diferencial para assegurar a integridade das informações trafegadas entre os sistemas integrados (Novais e Stekel, 2025).

No que se refere ao monitoramento das APIs, os autores convergem na necessidade de estabelecer ferramentas e métricas que permitam acompanhar o desempenho das requisições, o tempo de resposta, os erros mais frequentes e os picos de consumo, sendo o uso de plataformas como Loggly citado por Sousa como uma solução eficaz para identificar falhas em tempo real e agir preventivamente, o que demonstra que a visibilidade operacional é tão importante quanto a robustez técnica da interface, já que qualquer indisponibilidade pode gerar prejuízos financeiros e perda de credibilidade para o serviço oferecido (Sousa, 2021).

Ao tratar da estrutura interna das APIs, os textos analisados demonstram a importância de adotar modelos arquiteturais que favoreçam o desacoplamento e a separação de responsabilidades, como a arquitetura limpa ou orientada a eventos, pois esses padrões permitem maior flexibilidade na evolução do sistema e reduzem os riscos de regressão em implementações futuras, sendo esse princípio aplicado com sucesso no trabalho de Aguiar, onde a organização modular da API permitiu testes automatizados e entregas contínuas sem comprometer a estabilidade do sistema (Aguiar, 2024).

O princípio de separação entre lógica de negócios e lógica de apresentação foi reforçado por diversos autores como fator essencial para garantir a clareza do código e facilitar a manutenção da aplicação, o que se reflete na organização dos controladores, serviços e repositórios das APIs estudadas, como exemplificado por Campos ao apresentar exemplos práticos de segmentação funcional dentro de projetos baseados em REST, permitindo que cada componente do sistema fosse responsável por uma tarefa específica e independente das demais (Campos, 2013).

A adoção de boas práticas em testes automatizados foi outro fator recorrente entre os materiais analisados, com ênfase no uso de TDD (Desenvolvimento Orientado a Testes) como forma de garantir a integridade do código desde os primeiros estágios do desenvolvimento, prática implementada por Aguiar como elemento estruturante do seu projeto, possibilitando a construção de uma API

educacional confiável e resiliente, com verificação contínua das funcionalidades e resposta rápida a eventuais falhas identificadas nos ciclos de desenvolvimento (Aguiar, 2024).

Do ponto de vista da experiência do desenvolvedor que consome a API, os resultados apontam que a clareza na estrutura das rotas, a previsibilidade das respostas e a padronização das mensagens são aspectos que influenciam diretamente na adoção da interface e na redução do tempo necessário para integrar novos serviços, sendo essas qualidades obtidas por meio da aplicação consistente dos princípios REST, como defendido por Franklin e Coustan, que indicam que o verdadeiro valor de uma API está na sua capacidade de ser compreendida e utilizada com facilidade por qualquer profissional com conhecimentos básicos sobre protocolos da web (Franklin e Coustan, 2009).

A interoperabilidade entre sistemas heterogêneos foi abordada como uma das grandes conquistas promovidas pelas APIs, especialmente quando estas são desenvolvidas com foco na compatibilidade de dados e no uso de padrões abertos, sendo essa característica decisiva para o sucesso de projetos que envolvem múltiplos departamentos, fornecedores ou clientes, como ocorreu no caso da Embrapa, onde a adoção das APIs permitiu a reutilização de dados e a construção de novos módulos sem a necessidade de reescrever funcionalidades já existentes, gerando economia e agilidade para a organização (Vacari et al., 2016).

O alinhamento entre os objetivos técnicos das APIs e as estratégias de negócio das organizações foi identificado como fator crítico para o sucesso dos projetos de integração, pois a API deixa de ser apenas um canal de comunicação e passa a ser vista como uma peça estratégica que viabiliza novos modelos de serviços, amplia canais de atendimento e melhora a entrega de valor ao cliente, como afirmado por Bacili, que descreve a API como a materialização digital das competências centrais da empresa, ao permitir que suas funcionalidades sejam expostas de forma estruturada e reutilizável em múltiplos contextos (Bacili, 2012).

Os resultados também demonstraram que a participação ativa de equipes multidisciplinares no processo de concepção, desenvolvimento e manutenção das APIs contribui para a criação de interfaces mais robustas, seguras e alinhadas às necessidades reais dos usuários, pois a diversidade de visões e experiências permite antever cenários de uso variados e ajustar os requisitos antes que se tornem gargalos operacionais, abordagem valorizada nos projetos analisados, em especial na Embrapa, onde a colaboração entre analistas, pesquisadores e desenvolvedores foi decisiva para o sucesso do SIExp (Vacari et al., 2016).

A discussão dos resultados permite concluir que, embora as boas práticas estejam amplamente disseminadas na literatura técnica, sua aplicação prática depende da maturidade da equipe, da cultura organizacional e da existência de uma governança de TI que valorize a documentação, os testes, a segurança e o monitoramento, sendo que as experiências bem-sucedidas analisadas neste estudo demonstram que a adoção dessas práticas não apenas melhora a qualidade das integrações, mas

também reduz os riscos, os custos de manutenção e aumenta a capacidade de adaptação do sistema frente a novas demandas (Novais e Stekel, 2025).

De forma geral, as evidências coletadas a partir dos materiais analisados indicam que o sucesso no desenvolvimento de APIs voltadas à integração de sistemas depende de uma combinação equilibrada entre conhecimento técnico, planejamento arquitetural, alinhamento com o negócio e aplicação sistemática de boas práticas, sendo esse conjunto de fatores o que diferencia projetos frágeis de soluções sustentáveis e preparadas para o crescimento, sustentando a tese de que a API não é apenas uma interface, mas um recurso estratégico que articula tecnologia e resultado dentro das organizações que operam na era digital.

5 CONSIDERAÇÕES FINAIS

Com o desenvolvimento de APIs voltadas à integração de sistemas deixou de ser uma demanda técnica isolada para se tornar um elemento central na construção de ecossistemas digitais modernos, exigindo planejamento estratégico, conhecimento aprofundado sobre arquitetura de software e comprometimento com boas práticas desde as fases iniciais do projeto até a sua manutenção em ambientes de produção.

Ao longo da análise realizada neste estudo, foi possível constatar que a adoção de princípios como clareza nos endpoints, padronização das respostas, uso de versionamento adequado e aplicação de mecanismos robustos de segurança não apenas facilitam a integração entre plataformas distintas, como também contribuem para a escalabilidade, a confiabilidade e a manutenibilidade dos sistemas envolvidos.

A abordagem REST mostrou-se como o modelo mais amplamente adotado e valorizado pelas aplicações modernas, tanto pela sua compatibilidade com os padrões da web quanto pela simplicidade na definição e no consumo dos recursos disponibilizados, sendo essa escolha arquitetural uma das responsáveis pela popularização das APIs em contextos empresariais diversos, desde plataformas educacionais até sistemas de gestão corporativa.

Foi possível observar ainda que, para que as APIs cumpram seu papel com eficiência, é imprescindível que elas sejam acompanhadas de documentação acessível, clara e atualizada, capaz de orientar os desenvolvedores no consumo das funcionalidades oferecidas, minimizando dúvidas, evitando erros e promovendo agilidade nos processos de integração entre sistemas heterogêneos.

A manutenção da qualidade das APIs também depende de práticas estruturadas de testes automatizados, que assegurem a integridade dos serviços frente a atualizações e mudanças nos requisitos do sistema, permitindo que as evoluções ocorram de maneira segura e previsível, preservando a estabilidade da comunicação entre os sistemas envolvidos.

Foi identificado um fator decisivo como o monitoramento contínuo das requisições e respostas, com coleta de métricas, análise de logs e rastreamento de falhas em tempo real, prática que possibilita a identificação antecipada de gargalos ou falhas críticas, aumentando a capacidade de resposta das equipes técnicas e reduzindo o impacto negativo de eventuais indisponibilidades.

A arquitetura limpa e modular também se destacou como diferencial positivo, permitindo que o sistema seja mantido e expandido com menos esforço, já que a separação clara entre lógica de negócios e lógica técnica garante flexibilidade e facilita a compreensão do funcionamento interno da aplicação por parte de diferentes profissionais.

O estudo demonstrou que o sucesso das APIs não está restrito ao seu desempenho técnico, mas também à sua capacidade de alinhar-se aos objetivos estratégicos da organização, tornando-se facilitadoras da inovação, da abertura de novos canais de serviço e da otimização de processos internos e externos por meio da conectividade digital.

No entanto, o desenvolvimento de APIs precisa ser tratado como uma prática multidisciplinar, que envolve tanto competências técnicas quanto visão de negócio, sensibilidade à experiência do usuário e comprometimento com a qualidade contínua, sendo essa compreensão essencial para transformar as APIs em ativos estratégicos e sustentáveis.

Portanto, a aplicação de boas práticas no desenvolvimento de APIs para integração de sistemas não só fortalece a infraestrutura tecnológica das organizações, como também amplia sua capacidade de inovação, adaptação e geração de valor no cenário competitivo atual, caracterizado pela interdependência entre serviços, plataformas e dados em escala global.

REFERÊNCIAS

ALECRIM, Emerson. Computação em nuvem: entendendo a tecnologia que está mudando a forma de trabalhar. Rio de Janeiro: Nova Terra, 2008.

BACILI, Luiz Fernando. Computação em nuvem e as transformações no mercado corporativo. São Paulo: Editora Érica, 2012.

CAMPOS, Marcos Vinicius de Souza. Desenvolvimento de APIs baseadas em REST. 2013. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Centro Paula Souza, Campinas, 2013.

FRANKLIN, Curtis; COUSTAN, Dave. APIs: do básico à prática. São Paulo: Bookman, 2009.

GIL, Antônio Carlos. Como elaborar projetos de pesquisa. 7. ed. São Paulo: Atlas, 2019.

JACOBSON, Ivar et al. Software Engineering: a practitioner's approach. New York: Addison-Wesley, 2011.

KASTHURIRATHNE, Shantha et al. Enhancing healthcare interoperability with APIs. Journal of Biomedical Informatics, v. 55, p. 199–209, 2015.

LAKATOS, Eva Maria; MARCONI, Marina de Andrade. Fundamentos de metodologia científica. 8. ed. São Paulo: Atlas, 2017.

LEE, Minsuk et al. Design and Evaluation of APIs. ACM Computing Surveys, v. 46, n. 4, p. 1–38, 2014.

MOYER, Stephen. Hype Cycle for Open Banking APIs, Apps and App Stores. Gartner, 2015.

NOVAIS, Ruth Queiroz da Costa; STEKEL, Tardelli Ronan Coelho. Integração escalável e segura de sistemas de gestão com APIs: uma abordagem prática. Revista Foco, v. 18, n. 2, p. 1–10, 2025.

SOUSA, Vinícius Rodrigues de. Desenvolvimento e monitoramento de uma API de e-commerce. 2021. Trabalho de Conclusão de Curso (Engenharia de Controle e Automação) – UNESP, Sorocaba, 2021.

VACARI, Isaque; APOLINÁRIO, Daniel Rodrigo de Freitas; QUEIROS, Leonardo Ribeiro. Um estudo sobre a adoção de APIs: caso do Sistema de Informação de Experimentos da Embrapa (SIExp). Campinas: Embrapa Informática Agropecuária, 2016. (Comunicado Técnico, 123).

WEBBER, Jim. REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media, 2010.

WILSON, Jeff. API Economy: Opening New Doors for Innovation and Value. Red Hat Publications, 2014.