UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

VICTOR CAIO ARAUJO SIQUEIRA

ANÁLISE COMPARATIVA DE FERRAMENTAS LOW-CODE PARA DESENVOLVIMENTO DE SOFTWARE

QUIXADÁ

2024

VICTOR CAIO ARAUJO SIQUEIRA

ANÁLISE COMPARATIVA DE FERRAMENTAS LOW-CODE PARA DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em sistemas de informação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em sistemas de informação.

Orientador: Prof. Dr. Regis Pires Magalhães.

QUIXADÁ

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S632 Siqueira, Victor Caio Araujo.
Análise comparativa de ferramentas low-code para desenvolvimento de software / Victor Caio Araujo Siqueira. – 2024.
48 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2024.
Orientação: Prof. Dr. Regis Pires Magalhães.

1. low-code. 2. desenvolvimento de software. 3. backend. 4. plataformas de desenvolvimento. 5. comparação entre plataformas low-code. I. Título.

CDD 004

VICTOR CAIO ARAUJO SIQUEIRA

ANÁLISE COMPARATIVA DE FERRAMENTAS LOW-CODE PARA DESENVOLVIMENTO DE SOFTWARE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em sistemas de informação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em sistemas de informação.

Aprovada em: 27/09/2024.

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Bruno Góis Mateus
Universidade Federal do Ceará (UFC)

Prof. Dr. Sidartha Azevedo Lobo de Carvalho
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Primeiramente, obrigado Deus por todas as bençãos que me permitiram estar aqui. Segundamente, quero agradecer a minha mãe, por tudo até aqui, obrigado por sempre me apoiar, também quero agradecer a minha companheira por sempre estar ao meu lado, me apoiando e me motivando a alcançar meus objetivos. Meus sinceros obrigado a Dona Raimunda, Seu Eudes, Tia Kelma e minha irmã Victória por todo o suporte e toda ajuda na minha caminhada universitária, só tenho a agradecer a vocês, espero um dia por retribuir tudo o que fizeram por mim.

Agradeço ao Prof. Dr. Regis Pires Magalhães por ter me aceitado como orientando no Trabalho de Conclusão de Curso (TCC) e pelo apoio e paciência ao longo desse desafio.

Aos professores participantes da banca examinadora Bruno Góis Mateus, Davi Romero de Vasconcelos e Sidartha Azevedo Lobo de Carvalho pelo tempo, pelas valiosas colaborações e sugestões.

Ao amigo Gabriel Holanda Raulino, que o TCC me trouxe, cuja contribuição foi fundamental nesse processo, oferecendo ajuda e apoio nos momentos difíceis. Sou grato por toda a colaboração. No final, deu tudo certo, e juntos conseguimos superar essa etapa.

Meus agradecimentos também vão a todos os servidores e professores da UFC campus Quixadá, que dia após dia me inspiraram a continuar na minha jornada pela vontade e paixão transmitida em suas tarefas e orientações, obrigado por me inspiraram a ser um excelente profissional.

Também gostaria de agradecer aos meus amigos que fiz ao longo da graduação, sem vocês a minha trajetória não teria sido a mesma nem tão satisfatória quanto foi.

"Tudo o que um sonho precisa para ser realizado é alguém que acredite que ele possa ser realizado." (Roberto Shinyashiki)

RESUMO

O desenvolvimento de software tem passado por grandes transformações, com o aumento da demanda por soluções ágeis e eficientes, e a escassez de desenvolvedores qualificados. Nesse contexto, as plataformas *low-code* emergem como uma alternativa promissora, permitindo o desenvolvimento de aplicativos com pouca ou nenhuma codificação, sendo assim, possibilitando uma solução para essa realidade. Este trabalho realizou uma análise comparativa de diferentes plataformas de desenvolvimento *low-code*, como Directus, Strapi, Parse, Supabase e Firebase, visando identificar a mais adequada para o desenvolvimento de soluções de *backend*. Utilizando a metodologia de implementação prática, foram avaliadas características como simplicidade de configuração, desempenho, curva de aprendizado e tamanho da comunidade. A pesquisa destacou os benefícios das plataformas *low-code*, como desempenho, documentação e facilidade de uso, mas também apontou limitações, como escalabilidade e fragmentação entre as plataformas. Firebase e Supabase possuem a limitação de não ser tão customizável como Directus, Strapi e Parse. Os resultados indicam que, apesar das vantagens, a escolha da ferramenta ideal deve considerar o contexto específico do projeto, sendo essencial uma análise criteriosa das necessidades e funcionalidades disponíveis. Este estudo contribui para o entendimento das possibilidades e desafios das plataformas *low-code* no desenvolvimento de software, oferecendo recursos para a tomada de decisão de desenvolvedores e organizações.

Palavras-chave: low-code; desenvolvimento de software; backend; plataformas de desenvolvimento; comparação entre plataformas low-code.

ABSTRACT

The development of software has undergone significant transformations, with the increasing demand for agile and efficient solutions and the shortage of qualified developers. In this context, low-code platforms emerge as a promising alternative, enabling the development of applications with little to no coding, thus offering a solution to this reality. This study conducted a comparative analysis of different low-code development platforms, such as Directus, Strapi, Parse, Supabase, and Firebase, aiming to identify the most suitable for backend solution development. Using a practical implementation methodology, characteristics such as ease of setup, performance, learning curve, size of the support community, among others, were evaluated. The research highlighted the benefits of low-code platforms, such as agility, cost reduction, and ease of maintenance, but also pointed out limitations, such as scalability and fragmentation across platforms. The results indicate that, despite the advantages, the choice of the ideal tool should consider the specific context of the project, making a careful analysis of the needs and available features essential. This study contributes to the understanding of the possibilities and challenges of low-code platforms in software development, offering resources for decision-making by developers and organizations.

Keywords: low-code; software development; development platforms; backend; comparison between low-code platforms.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, o campo do desenvolvimento de *software* tem passado por uma evolução notável, impulsionada pelo avanço tecnológico e pela necessidade de soluções mais eficientes, ágeis, escaláveis e seguras. A demanda por *software* também tem aumentado significativamente, com empresas buscando soluções inovadoras para se manterem competitivas em um mercado em constante mudança (Mariana Alves, 2024). Nesse contexto, investir em soluções e em profissionais altamente qualificados torna-se essencial para garantir o sucesso e a efetividade dos projetos de software (Brasscom, 2021).

Além disso, as organizações estão cada vez mais focadas em adotar técnicas e metodologias que permitam atender aos prazos e orçamentos estabelecidos (KÄSS *et al.*, 2023). Estudos como o de Richardson *et al.* (2014) destacam a importância de gerenciar eficientemente o desenvolvimento de software, aplicando boas práticas, metodologias ágeis e técnicas de gestão de projetos. Essas abordagens permitem que as empresas maximizem a produtividade, reduzam os riscos e alcancem resultados de alta qualidade.

Portanto, o desenvolvimento de software tornou-se um componente estratégico para as empresas, exigindo uma abordagem multidisciplinar que combina conhecimentos técnicos sólidos, habilidades de gestão e capacidade de adaptação às novas tecnologias (FLEURY *et al.*, 2014). O investimento em profissionais qualificados, a adoção de práticas e de soluções eficientes são fundamentais para o sucesso das empresas nesse ambiente altamente competitivo (Safetec, 2023).

Entretanto, o crescente avanço tecnológico e a demanda por aplicações de software criam um cenário desafiador em relação à disponibilidade de desenvolvedores qualificados para atender a essa demanda. Estudos como o de Sahay *et al.* (2020) ressaltam o déficit significativo de profissionais capacitados nessa área, o que acaba impactando diretamente as empresas e seus serviços. A Brasscom (2021) estima que anualmente há uma demanda de 159 mil talentos no setor, e esse número deve aumentar para impressionantes 797 mil até o ano de 2025, este cenário é rotulado pela tal, como um “apagão de profissionais de TI”. Essa escassez de desenvolvedores competentes ressalta a necessidade urgente de investimentos em capacitação e formação profissional, bem como a adoção de estratégias para atrair e reter talentos na área de desenvolvimento de software.

Nesse contexto de constante evolução, as ferramentas *low-code* têm se destacado como uma alternativa promissora para acelerar o processo de criação de aplicações (KÄSS

et al., 2023). Conforme mencionado por Vincent *et al.* (2019), essas plataformas visam lidar com a escassez de profissionais altamente qualificados, permitindo que indivíduos com pouca ou nenhuma experiência em programação possam desenvolver aplicações de forma mais ágil, intuitiva e eficiente.

Sahay *et al.* (2020) reitera que as plataformas de desenvolvimento *low-code* tem como principal objetivo lidar com a carência de desenvolvedores qualificados. As plataformas de desenvolvimento *low-code* são fornecidas através do modelo *platform as a service* (PaaS) e permitem o desenvolvimento e a implantação de aplicativos totalmente funcionais utilizando interfaces gráficas e abstrações visuais que exigem o mínimo ou nenhum código.

Por volta de 2014 o termo *low-code* surgiu na Forrest Research (RICHARDSON *et al.*, 2014) declarando que as empresas tem uma certa tendência de escolher essas ferramentas para uma entrega mais rápida, contínua e de aprendizado por meio de testes. Na maioria das vezes as plataformas *low-code* possuem ecossistemas para os aplicativos poderem ser desenvolvidos, visando a rapidez, já que tudo está pré-configurado e pré-construído, acarretando uma diminuição da configuração manual do código. Essas plataformas possuem uma interface visual que permite com que pessoas sem conhecimento técnico em desenvolvimento de *software* consigam criar e implantar aplicativos com uma maior facilidade (WASZKOWSKI, 2019), em contrapartida, as demais alternativas de desenvolvimento requer conhecimento prévio no âmbito da programação.

Outro principal objetivo das plataformas de desenvolvimento *low-code* é permitir que aplicativos sejam desenvolvidos sem a necessidade de amplos conhecimentos em programação, isento da preocupação de como o sistema deve se comportar e ser estruturado, sendo assim, facilitando a configuração, alcançando agilidade e rapidez (KÄSS *et al.*, 2023). Além disso, outro objetivo é a economia, as empresas criam soluções mais econômicas que conseguem atender os requisitos do mercado ou até mesmo para a própria empresa. Assim como também, essas aplicações podem ser desenvolvidas para diversas plataformas simultaneamente, desempenhando multifunções e com alta capacidade de gerenciamento de informações (KÄSS *et al.*, 2023).

Vincent *et al.* (2019) mostram nos seus relatórios que até 2025 aproximadamente 70% de todas as aplicações empresariais utilizarão algum tipo de *low-code*, por volta de 90% das organizações estarão usando ou considerando utilizar as LCPDs. Tudo isso se deve ao fato das vantagens que esse tipo de tecnologia traz, como Sanchis *et al.* (2019) estabelece que as principais vantagens são:

- Privacidade;

- Rapidez;
- Redução de custos;
- Redução da complexidade;
- Fácil manutenção;
- Para leigos em programação;
- Estabilização dos requisitos.

Sanchis *et al.* (2019) discorrem que as plataformas de desenvolvimento de baixo código tem seus benefícios, mas é preciso considerar os riscos de adotar esse tipo de tecnologia. O mesmo fala que os riscos são:

Escalabilidade: esse tipo de plataforma de desenvolvimento pode não se adequar a projetos críticos e de larga escala, sendo mais voltada para projetos menores.

Fragmentação: cada plataforma *low-code* pode seguir com um tipo de paradigma diferente, implicando na falta de padronização clara ou uniformidade entre as plataformas. Acarretando implicações para os desenvolvedores e as organizações que adotam essa tecnologia, podendo dificultar a interoperabilidade entre as diferentes plataformas e a portabilidade entre elas.

Desenvolvimento de sistemas puramente de software: esses desenvolvedores *low-code* tem pouco experiência em programação, mas habitualmente são especialistas em outras áreas como engenharia de *software*, *marketing*, administradores de empresas e entre outras áreas. É importante permitir que esses especialistas utilizem seu conhecimento e habilidades. Implicando que pode ser preciso fornecer ferramentas e abstrações de programação que sejam acessíveis para esses especialistas, assim permitindo que eles utilizem toda sua *expertise* eficazmente no desenvolvimento.

O TV e Abdulla (2022) enfatizaram que *no-code* é uma subseção do *low-code*, portanto este trabalho trata essas duas tecnologias como uma só. Alguns exemplos de soluções *no-code* são: AppSheet do Google e Bubble, seguidamente, exemplos de *low-code*: Salesforce e Zoho. Também mencionado que as plataformas de desenvolvimento *low-code* podem ser usadas para desenvolver aplicativos mais complexos, já as plataformas de desenvolvimento *no-code* podem ser utilizadas para criar relatórios, aplicativos de análise e rastreamento.

Atualmente, uma das principais desvantagens no desenvolvimento de *software* é a dificuldade em escolher a solução mais adequada entre as diversas alternativas disponíveis para o projeto. Então, partindo da hipótese que não haja a solução "coringa", o presente trabalho tem como cerne realizar uma análise comparativa dessas soluções. Visando avaliar e explorar suas características, funcionalidades e impacto no desenvolvimento de *software*, para auxiliar os desenvolvedores e organizações na escolha de uma solução para o projeto. A comparação irar avaliar características que poderão contribuir para com o sucesso do projeto, por exemplo, uma melhor percepção de qual ferramenta é mais adequada e quais recursos são melhores para serem utilizados, tamanho da comunidade, desempenho, curva de aprendizagem e entre outros.

O objetivo central é avaliar e explorar as características, funcionalidades e impactos no desenvolvimento de software de cada solução, com vistas a auxiliar desenvolvedores e organizações na tomada de decisão mais adequada para seus projetos.

A análise comparativa contemplará características que podem influenciar significativamente o sucesso do projeto, tais como:

Adequação da ferramenta às necessidades específicas do projeto; Recursos disponíveis e sua efetividade; Tamanho e engajamento da comunidade de usuários; Desempenho da ferramenta; Curva de aprendizado; Funcionalidades; Outros aspectos relevantes. Ao fornecer uma visão abrangente e comparativa das diferentes soluções disponíveis, este trabalho visa contribuir para a escolha mais consciente e estratégica da ferramenta ideal para cada projeto, otimizando o processo de desenvolvimento e maximizando as chances de sucesso.

1.1 Objetivos

Nesta seção, são apresentados o objetivo geral e os objetivos específicos deste trabalho, os quais orientam e fundamentam a realização da pesquisa com o intuito de contribuir para a comunidade acadêmica e de desenvolvimento.

1.1.1 Objetivo geral

O objetivo deste trabalho é realizar uma análise comparativa das ferramentas de desenvolvimento *low-code*, com ênfase em soluções de *backend*, a fim de listar a plataforma mais adequada.

1.1.2 Objetivos específicos

- Realizar uma pesquisa e seleção de ferramentas *low-code* com ênfase em soluções de backend;
- Especificar os requisitos funcionais e não funcionais da aplicação a ser implementada em diferentes plataformas *low-code*;
- Implementar um *backend* com os requisitos especificados nas plataformas *low-code* selecionadas;
- Comparar as plataformas *low-code* voltadas para o *backend*;

Os próximos capítulos estão organizados da seguinte forma: O Capítulo 2 aborda a fundamentação teórica, com termos técnicos e embasamento literário para a análise comparativa proposta; O Capítulo 3 discute trabalhos relacionados relevantes para este estudo; O Capítulo 4 descreve a metodologia e os passos seguidos; O Capítulo 5 são apresentados os resultados obtidos a partir da metodologia usada; Por último, no Capítulo 6 são apresentadas as conclusões a partir dos resultados obtidos, com as contribuições do presente trabalho e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta e explica os principais conceitos técnicos que contribuíram para o desenvolvimento deste trabalho, visando proporcionar um melhor entendimento do mesmo. Inicialmente, será introduzido o conceito de *low-code* na Seção 2.1; posteriormente apresentado API na Seção 2.2; na Seção 2.3 é apresentado o conceito de banco de dados; na Seção 2.4 é abordado a definição de especificação de *software*; por fim, o último ponto a ser apresentado são as plataformas de desenvolvimento *low-code* na Seção 2.5.

2.1 *Low-code*

O termo *low-code* surgiu supostamente em meados de 2014 com base em uma pesquisa de mercado realizada por Richardson *et al.* (2014). É um paradigma de desenvolvimento de software que requer pouca codificação, utilizando *interfaces* gráficas e design orientado a modelos (ALAMIN *et al.*, 2021). Segundo Moraes (2022), a implantação de soluções *low-code* é benéfico por várias razões, tais como a capacidade de desenvolver aplicativos, sistemas, automações, integrações e com um custo reduzido.

Segundo Gartner (2022) afirma que as organizações estão cada vez mais adotando tecnologias de desenvolvimento *low-code* para atender às demandas crescentes de velocidade de entrega de aplicativos e fluxos de trabalho de automação altamente personalizados. O que se pode perceber pelo crescimento previsto para 2023 de 19.6% com relação ao ano de 2022, conforme o Quadro 1.

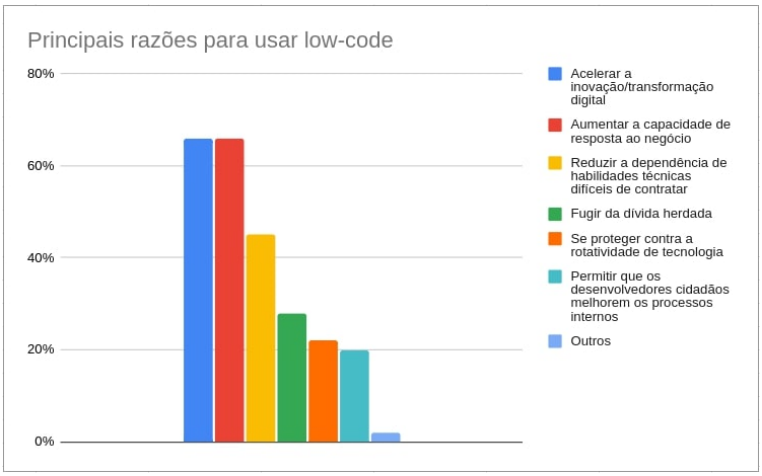
Quadro 1 – Receita de tecnologias de desenvolvimento de *low-code* (em milhões de dólares)

	2021	**2022**	**2023 (previsão)**	**2024 (previsão)**
Plataformas de aplicativos Low-Code (LCAP)	6,32	7,97	9,96	12,35
Automação de processos de negócios (BPA)	2,42	2,59	2,76	2,94
Plataformas de Desenvolvimento Multiexperiência (MDXP)	2,08	2,51	3,00	3,56
Automação Robótica de Processos (RPA)	2,35	2,89	3,40	3,88
Plataforma de integração como serviço (iPaaS)	4,68	5,67	6,67	7,84
Plataformas de automação e desenvolvimento do cidadão (CADP)	554	732	953	1,23
Outras tecnologias de desenvolvimento de baixo código (LCD)	92	109	126	146

Fonte: Adaptação e tradução livre do Gartner (2022).

Mediante uma pesquisa realizada pela Outsystems (2019), por meio de um questionário aplicado em empresas, foi possível identificar as principais razões que levam as organizações a adotarem o *low-code*. A Figura 1 apresenta as respostas mais relevantes. Essas respostas oferecem informações valiosas sobre as motivações que impulsionam a adoção dessa tecnologia no ambiente corporativo. É possível perceber que as motivações giram em torno dos objetivos que as soluções *low-code* aspiram atingir.

Figura 1 – Gráfico das razões para usar *low-code*

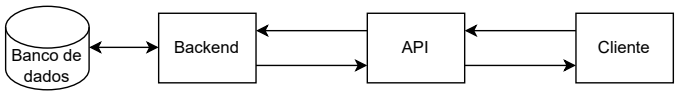


Fonte: Adaptação e tradução livre do Outsystems (2019, p. 35).

2.2 API

API (*Application Programming Interface*) é um conjunto de regras e definições que permitem que um software interaja com outro, facilitando a comunicação entre diferentes sistemas. *APIs* definem os métodos que uma aplicação pode usar para solicitar dados ou funcionalidades de outro sistema, seja ele um serviço externo ou interno. Elas abstraem a complexidade do sistema e fornecem interfaces simplificadas para acesso a suas funcionalidades (MASSE, 2011). A Figura 2 ilustra uma representação simplificada da interação entre os principais componentes de um sistema que utiliza APIs: o cliente, responsável por realizar as solicitações; a API, que atua como intermediária entre os sistemas; o *backend*, que processa as solicitações e fornece os dados ou serviços solicitados; e o banco de dados salva, edita, deleta ou fornece as informações que o *backend solicita*.

Figura 2 – Representação de uma *API*



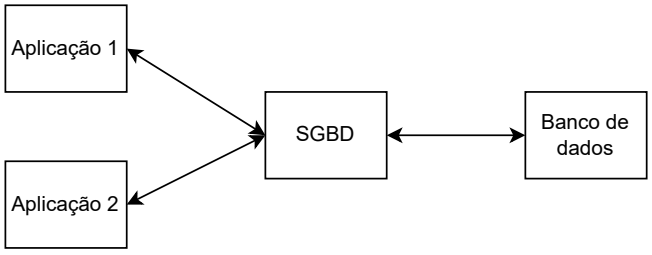
Fonte: Elaborado pelo o autor.

2.3 Banco de dados

Segundo Date (2004) um banco de dados pode ser considerado um armário eletrônico no qual é possível guardar informações, buscar, atualizar e deletar. O conceito de banco de dados está intrinsecamente ligado ao armazenamento, organização e manipulação de informações em sistemas computacionais. Um banco de dados pode ser definido como um conjunto de dados estruturados de maneira a permitir o acesso eficiente, a inserção, a atualização e a remoção de informações (DATE, 2004). Sua principal função é servir como uma ferramenta para o gerenciamento de dados, oferecendo recursos que permitem consultas e manipulações de maneira otimizada e segura (DATE, 2004).

Elmasri *et al.* (2005) refere-se acerca do SGBD (Sistema de Gerenciamento de Banco de Dados) é um sistema computadorizado voltado para gerenciar um banco de dados, no qual o usuário pode solicitar algo para o sistema. SGBD é um *software* que permite a criação, gerenciamento e manipulação de bancos de dados. Ele fornece uma interface entre os usuários e o banco de dados, facilitando a interação com os dados de forma segura, eficiente e estruturada (ELMASRI *et al.*, 2005). De acordo com Date (2004), um SGBD é essencial para a gestão de grandes volumes de dados, já que controla a maneira como os dados são armazenados, recuperados e organizados, além de fornecer mecanismos de segurança e integridade. A Figura 3 ilustra como diferentes aplicações podem se conectar ao SGBD, que por sua vez gerencia o acesso ao banco de dados, proporcionando uma interface uniforme para interação com os dados.

Figura 3 – Representação de um banco de dados e SGBD



Fonte: Elaborado pelo o autor.

2.4 Especificação de software

Conforme afirmado por Sommerville (2011), a especificação de software envolve compreender e definir os serviços solicitados pelo sistema, além de identificar restrições relacionadas à operação e ao desenvolvimento do sistema. A engenharia de requisitos é particularmente crucial no desenvolvimento de software, uma vez que erros nessa fase podem resultar em problemas no projeto, especialmente na fase da implementação (VAZQUEZ; SIMÕES, 2016).

Os requisitos de software são a expressão das necessidades dos usuários ou clientes, que devem ser consideradas e incorporadas ao sistema em desenvolvimento. Sommerville (2011) compreende que os requisitos são:

- Requisitos funcionais: são responsáveis por descrever as funcionalidades e serviços que o sistema a ser desenvolvido deve oferecer, ou seja, eles definem as ações que o sistema deve conseguir realizar;
- Requisitos não funcionais: são as restrições relacionadas sobre como os requisitos funcionais deverão ser implementados.

Durante esta etapa, ocorre a transposição das informações obtidas durante a atividade de análise para um documento que estabelece um conjunto de requisitos. Esse documento pode abranger dois tipos de requisitos, os requisitos do usuário e os requisitos do sistema. Ademais, é essencial realizar uma validação dos requisitos. Ao decorrer do processo, eventuais erros no documento de requisitos podem ser identificados e faz-se necessário corrigi-los.

2.5 Plataformas de desenvolvimento *low-code*

As plataformas de desenvolvimento *low-code* são utilizadas para desenvolver aplicações sem conhecimento em programação, tendo como público alvo os não programadores, se utilizando de um estilo de programação com o mínimo de codificação para criar *interfaces* gráficas, com componentes pré-construídos (KHORRAM *et al.*, 2020).

2.5.1 Directus

Directus é uma plataforma open source de gerenciamento de conteúdo (CMS) e *headless*, projetada para facilitar o gerenciamento de dados estruturados em bancos de dados SQL, sem impor um esquema fixo. Sendo *headless*, o Directus não gera interfaces para os

usuários finais, atuando exclusivamente como um backend, o que permite aos desenvolvedores criar suas próprias *interfaces front-end* ou integrar a API com outros sistemas.

A plataforma permite que os usuários escolham entre uma variedade de bancos de dados relacionais, como PostgreSQL, MySQL, SQLite, OracleDB, Microsoft SQL Server e CockroachDB, garantindo flexibilidade e adaptabilidade às diferentes necessidades dos projetos. O Directus oferece uma API gerada automaticamente, com suporte a REST e GraphQL, permitindo acesso e manipulação de dados de forma simples e eficiente.

O Directus se destaca por sua interface de administração amigável e personalizável, facilitando a gestão de dados, coleções e permissões de usuários sem a necessidade de codificação. Além de ser auto-hospedado, ele oferece planos pagos para hospedagem gerenciada, proporcionando uma solução pronta para uso sem a preocupação com infraestrutura.

2.5.2 Strapi

Strapi é um sistema de gerenciamento de conteúdo (CMS) e *headless*. A plataforma é *open-source* e permite que o usuário hospede a aplicação em seu próprio servidor, oferecendo flexibilidade e controle sobre o ambiente. Além disso, o Strapi oferece planos pagos que incluem serviços como hospedagem gerenciada, eliminando a necessidade de o usuário se preocupar com a infraestrutura.

O Strapi é compatível com diversos bancos de dados, incluindo PostgreSQL, MySQL, MariaDB e SQLite, todos seguindo as estruturas relacionais do SQL. Uma API é gerada automaticamente quando as coleções são definidas, possibilitando a criação de endpoints REST ou GraphQL sem esforço adicional. A plataforma também é altamente personalizável, oferecendo uma ampla gama de tipos de dados e opções de customização para se adequar às necessidades de cada projeto.

2.5.3 Parse

Parse é uma plataforma open source para desenvolvimento de backend que permite gerenciar dados, autenticação de usuários, notificações *push*, entre outros recursos, sem a necessidade de construir um servidor do zero. Originalmente criado como um serviço de backend hospedado pela Parse, que foi adquirida pelo Facebook, o Parse se tornou uma solução auto-hospedada após o serviço da Parse ser descontinuado em 2017.

A plataforma utiliza um banco de dados MongoDB ou PostgreSQL, oferecendo

flexibilidade na escolha da estrutura de dados. Ela gera automaticamente uma API REST para os desenvolvedores poderem interagir com os dados e realizar operações CRUD sem esforço adicional. Além disso, Parse suporta GraphQL e WebSockets de forma não nativa, proporcionando uma comunicação em tempo real.

Parse também oferece diversos serviços integrados, como gerenciamento de arquivos, controle de permissões e funções em nuvem (Cloud Code), que permitem a execução de lógica personalizada diretamente no backend. A plataforma pode ser facilmente escalada em ambientes como Heroku, AWS, Google Cloud ou qualquer infraestrutura compatível com Node.js.

2.5.4 Firebase

Firebase é uma plataforma de desenvolvimento de aplicativos criada pelo Google, que oferece uma série de ferramentas e serviços backend como banco de dados, autenticação, armazenamento e notificações *push*, facilitando o desenvolvimento de apps móveis e web.

O Firebase oferece dois principais bancos de dados: o Realtime Database, que sincroniza dados entre os clientes em tempo real, e o Firestore (Cloud Firestore), que oferece um modelo de dados mais flexível, com suporte a consultas mais avançadas e maior escalabilidade. Ambos são bancos NoSQL, projetados para lidar com dados dinâmicos e interações em tempo real.

Além do banco de dados, o Firebase possui um sistema de autenticação integrado com suporte a provedores como Google, Facebook, GitHub, Apple, além de autenticação por e-mail, número de telefone e métodos personalizados. Ele também oferece Firebase Storage para o armazenamento de arquivos, Firebase Hosting para hospedar sites estáticos, e o Firebase Cloud Messaging (FCM) para envio de notificações *push*.

Outros recursos incluem Firebase Functions, que permite executar código *backend serverless*, e Firebase Analytics, que fornece informações detalhadas sobre o comportamento dos usuários.

2.5.5 Supabase

Supabase é uma plataforma open source que fornece uma alternativa ao Firebase, permitindo a criação de *backends* completos com funcionalidades como banco de dados, autenticação e armazenamento em tempo real. Ele se baseia no PostgreSQL como seu banco de dados principal, oferecendo suporte nativo a funcionalidades como *triggers*, *views* e *stored procedures*,

além de uma API RESTful gerada automaticamente a partir de qualquer tabela ou *schema* no banco de dados.

A plataforma oferece um sistema de autenticação completo com suporte a provedores como Google, GitHub, Twitter e Apple, além de autenticação por e-mail e senha. O Supabase também disponibiliza um sistema de permissões detalhado, baseado em *Row-Level Security* (RLS), permitindo que os desenvolvedores definam regras diretamente no banco de dados para controlar o acesso a dados sensíveis.

Além disso, o Supabase tem suporte a WebSockets para fornecer sincronização em tempo real, o que é útil para aplicações colaborativas e atualizações instantâneas. Ele também oferece armazenamento de arquivos com um sistema de permissões integrado, e a plataforma pode ser facilmente escalada, sendo compatível com diversos serviços de nuvem como AWS e DigitalOcean. O Supabase oferece uma interface amigável e pode ser hospedado pelo próprio usuário ou utilizado via os planos gerenciados fornecidos pela empresa.

3 TRABALHOS RELACIONADOS

Este capítulo, apresenta os estudos anteriores relevantes para situar o contexto do presente trabalho e identificar o domínio ao qual ele pertence.

3.1 *Comparison between Headless CMS and Backend-as-a-Service Products for E-Suripreneur Backend*

Em um estudo conduzido por Sobri *et al.* (2022), são comparadas três soluções *low-code*: dois *headless content management system* (CMS) e uma plataforma de *backend-as-a-service* (BaaS). As opções de *headless* CMS avaliadas são o Strapi e o Directus, enquanto o Supabase é a opção de BaaS. O objetivo do estudo é identificar a solução que melhor atenda aos requisitos do E-Suripreneur, que incluem recursos como CRUD, autenticação, autorização de usuário, migração de banco de dados, baixo custo, capacidade de auto-hospedagem, personalização com código customizado e capacidade de enviar dados para o cliente por meio de uma API REST ou outro método amplamente suportado.

Com base nos requisitos e análises, o Sobri *et al.* (2022) chegou na resposta que o Directus é a solução que mais se adequa aos requisitos do E-Suripreneur, baseado nos resultados. Em trabalhos futuros o autor planeja investigar opções para terceirizar serviços do Directus.

Diferentemente do estudo de Sobri *et al.* (2022) este trabalho vai além da comparação das ferramentas com base em requisitos. Neste trabalho, são analisados critérios qualitativos e quantitativos. Aliás, será desenvolvida uma aplicação específica para testar o desempenho de cada uma delas.

3.2 *Supporting the understanding and comparison of low-code development platforms*

No estudo conduzido por Sahay *et al.* (2020), foi realizado uma análise comparativa de oito *low-code development platforms* (LCDPs) líderes no mercado, visando extrair os recursos e funcionalidades mais relevantes e organizá-las de forma sistemática, com base em uma taxonomia definida e organizada.

O objetivo é fornecer elementos que auxiliem os tomadores de decisões e adotantes a fazer escolhas educadas e considerações ao selecionar uma plataforma adequada às suas necessidades. Além disso, o artigo apresenta uma experiência prática de adoção dessas plataformas para o desenvolvimento de um aplicativo simples, mostrando como as LCDPs podem ser usadas

por pessoas sem conhecimento em desenvolvimento de *software*.

O artigo identifica recursos e funcionalidades relevantes que caracterizam diferentes LCDPs, como suporte a múltiplas plataformas, integração com outras ferramentas, facilidade de uso e personalização. As funcionalidades foram extraídas de oito plataformas, sendo elas: OutSystem, Mendix, Zoho Creator, Microsoft PowerApps, Google App Maker, Kissflow, Salesforce App Cloud e o Appian.

A comparação das plataformas é feita com base nos recursos e funcionalidades identificados anteriormente. O artigo também apresenta um resumo das limitações e desafios encontrados durante o desenvolvimento do aplicativo de benchmark. No entanto, o artigo não indica qual é a melhor plataforma de desenvolvimento de *low-code* (LCDP), deixando a decisão final para o leitor.

3.3 *Comparative Study on Various Low Code Business Process Management Platforms*

O artigo de Krishnaraj *et al.* (2022) aborda uma comparação entre diferentes plataformas de desenvolvimento de processos de negócios de *low-code*. Essas plataformas oferecem soluções modernas para os métodos de codificação, reduzindo preocupações com integração e simplificando os processos de desenvolvimento Krishnaraj *et al.* (2022). O estudo analisa três plataformas populares: Appian, SalesForce Lightning e Caspio.

A plataforma Appian se destaca por sua capacidade de fornecer um ambiente personalizado e flexível para relatórios, permitindo que os clientes entreguem e modifiquem rapidamente relatórios. Além disso, o Appian oferece recursos de análise em tempo real, painéis de controle personalizáveis e monitoramento de atividades empresariais.

O SalesForce Lightning é mencionado como uma plataforma que oferece uma ampla gama de recursos, incluindo automação de processos de negócios, gerenciamento de relacionamento com o cliente (CRM) e análise de dados. Ele também permite a criação de aplicativos personalizados sem a necessidade de codificação.

A plataforma Caspio se destaca por sua capacidade de criar aplicativos de banco de dados online escaláveis e seguros sem a necessidade de codificação. Ela oferece recursos de criação de formulários, relatórios e painéis personalizados, além de integração com outras ferramentas e serviços.

No geral, o artigo fornece uma visão geral das características e benefícios dessas plataformas de *low-code*, destacando suas capacidades de relatórios, análise em tempo real,

automação de processos e criação de aplicativos personalizados sem a necessidade de codificação.

3.4 Uma análise comparativa entre *frameworks* de persistência de dados em *Javascript*

No trabalho de Rocha (2022) é realizado uma comparação entre seis *frameworks* de persistência de dados em *Javascript*, os resultados servem como um catálogo para os desenvolvedores e instituições de tecnologia. Os critérios usados para a comparação são de dois tipos: quantitativos e qualitativos.

Primeiramente, os critérios quantitativos são: desempenho em tempo de execução, simplicidade de treino, custo de manutenção, banco de dados suportados, tamanho da comunidade, modelos de dados aceitos, padrões de persistência implementados, início do projeto vs última atualização. Essa categoria serve qualificar um *framework*, podendo afetar a decisão de um desenvolvedor ou de uma empresa, esses fatores podem ser cruciais para o desenvolvimento correto e bem-sucedido de um projeto de banco de dados, incluindo a complexidade dos passos necessários para realizar uma ação, o desempenho em termos de tempo de resposta de uma determinada ferramenta em casos específicos, entre outros aspectos relevantes.

Segundamente, os critérios qualitativos: curva de aprendizagem, simplicidade, facilidade de escrita, estratégia no carregamento (loading) de dados, documentação. Os critérios mencionados são cuidadosamente elaborados para influenciar a decisão de escolha de um framework e assegurar a qualidade e o sucesso do projeto de banco de dados. No entanto, é importante ressaltar que esses critérios não podem ser apenas quantificados e generalizados, uma vez que variam entre os desenvolvedores e os diferentes projetos.

Rocha (2022) conduziu uma comparação entre ferramentas de persistência por meio da criação de seis projetos de banco de dados, visando avaliar os atributos de cada *frameworks*. Para todos os projetos, o trabalho adotou a arquitetura de *application programming interface* (API), considerada uma das três categorias de API mais populares, conforme o autor.

Após a definição dos atributos e sua agrupação em critérios mais abrangentes, que orientam a análise comparativa e inseridos no catálogo, juntamente com a implementação de todas as APIs, os *frameworks* são avaliados de forma independente em relação a cada atributo. Ao finalizar cada teste em um determinado atributo, o resultado da avaliação do *framework* é adicionado à linha correspondente da ferramenta e à coluna correspondente ao atributo recém-testado.

Visando realizar uma comparação entre soluções *low-code* para *backend*, o presente

trabalho utiliza os atributos e os agrupa em critérios quantitativos e qualitativos, conforme efetuado no trabalho do Rocha (2022), também será desenvolvido uma API para cada uma das plataformas, com intenção de melhor avaliar o desempenho de cada atributo e os colocar em uma tabela comparativa.

3.5 Análise Comparativa

O presente trabalho realiza uma pesquisa técnica com a intenção de identificar as funcionalidades das plataformas de desenvolvimento *low-code* e as comparar, assim como realizado nos trabalhos de Sahay *et al.* (2020) e Rocha (2022), como abordado no Quadro 2, já o do Krishnaraj *et al.* (2022) busca apenas comparar as ferramentas conforme os requisitos do E-Suripreneur. Distintivamente da obra de Sahay *et al.* (2020), as plataformas analisadas não estão nos relatórios do Gartner, mas são populares, a comunidade cada vez mais adota essas tecnologias e são apenas ferramentas para *backend*, diferentemente do Krishnaraj *et al.* (2022) sendo soluções de *low-code* para processos de negócio. Quando se trata do trabalho do Rocha (2022), que analisa apenas *frameworks javascript*, porém o trabalho deste sustenta os procedimentos metodológicos desta obra.

Quadro 2 – Quadro comparativo entre os trabalhos relacionados e proposta de trabalho.

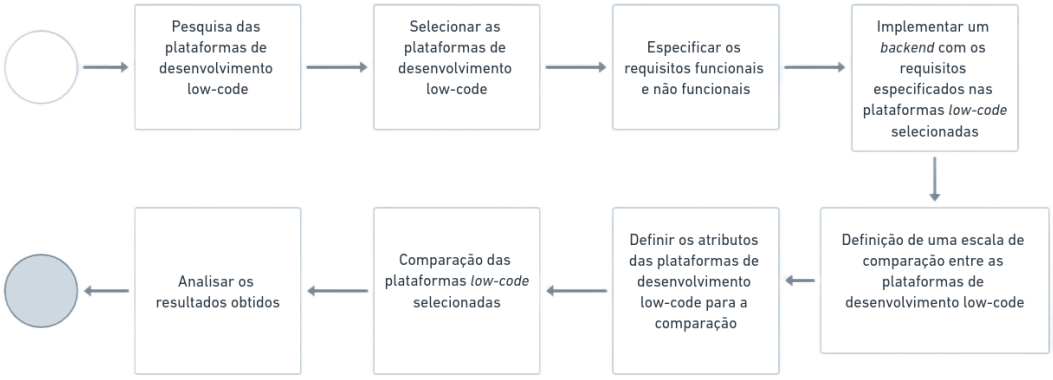
Trabalhos	**Definição da lista de plataformas comparadas**	**Plataformas low-code para backend comparadas**	**Tipos dos requisitos especificados**	**Elaboração das características**	**Quantas plataformas low-code foram comparadas a partir de uma implementação do backend contendo os requisitos especificados**
Sobri *et al.* (2022)	Definido pelo autor	Directus, Strapi e Supabase	Funcionais e não funcionais	A partir dos requisitos e ao decorrer do uso das plataformas	-
Sahay *et al.* (2020)	Baseado no relatório do Gartner e Forrester	-	-	Definido pelo autor ao utilizar as plataformas	-
Krishnaraj *et al.* (2022)	Definido pelo autor	-	-	-	-
Rocha (2022)	Definido pelo autor	-	Funcionais	Definido pelo autor ao utilizar as plataformas	-
Proposta de trabalho	Pesquisa técnica	Directus, Strapi, Parse, Supase e Firebase	Funcionais e não funcionais	A partir da utilização das plataformas e critérios propostos por Rocha (2022)	5

Fonte: Elaborado pelo autor.

4 PROCEDIMENTOS METODOLÓGICOS

Neste capitulo são descritos os passos necessários para a realização do presente trabalho. A Figura 4 apresenta os passos essenciais para a execução do trabalho. Passo I, pesquisa das plataformas de desenvolvimento *low-code*; Passo II, selecionar as plataformas de desenvolvimento *low-code*; Passo III, definir os requisitos funcionais e não funcionais; Passo IV, implementar um *backend* com os requisitos especificados nas plataformas *low-code* selecionadas; Passo V, definição de uma escala de comparação entre as plataformas de desenvolvimento *low-code*; Passo VI, definir os atributos das plataformas de desenvolvimento *low-code* para a comparação; Passo VII, comparação das plataformas *low-code* selecionadas; Passo VIII, analisar os resultados obtidos.

Figura 4 – Fluxo das atividades dos procedimentos metodológicos



Fonte: Elaborado pelo autor.

4.1 Pesquisa das plataformas de desenvolvimento *low-code*

Nesta seção, é apresentado o processo de pesquisa das plataformas de desenvolvimento *low-code*. Essa pesquisa envolve a busca em diversos recursos, como sites, fóruns, blogs e comunidades especializadas. Ao concluir essa pesquisa, são obtidos os resultados que revelam as tecnologias *low-code* em ascensão para o desenvolvimento backend.

4.1.1 Pesquisa técnica

A pesquisa técnica tem o objetivo de buscar as soluções *low-code* para *backend* mais populares ou as mais mencionadas nas pesquisas no buscador Google. As *strings* de busca utilizadas foram: ("plataformas de desenvolvimento low-code backend") AND ("low-code

backend") AND ("plataformas de desenvolvimento no-code para backend").

4.2 Seleção das plataformas de desenvolvimento *low-code*

Este passo consiste na seleção de uma plataforma de desenvolvimento *low-code* a partir da pesquisa do tópico 4.1.1. São obtidos os dados, é realizado uma análise dos resultados e selecionadas as plataformas que mais se destacaram na pesquisa. A análise consiste em identificar as plataformas que mais são mencionadas, a partir disso, é montado um ranking e as mais cotadas são escolhidas, a partir dos seguintes critérios: tipo de aplicação, comunidade, *vendor lock-in* e *on-premise*.

4.3 Especificação dos requisitos funcionais e não funcionais

Nesta etapa, são especificados os requisitos do sistema de forma detalhada, definindo os objetivos e funções, com clareza, que o software deve executar. Essa especificação abrange tanto os requisitos funcionais, que descrevem as principais funcionalidades e comportamentos do sistema, quanto os requisitos não funcionais, que estabelecem as restrições e características desejadas para o software.

4.3.1 Documentação dos requisitos

Nesse passo, os requisitos funcionais e não funcionais identificados são documentados de forma clara e organizada. Os requisitos estão organizados em uma tabela, para a melhor visualização, entendimento e acompanhamento dos mesmos.

4.3.2 Utilização de linguagem

A linguagem utilizada na especificação dos requisitos funcionais se dá de forma clara, evitando ambiguidades e redundâncias. É importante evitar o uso de termos técnicos complexos ou jargões desnecessários, garantindo que os requisitos estão compreensíveis. A clareza na linguagem facilita a comunicação e o entendimento das necessidades e expectativas em relação ao software a ser desenvolvido. Além disso, uma linguagem acessível e concisa contribui para uma documentação mais eficiente e eficaz, permitindo que os requisitos sejam facilmente interpretados e implementados conforme as necessidades do sistema.

A utilização da linguagem técnica nos requisitos não funcionais é apropriada pelo

fato de que não se deve ter ambiguidade na definição desses requisitos. A linguagem técnica permite uma especificação precisa e detalhada dos aspectos não funcionais do sistema, como desempenho, segurança, confiabilidade, usabilidade, entre outros. Isso garante que as metas e limites dos requisitos não funcionais sejam claramente estabelecidos, facilitando a compreensão e o atendimento desses requisitos durante o processo de desenvolvimento do software.

4.4 Implementação de um *backend* com os requisitos especificados nas plataformas *low-code* selecionadas

Nesta etapa do processo, é desenvolvida uma aplicação em cada uma das plataformas de desenvolvimento *low-code* selecionadas na Seção 4.2, com base nos requisitos especificados na Seção 4.3. Em seguida, realiza-se uma comparação das soluções na Seção 4.7, visando obter uma análise mais precisa e abrangente das mesmas. Durante essa análise, são considerados critérios qualitativos, quantitativos, funcionalidades e entre outros, a fim de identificar a plataforma que mais se destaca.

4.5 Definição de uma escala de comparação entre as plataformas de desenvolvimento *low-code*

No trabalho de Rocha (2022), uma escala é utilizada para uma avaliação mais precisa de alguns dos atributos qualitativos e quantitativos. Neste trabalho, uma abordagem semelhante será adotada, porém, será utilizada a escala Leikert, diferentemente do trabalho de Rocha (2022), que propôs uma escala própria. Isso permite uma comparação mais consistente e objetiva dos atributos analisados.

A escala de valores utilizada é a Leikert variando de 1 a 5, representando diferentes níveis de satisfação para um determinado cenário de uso da plataforma. O valor 1 indica a pior atribuição, indicando que o critério avaliado não corresponde tão bem ao cenário em questão. Já o valor 5 indica a atribuição mais alta, sugerindo que o critério corresponde muito bem ao cenário específico (JÚNIOR; COSTA, 2014).

Os valores da escala variam da seguinte forma:

1. Ruim.
2. Razoável;
3. Bom;

4. Muito bom;
5. Excelente;

Essa escala permite uma avaliação objetiva das plataformas, fornecendo uma medida clara sobre a adequação e desempenho de cada uma delas em relação ao cenário em questão (FRANKENTHAL, 2017).

4.6 Definição dos atributos das plataformas de desenvolvimento *low-code* para a comparação

Assim como Rocha (2022) se utiliza de atributos e os agrupa em dois tipos de critérios, o mesmo será feito nesta análise. Os atributos estão divididos em dois grupos de critérios: quantitativos e qualitativos.

Nesta seção, é apresentado os atributos que guiam a avaliação das plataformas de desenvolvimento *low-code*, classificados em critérios quantitativos e qualitativos. Em seguida, é detalhado os critérios e os atributos. Durante as comparações, em alguns atributos é considerado os passos necessários para atingir um cenário específico, ou seja, as ações que os desenvolvedores devem executar para alcançar o objetivo proposto. Ao contabilizar esses passos, é determinado o número total de ações requeridas para alcançar o resultado desejado.

4.6.1 Critérios quantitativos

Nesta categoria estão dispostos os atributos com o poder de influenciar a decisão de uso de uma solução *low-code*. A quantidade de passos para realizar determinada ação é considerado, bem como o desempenho em termos de tempo de resposta, entre outros. Essas considerações são essenciais para garantir a escolha correta e eficiente da plataforma de desenvolvimento *low-code*.

Desempenho em tempo de execução

Neste critério, é realizada uma avaliação do tempo de resposta, avaliado em milissegundos, das operações típicas de CRUD, como criar, atualizar, ler e deletar. Para cada plataforma de desenvolvimento *low-code* é ser realizado quatro chamadas *Hypertext Transfer Protocol* (HTTP) e os resultados obtidos são colocados em uma tabela.

Modelos de dados aceitos

A avaliação deste critério é simples, envolvendo a análise dos modelos de dados aceitos por cada plataforma de desenvolvimento *low-code*. Durante a avaliação, são examinados os tipos de modelos de dados suportados. Essa informação é utilizado para identificar as plataformas que melhor atendem aos requisitos específicos relacionados ao armazenamento e gerenciamento de dados.

Banco de dados suportados

A avaliação é baseada na quantidade de bancos de dados suportados pelas plataformas de desenvolvimento *low-code*. É importante destacar que existem soluções que aceitam apenas um banco de dados, enquanto outras oferecem suporte a múltiplos bancos de dados. Esses aspectos são avaliados para identificar as plataformas que melhor atendem às necessidades específicas relacionadas à gestão e armazenamento de dados.

Tamanho da comunidade

A verificação desse aspecto baseia-se no tamanho das comunidades das soluções *low-code* presentes nas plataformas *Github* e *Stack Overflow*, assim como nas comunidades específicas criadas por cada solução.

Funcionalidades extras

A avaliação desse critério baseia-se nos seguintes critérios: tipos de APIs, autenticação, *interface* de usuário, modularidade e extensibilidade e suporte a tempo real.

4.6.2 Critérios qualitativos

Os critérios estabelecidos na subseção referem-se à avaliação da qualidade e ao potencial de sucesso do projeto, desempenhando um papel fundamental na escolha da plataforma a ser adotada. Esses critérios abrangem diversos aspectos, como a curva de aprendizado, a simplicidade de configuração e recursos de documentação. Essas considerações são essenciais para garantir que a escolha da plataforma esteja alinhada aos objetivos e às necessidades do projeto, visando o êxito e a qualidade na entrega do produto final.

Curva de aprendizado

Essa propriedade é avaliada pela quantidade de passos necessários para a conclusão de um determinado cenário. Os passos contabilizados são considerados em um cenário ideal, não havendo erros de configuração ou erros no decorrer do processo. Os cenários determinados são: quantidade de passos necessários para executar as operações CRUD nas plataformas.

Simplicidade de configuração

Semelhantemente a propriedade anterior, também é estipulado cenários ideais para a realização das atividades e a quantidade de passos é contabilizada. Os cenários são:

1. Configuração inicial;
2. Quantidade de passos para criar uma entidade;
3. Quantidade de passos para criar uma relação entre as entidades já criadas.

Documentação

A documentação é avaliada levando em consideração a qualidade dos recursos e informações disponíveis, o que desempenha um papel fundamental na compreensão e utilização da ferramenta. A documentação abrange informações, guias e exemplos que auxiliam os usuários no processo de desenvolvimento. A qualidade e abrangência desses recursos são analisadas para garantir que os desenvolvedores tenham acesso a informações claras e abrangentes, facilitando assim o uso da ferramenta.

Os recursos e informações avaliados são:

- Criação das tabelas;
- Acesso aos dados das tabelas;
- Autenticação;
- Qualidade dos exemplos e guias.

4.7 Comparação das plataformas de desenvolvimento *low-code* selecionadas

Nesta etapa, para realizar a comparação, uma aplicação é desenvolvida para cada uma das plataformas selecionadas na Seção 4.2, conforme mencionado na Seção 4.4. As

aplicações são construídas com base nos requisitos especificados na Seção 4.3, e as plataformas são comparadas com base nos atributos definidos na Seção 4.6. Essa abordagem permite uma análise detalhada e imparcial das plataformas, levando em consideração sua capacidade de atender aos requisitos e os critérios estabelecidos.

4.8 Análise dos resultados obtidos

Após a realização da comparação, os resultados obtidos são discutidos, analisados e, por fim, são relatadas as lições aprendidas em relação às comparações realizadas. Essa etapa é crucial para avaliar e compreender os pontos fortes e fracos de cada plataforma, bem como identificar *insights* relevantes que possam contribuir para futuras decisões e aprimoramentos. As conclusões obtidas são documentadas de forma a fornecer um panorama abrangente e embasado para auxiliar na tomada de decisão.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os resultados obtidos a partir da metodologia adotada no presente trabalho.

5.1 Levantamento das plataformas *low-code*

A pesquisa é realizada no mecanismo de busca Google a partir das *strings* de busca mencionadas na Seção 4.1.1. Os resultados são obtidos com base na frequência de menções de cada plataforma nos resultados. O Quadro 3 foi organizado com os valores obtidos:

Quadro 3 – Quantidade de citações das plataformas nos resultados da pesquisa do Google

Plataformas *low-code/no-code*	Quantidade de citações
Airtable	2
AWS *Amplify*	1
Backendless	6
Back4app	6
Bubble	3
Canonic	1
Direcuts	3
Firebase	3
Heroku	2
Parse	3
Strapi	4
Supabase	5
Xano	7
Zapier	2

Fonte: Elaborado pelo autor.

O Quadro 3 apresenta a quantidade de citações de cada plataforma nos resultados da pesquisa. Esses dados são usados para analisar a popularidade e o reconhecimento das plataformas com base em sua frequência de menção nos resultados do Google. A busca obteve 51.700 respostas no mecanismo de busca, pela abundância respostas, foi considerado até a 2ª página do mesmo.

5.2 Seleção das plataformas *low-code*

A seleção das plataformas *low-code* para o desenvolvimento de sistemas baseia-se em critérios como o tipo de aplicação, ausência de *vendor lock-in*, capacidade *on-premise* e popularidade nas comunidades do Stack Overflow e GitHub. A análise do *vendor lock-in* garante

que a plataforma escolhida não restrinja o usuário a um único fornecedor, proporcionando, assim, maior flexibilidade e potencial para escalabilidade futura. A possibilidade de hospedar a aplicação em servidores diferentes (*on-premise*) é igualmente relevante, por assegurar maior independência e controle sobre a infraestrutura utilizada.

Além disso, a popularidade nas comunidades do Stack Overflow e GitHub é considerada um indicador de solidez e suporte oferecido pela comunidade, demonstrando que a plataforma é amplamente adotada, capaz de facilitar a resolução de problemas e a integração de novas funcionalidades. A popularidade no Stack Overflow reflete o número de perguntas e discussões relacionadas à plataforma, o que pode indicar seu nível de uso e o interesse contínuo da comunidade em resolver questões associadas a ela. Quanto maior o número de perguntas no Stack Overflow, maior a probabilidade de haver uma ampla gama de recursos e soluções disponíveis, facilitando o desenvolvimento e a resolução de problemas.

No GitHub, o número de seguidores serve como um sinal de aceitação e do desenvolvimento ativo das plataformas. Uma quantidade maior de seguidores demonstra que a plataforma possui uma base de usuários ativa e engajada, reforçando sua confiabilidade.

As plataformas selecionadas que se destacam nesse contexto são: Directus, Strapi, Parse, Supabase e Firebase, conforme apresentado no Quadro 4. O Firebase, em particular, é escolhido por se destacar como uma das maiores em seu segmento, sendo amplamente reconhecido por sua popularidade e aceitação no mercado.

Quadro 4 – Seleção das plataformas, dados obtidos 19/07/2024

Plataformas low-code/no-code	**Tipo de aplicação**	**Stack Overflow**	**Github**	**Vendor lock-in**	**On-premise**
Airtable	*Fullstack*	287	224	Sim	Não
AWS Amplify	*Fullstack*	4970	636	Sim	Não
Backendless	*Backend*	254	32	Sim	Sim, versão paga
Back4app	*Backend*	357	113	Não	Sim
Bubble	*Fullstack*	-	-	-	-
Canonic	*Fullstack*	-	-	-	-
Directus	*Backend*	373	645	Não	Sim
Firebase	*Backend*	**143249**	3200	Sim	Não
Heroku	Platforma de hospedagem de aplicativos	-	-	-	-
Parse	*Backend*	1846	457	Não	Sim
Strapi	*Backend*	2855	1600	Não	Sim
Supabase	*Backend*	1715	**3500**	Não	Sim
Xano	*Backend*	-	-	Sim	Não
Zapier	Plataforma de automação de tarefas	-	-	-	-

Fonte: Elaborado pelo autor.

5.3 Especificação dos requisitos funcionais e não funcionais

Após concluir a seleção das plataformas de desenvolvimento *low-code*, os requisitos foram especificados. Esses requisitos definem o que o sistema deve fazer e como deve se comportar em todas as plataformas. A seguir, são apresentados os requisitos especificados:

Requisitos Funcionais:

1. O sistema deve permitir que os usuários se cadastrem e façam login.
2. O sistema deve permitir que os usuários criem, editem e excluam tarefas.
3. Cada tarefa deve ter um título, descrição, data de criação, data de atualização e status (pendente, em andamento, concluída).

Requisitos Não Funcionais:

1. O sistema deve garantir segurança de autenticação com o uso de *tokens* JWT.
2. As operações de CRUD de tarefas devem ser rápidas, com um tempo de resposta inferior a 1000ms.
3. O sistema deve possuir um banco de dados para o armazenamento das informações.

Estes são os requisitos especificados para o sistema, levando em consideração as plataformas de desenvolvimento *low-code* selecionadas para ser possível testá-las da melhor maneira.

5.4 Definição de uma escala de comparação entre as plataformas de desenvolvimento *low-code*

Alguns critérios, especialmente os relacionados à qualidade, exigem o uso de uma escala para uma análise mais precisa de atributos, que podem variar conforme cada pessoa utiliza uma função específica nas plataformas. Para indicar de maneira mais eficaz a qualidade do atributo, utiliza-se uma escala, como mencionado na Subseção 4.5.

5.5 Definição dos atributos para a comparação entre as plataformas *low-code*

Em seguida, os atributos para a comparação foram definidos com base nos requisitos especificados. Os atributos foram definidos em duas categorias como na Seção 4.6, sendo elas: qualitativas e quantitativas (VINCENT *et al.*, 2019).

Critérios qualitativos:

- Curva de aprendizado: define o nível de aprendizado da plataforma;
- Simplicidade de configuração: refere-se ao nível de facilidade de configurar a plataforma;
- Documentação: referente a qualidade da documentação da plataforma.

5.5.1 Critérios qualitativos

Nesta seção são descritos todos os atributos qualitativos escolhidos para realizar os testes em cada ferramenta de persistência de dados, sendo agrupados em um dos dois grupos de critérios maiores, o de critério qualitativo. Os atributos agrupados neste critério, são os atributos que mesmo que testes sejam realizados para analisar sua qualidade em certos cenários, não podem ser generalizados entre projetos e desenvolvedores, visto que cada um tem sua peculiaridade.

Simplicidade de configuração

Este atributo não pode ser padronizado, pois a facilidade ou dificuldade de configuração varia conforme as definições de arquitetura de software, que são externas ao desenvolvedor. Para tornar a análise mais detalhada, o atributo é subdividido em três subitens: configuração inicial, configuração das tabelas e definição de relacionamentos. Esses subitens auxiliam na obtenção de resultados mais descritivos e alcançáveis dentro deste contexto.

O Quadro 5 apresenta os dados resultantes dessa divisão e análise, indicando a quantidade de passos necessários para realizar cada uma dessas configurações nas diferentes plataformas. Assim, valores menores indicam um processo mais simplificado e ágil para o desenvolvedor.

Quadro 5 – Quantidade de passos para configuração das plataformas

Plataformas	**Configuração inicial**	**Configuração das tabelas**	**Definição de relacionamentos**
Directus	2	2	2
Strapi	2	2	1
Parse	3	3	2
Supabase	1	3	3
Firebase	**1**	**1**	**1**

Fonte: Elaborado pelo o autor.

Curva de aprendizado

Esse atributo pode variar muito de acordo com cada pessoa, influenciado tanto pela experiência prévia em programação quanto pela capacidade de assimilar novos conceitos. De acordo com 4.6.2 será considerado a quantidade de passos para montar os seguintes cenários:

- Criar as tabelas de *users* e *tasks*;
- Montar os *endpoints* das duas tabelas acima para as operações CRUD.

Quadro 6 – Quantidade de Passos para Criar Tabelas e Endpoints

Plataformas	**Passos para a criação das tabelas**	**Passos para montar os *endpoints***
Directus	1	1
Strapi	1	1
Parse	2	1
Supabase	1	1
Firebase	1	1

Fonte: Elaborado pelo o autor.

A partir do Quadro 6, é possível observar que a curva de aprendizado para a criação de tabelas e montagem dos endpoints nas diferentes plataformas apresenta variações. Para as tarefas de criação das tabelas de *users* e *tasks*, as plataformas Directus, Strapi, Supabase e Firebase necessitam de apenas um passo, enquanto o Parse requer dois passos. Já para a montagem dos endpoints CRUD (*Create*, *Read*, *Update*, *Delete*), todas as plataformas analisadas (Directus, Strapi, Parse, Supabase e Firebase) precisam de apenas um passo.

Esses dados indicam que, em termos de simplicidade e quantidade de passos para as atividades propostas, as plataformas Directus, Strapi, Supabase e Firebase oferecem uma curva de aprendizado mais uniforme e simplificada, enquanto o Parse exige um passo adicional para a criação das tabelas, o que pode influenciar a escolha da ferramenta conforme o nível de experiência e familiaridade do desenvolvedor.

Documentação

A qualidade das documentações das ferramentas foi avaliada com base nos seguintes critérios:

- Criação das tabelas;
- Acesso aos dados das tabelas;

- Autenticação;
- Qualidade dos exemplos e guias.

Cada critério foi analisado para verificar a clareza, completude e facilidade de uso, proporcionando uma visão geral da utilidade e eficácia da documentação para os desenvolvedores. Os resultados seguem no Quadro 7.

A partir do Quadro 7, observa-se que as plataformas Strapi e Supabase são as únicas a receber a avaliação mais alta ("Excelente") na categoria de criação de tabelas. Todas as plataformas recebem a avaliação de "Muito bom"no critério de acesso aos dados das tabelas. No que diz respeito à autenticação, apenas a plataforma Directus é avaliada de forma inferior às demais, com um "Razoável". Quanto à qualidade dos exemplos e guias, todas as plataformas são classificadas como "Bom".

Quadro 7 – Resultados da Avaliação da Documentação

Plataforma	**Criação das tabelas**	**Acesso aos dados das tabelas**	**Autenticação**	**Qualidade dos exemplos e guias**
Directus	4 - Muito bom	4 - Muito bom	2 - Razoável	3 - Bom
Strapi	5 - Excelente	4 - Muito Bom	3 - Bom	3 - Bom
Parse	4 - Muito bom	4 - Muito bom	3 - Bom	3 - Bom
Supabase	5 - Excelente	4 - Muito bom	3 - Bom	3 - Bom
Firebase	4 - Muito bom	4 - Muito bom	3 - Bom	3 - Bom

Fonte: Elaborado pelo o autor.

5.5.2 Critérios quantitativos

Nesta seção são descritos todos os atributos quantitativos escolhidos para realizar os testes em cada plataformas, sendo agrupados em um dos dois grupos de critérios maiores, o de critério quantitativo. Os atributos agrupados neste critério são os que podem ser quantificados e ter sua qualidade avaliada em determinados cenários por meio de resultados em quantidade de passos necessários para realizar uma ação, desempenho em tempo de resposta de determinada ferramenta em casos específicos, etc. Para a obtenção dos resultados dos atributos presentes neste grupo de critério que visam tempo de resposta em execução ou passos, foi utilizado um notebook Lenovo IdeaPad Gaming 3i, com sistema operacional Windows 11, versão 23H2, com 16GB de RAM e um processador Intel i5 de 11ª geração. As operações que visam tempo de resposta em execução de cada plataforma foram executadas 20 vezes para o menor enviesamento dos dados ou a menor disparidade entre as plataformas na primeira conexão com o banco de dados.

Tamanho da comunidade

Para este atributo, foram reunidos dados provenientes das plataformas GitHub e Stack Overflow e a partir da reunião dos dados foi preciso a divisão do atributo em dois outros atributos filhos, respectivos ao tamanho da comunidade em cada uma das plataformas. Os dados extraídos do GitHub foram os seguintes:

- Número de *commits* no repositório central;
- Número de desenvolvedores que contribuíram (*contributors*) para o repositório;
- Número de estrelas (*stars*) fornecidas;
- Número de *forks* do projeto.

Quadro 8 – Tamanho da comunidade, dados obtidos dia 24 de Agosto

Plataformas	**Tamanho da comunidade (Github)**	**Tamanho da comunidade (Stack Overflow)**
Directus	12.173 commits, 409 contributors, 26.840 stars e 3.826 forks	373
Strapi	**33.963 commits**, 958 contributors, 62.468 stars e **7.816 forks**	2.867
Parse	4.596 commits, 317 contributors, 20.793 stars e 4.767 forks	1.847
Supabase	28.924 commits, **1.295 *contributors***, **70.810 stars** e 6.723 forks	2.066
Firebase	-*	**144.028**

Fonte: Elaborado pelo o autor.
(*): Sem dados sobre o projeto Firebase.

A partir dos dados reunidos no Quadro 8, é observado que nenhuma plataforma se destacou em todos os critérios avaliados. O Firebase apresentou o melhor desempenho em termos de presença na comunidade do Stack Overflow. O Supabase se destacou no número de contribuidores e estrelas no GitHub, enquanto o Strapi liderou nos quesitos de número de *commits* e *forks*.

Modelos de dados aceitos

A análise desse atributo consiste em uma pesquisa na documentação oficial de cada plataforma. As plataformas *low-code*, em sua maioria, oferecem suporte a uma variedade de modelos de dados, permitindo que os desenvolvedores escolham a abordagem mais adequada para cada projeto. Dentre os modelos mais comuns, as plataformas utilizam os seguintes modelos:

Quadro 9 – Modelos de dados aceitos pelas plataformas

Plataformas	**Modelos de dados**
Directus	SQL
Strapi	SQL
Parse	**SQL, NoSQL**
supabase	SQL
Firebase	NoSQL

Fonte: Elaborado pelo o auto.

Banco de dados aceitos pelas plataformas

Para a análise e comparação deste atributo, foi considerada a documentação oficial de cada plataforma para reunir a quantidade de banco de dados suportados. O Quadro 10 apresenta os resultados obtidos a partir da pesquisa mencionada acima.

Quadro 10 – Banco de dados aceitos pelas plataformas

Plataformas	**Bancos de dados**
Directus	PostgreSQL, MySQL, SQLite, OracleDB, Microsoft SQL Server e CockroachDB
Strapi	PostgreSQL, SQLite, MySQL
Parse	PostgreSQL, MongoDB
Supabase	PostgreSQL
Firebase	Firestore, Realtime Database

Fonte: Elaborado pelo o autor.

Desempenho em tempo de execução

Conforme mencionado na Seção 4.4, desenvolve-se um *backend* para cada plataforma visando realizar os testes. Esses testes são executados em uma ferramenta específica para APIs, consistindo em 20 requisições na mesma rota. Os tempos de resposta são somados e divididos por 20, a fim de calcular a média de tempo.

Os testes aplicados concentram-se nas tabelas de usuário e tarefa, englobando as seguintes operações:

- Criação;
- Consulta;
- Atualização;
- Exclusão.

Usuários:

Quadro 11 – Desempenho em tempo de execução na tabela de usuários

Plataformas	**Criação**	**Consulta**	**Atualização**	**Exclusão**
Directus	**130,85ms**	129,01ms	**128,34ms**	128,27ms
Strapi	132,85ms	**127,80ms**	128,59ms	**102,90ms**
Parse	140,60ms	136,30ms	131,62ms	132,04ms
Supabase	150,07ms	146,68ms	138,58ms	141,25ms
Firebase	662,01ms	713,12ms	769,67ms	610,29ms

Fonte: Elaborado pelo o autor.

Tarefas:

Quadro 12 – Desempenho em tempo de execução na tabela de tarefas

Plataformas	**Criação**	**Consulta**	**Atualização**	**Exclusão**
Directus	**130.99ms**	130.12ms	129.21ms	128.77ms
Strapi	133.51ms	**127.95ms**	**128.79ms**	**103.41ms**
Parse	147.80ms	137.58ms	134.62ms	131.88ms
Supabase	152.46ms	147.97ms	140.82ms	124.65ms
Firebase	655.17ms	712.69ms	749.29ms	630.72ms

Fonte: Elaborado pelo o autor.

Funcionalidades extras

Este critério compara funcionalidades que podem ser um diferencial a ser considerado na escolha de uma solução. São recursos adicionais que, dependendo do contexto, podem agregar valor à aplicação.

Essas funcionalidades adicionais podem impactar a decisão final com base em requisitos específicos do projeto, como suporte em tempo real, facilidade de integração de autenticação e modularidade da plataforma.

Quadro 13 – Comparativo de funcionalidades extras

Funcionalidades	Directus	Strapi	Parse	Supabase	Firebase
APIs	Gera REST e GraphQL dinamicamente	Gera REST e GraphQL	Gera REST e GraphQL	Gera REST e GraphQL (pg_graphql)	Gera REST
Autenticação	Gerenciamento de usuários com controle de acesso fino	Suporte a autenticação personalizável	Suporte a autenticação (email/senha, social)	Autenticação integrada com provedores diversos	Autenticação integrada (email/senha, social)
Interface de Usuário	Directus Studio para gerenciamento sem código	Painel administrativo	Painel administrativo	Interface de administração para gerenciamento de dados	Console do Firebase para gerenciamento
Modularidade e Extensibilidade	Altamente extensível com plugins	Modular com suporte a plugins	Extensível via código	Menos extensível em comparação com outros	Menos flexível em personalização
Suporte a Tempo Real	Sim, via WebSockets	Sim, via Webhooks	Não nativo	Sim, via WebSockets	Sim, via Firestore

Fonte: Elaborado pelo o autor.

5.6 Quadro comparativo completo com os resultados obtidos

Esta seção disponibiliza uma referência que reúne, em um único local, todos os resultados obtidos ao longo deste trabalho, organizados em formato de tabela. A tabela, de acesso público[1], apresenta consolidadamente todos os dados já apresentados anteriormente, facilitando a consulta e a comparação dos resultados.

[1] https://encurtador.com.br/x3P0u

6 CONCLUSÃO E TRABALHOS FUTUROS

Com base nos experimentos e nos resultados obtidos ao longo do estudo, conclui-se que, apesar da variedade de ferramentas *low-code* disponíveis para o desenvolvimento de software, nenhuma se destaca absolutamente em todos os critérios avaliados. A análise comparativa engloba tanto aspectos qualitativos quanto quantitativos, como curva de aprendizado, simplicidade de configuração, desempenho em tempo de execução e a qualidade da documentação das plataformas.

Entre as ferramentas analisadas: Directus, Strapi, Parse, Supabase e Firebase observam-se variações significativas de desempenho em diferentes áreas. Por exemplo, o Firebase apresenta um desempenho inferior em testes de tempo de execução em comparação às demais plataformas, especialmente por não ser testado em ambiente local, impactando diretamente os tempos de resposta. Contudo, destaca-se pela robustez em outras áreas, como na qualidade de sua documentação e no suporte a funcionalidades adicionais, como a integração com APIs REST e WebSockets (não nativo), além de ser amplamente adotado pela comunidade de desenvolvedores.

O estudo indica que a escolha da plataforma mais adequada está diretamente relacionada aos requisitos específicos de cada projeto. Para projetos que exigem maior flexibilidade e controle sobre a infraestrutura, plataformas como Directus, Strapi, Parse e Supabase mostram-se mais adequadas, principalmente por possibilitarem a hospedagem em servidores próprios. Por outro lado, o Firebase permanece como uma opção viável para projetos que podem ser alocados na nuvem, devido ao seu suporte nativo a funcionalidades como autenticação e sincronização em tempo real.

Ademais, a modularidade e a extensibilidade variam entre as plataformas, com algumas se destacando pelo suporte a *plugins* e pela integração com outras ferramentas, enquanto outras apresentam limitações nesse aspecto. Essa característica pode ser um fator determinante dependendo do escopo do projeto, especialmente em casos de aplicações que exigem customizações frequentes ou um ecossistema rico em funcionalidades extras.

Portanto, ao avaliar diferentes ferramentas *low-code* para o desenvolvimento de software, os requisitos do projeto, tanto funcionais quanto não funcionais, assim como a necessidade de escalabilidade e controle sobre a infraestrutura, desempenham um papel central na escolha da plataforma ideal. Estudos futuros podem focar em testar essas plataformas sob condições de maior carga, avaliando seu desempenho com volumes maiores de dados, bem como sua

escalabilidade e capacidade de lidar com estruturas de dados mais complexas.

Para trabalhos futuros, é recomendável a criação de testes de carga a serem aplicados nas APIs desenvolvidas no presente estudo, visando aumentar a confiabilidade dos resultados obtidos, especialmente no que diz respeito ao desempenho de cada plataforma *low-code*. A implementação desses testes de carga em cada uma das APIs também permite a obtenção de novos dados, que podem ser incorporados ao catálogo de atributos, como, por exemplo, a escalabilidade das plataformas. Adicionalmente, é pertinente avaliar como as plataformas lidam com grandes volumes de dados nas tabelas, relacionamentos complexos de muitos para muitos, e consultas de alta complexidade.

REFERÊNCIAS

ALAMIN, M. A. A.; MALAKAR, S.; UDDIN, G.; AFROZ, S.; HAIDER, T. B.; IQBAL, A. An empirical study of developer discussions on low-code software development challenges. In: IEEE. **2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)**. [*S. l.*], 2021. p. 46–57.

Brasscom. **Demanda de Talentos em TIC e Estratégia TCEM**. 2021. Disponível em: https://brasscom.org.br/pdfs/demanda-de-talentos-em-tic-e-estrategia-tcem/. Acesso em: 25 de junho 2023.

DATE, C. J. **Introdução a sistemas de bancos de dados**. [*S. l.*]: Elsevier Brasil, 2004.

ELMASRI, R.; NAVATHE, S. B.; PINHEIRO, M. G. *et al.* Sistemas de banco de dados. Pearson Addison Wesley São Paulo, 2005.

FLEURY, A. L.; SPINOLA, M. d. M.; LAURINDO, F. J. B.; PESSÔA, M. S. d. P. Alinhando objetivos estratégicos e processo de desenvolvimento em empresas de software. **Production**, SciELO Brasil, v. 24, p. 379–391, 2014.

FRANKENTHAL, R. Entenda a escala likert e como aplicá-la em sua pesquisa. **Mind Miners. Disponível em:< https://mindminers. com/pesquisas/entenda-o-que-e-escala-likert>. Acesso em**, v. 28, 2017.

Gartner. **Gartner Forecasts Worldwide Low-Code Development Technologies Market to Grow 20% in 2023**. 2022. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2022-12-13-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-20-percent-in-2023. Acesso em: 29 de junho 2023.

JÚNIOR, S. D. d. S.; COSTA, F. J. Mensuração e escalas de verificação: uma análise comparativa das escalas de likert e phrase completion. **PMKT–Revista Brasileira de Pesquisas de Marketing, Opinião e Mídia**, v. 15, n. 1-16, p. 61, 2014.

KÄSS, S.; STRAHRINGER, S.; WESTNER, M. Practitioners' perceptions on the adoption of low code development platforms. **Ieee Access**, IEEE, v. 11, p. 29009–29034, 2023.

KHORRAM, F.; MOTTU, J.-M.; SUNYÉ, G. Challenges & opportunities in low-code testing. In: **Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings**. [*S. l.*: *s. n.*], 2020. p. 1–10.

KRISHNARAJ, N.; VIDHYA, R.; SHANKAR, R.; SHRUTHI, N. Comparative study on various low code business process management platforms. In: IEEE. **2022 International Conference on Inventive Computation Technologies (ICICT)**. [*S. l.*], 2022. p. 591–596.

Mariana Alves. **Por que a demanda por Desenvolvedores de Software será ainda maior em 2024?** 2024. Disponível em: https://ubiminds.com/pt-br/demanda-por-desenvolvedores-de-software-2024/. Acesso em: 28 de setembro 2024.

MASSE, M. **REST API design rulebook**. [*S. l.*]: "O'Reilly Media, Inc.", 2011.

MORAES, P. H. M. M. Aplicação de ferramentas low-code para melhoria e automação de processos em uma empresa de contabilidade. 2022.

Outsystems. **Top Application Development Trends in 2019**. 2019. Disponível em: https://www.outsystems.com/1/state-app-development-trends/. Acesso em: 28 de junho 2023.

RICHARDSON, C.; RYMER, J. R.; MINES, C.; CULLEN, A.; WHITTAKER, D. New development platforms emerge for customer-facing applications. **Forrester: Cambridge, MA, USA**, v. 15, 2014.

ROCHA, G. C. C. d. Uma análise comparativa entre frameworks de persistência de dados em javascript. 2022.

Safetec. **Confira as vantagens de aprimorar os investimentos em TI no negócio**. 2023. Disponível em: https://safetec.com.br/casos-de-sucesso/tecnologia/investimentos-em-ti/. Acesso em: 28 de setembro 2024.

SAHAY, A.; INDAMUTSA, A.; RUSCIO, D. D.; PIERANTONIO, A. Supporting the understanding and comparison of low-code development platforms. In: IEEE. **2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. [*S. l.*], 2020. p. 171–178.

SANCHIS, R.; GARCÍA-PERALES, Ó.; FRAILE, F.; POLER, R. Low-code as enabler of digital transformation in manufacturing industry. **Applied Sciences**, MDPI, v. 10, n. 1, p. 12, 2019.

SOBRI, N. A. N.; ABAS, M. A. H.; YASSIN, A. I. M.; ALI, M. S. A. M.; TAHIR, N. M.; ZABIDI, A.; RIZMAN, Z. I. Comparison between headless cms and backend-as-a-service products for e-suripreneur backend. **Mathematical Statistician and Engineering Applications**, v. 71, n. 3s2, p. 928–938, 2022.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson, 2011. v. 9. Disponível em: http://www.facom.ufu.br/~william/Disciplinas%202018-2/BSI-GSI030-EngenhariaSoftware/Livro/engenhariaSoftwareSommerville.pdf. Acesso em: 21 jul. 2023.

TV, L. C.; ABDULLA, M. S. A survey of low-code/no-code software development tools with an application. 2022.

VAZQUEZ, C. E.; SIMÕES, G. S. **Engenharia de Requisitos: software orientado ao negócio**. [*S. l.*]: Brasport, 2016.

VINCENT, P.; IIJIMA, K.; DRIVER, M.; WONG, J.; NATIS, Y. Magic quadrant for enterprise low-code application platforms. **Gartner report**, 2019.

WASZKOWSKI, R. Low-code platform for automating business processes in manufacturing. **IFAC-PapersOnLine**, Elsevier, v. 52, n. 10, p. 376–381, 2019.