UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

GABRIEL HOLANDA RAULINO

ANÁLISE COMPARATIVA ENTRE PLATAFORMAS DE DESENVOLVIMENTO *LOW CODE* PARA *WEB*

QUIXADÁ

2024

GABRIEL HOLANDA RAULINO

ANÁLISE COMPARATIVA ENTRE PLATAFORMAS DE DESENVOLVIMENTO *LOW CODE* PARA *WEB*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus 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)

R189a Raulino, Gabriel Holanda.
Análise comparativa entre plataformas de desenvolvimento low code para web / Gabriel Holanda Raulino. – 2024.
65 f. : il. color.

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

1. Low Code. 2. Comparação entre plataformas low code. 3. MVP. I. Título.

CDD 005

GABRIEL HOLANDA RAULINO

ANÁLISE COMPARATIVA ENTRE PLATAFORMAS DE DESENVOLVIMENTO *LOW CODE* PARA *WEB*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus 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. Jeferson Kenedy Morais Vieira
Universidade Federal do Ceará (UFC)

A todos que acreditaram em mim e me apoiaram ao longo desta jornada. Sua presença e incentivo foram fundamentais para que eu pudesse seguir em frente, mesmo nos momentos mais desafiadores. Sou eternamente grato por cada palavra de encorajamento e por estarem ao meu lado.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Meiry Any Holanda Saraiva, por estar sempre presente, zelando pelo meu bem-estar físico e mental, e José Alci Raulino, por investir em mim, mesmo nos momentos em que estive desmotivado. Vocês foram o suporte que me manteve de pé, possibilitando que eu seguisse em frente nesta jornada até o fim.

Ao Prof. Dr. Régis Pires Magalhães, por me guiar e orientar ao longo deste processo, mostrando que era possível. Suas observações e paciência foram fundamentais para que eu alcançasse este resultado.

À minha namorada, Ana Beatriz de Oliveira Ribeiro, pelo companheirismo e compreensão. Agradeço pelas noites em claro, pela paciência diante dos meus estresses e frustrações, e por sempre me incentivar a seguir em frente, sem me deixar abater.

A Luana Carolaine de Lima, que, mesmo sobrecarregada de tarefas, esteve sempre pronta a me ajudar. Sua orientação foi essencial desde o início, quando eu ainda não sabia nem como escrever uma introdução. Obrigado por me acalmar e mostrar que essa etapa era possível de ser concluída.

Ao meu irmão, Ricardo Holanda Raulino, pelos momentos de descontração com jogos de tabuleiro e por ter me salvado com o computador quando meu notebook ficou sem carregador.

Ao Ygor Duarte, chefe que se tornou colega de trabalho e, posteriormente, um amigo que planejo levar para a vida. Seus conselhos e confiança em mim me mostraram que tudo era possível, desde que eu me dedicasse. Sou grato por sempre acreditar em mim.

Ao meu irmão de outra mãe, Josué Nicholson, pela nossa longa amizade e pelas experiências que compartilhamos, desde o ensino fundamental até hoje, como colegas de trabalho. Agradeço por tudo o que vivemos juntos.

Ao meu melhor amigo, Devid Iriel Lima da Silveira, por me mostrar o significado de resiliência e proporcionar aventuras inimagináveis.

Ao Victor Caio, meu parceiro de TCC, por me apoiar nos momentos em que eu quase desistia, sempre compartilhando opiniões e conhecimentos que visavam nossa aprovação. Conseguimos juntos, meu amigo!

Por fim, meu colega de trabalho Luis Felipe Amorim, por me tranquilizar no final da jornada e auxiliar no desenvolvimento da apresentação, sua ajuda foi valiosa, obrigado.

"Você pode ser um guerreiro forte, mas nunca vai vencer lutando sozinho." (Vegeta)

RESUMO

Diante das restrições financeiras e da escassez de especialistas em tecnologia da informação, muitas empresas enfrentam dificuldades para desenvolver soluções de software ágeis. Nesse contexto, as plataformas de desenvolvimento low code surgem como uma solução eficaz, permitindo a criação rápida de software sem a necessidade de amplo conhecimento em programação. Além de facilitar o desenvolvimento de *Minimum Viable Product*s (MVPs), essas plataformas são amplamente utilizadas para criar aplicações mais robustas, capazes de escalar e atender às demandas de negócios em diferentes contextos. No entanto, mesmo com um escopo reduzido, a criação de um MVP pode ser complexa e demandar tempo significativo da equipe de desenvolvimento. Este trabalho realiza uma análise comparativa entre diversas plataformas de desenvolvimento *low-code* com foco no desenvolvimento de aplicações web. A metodologia adotada neste trabalho inclui a implementação de um MVP em plataformas como *Adalo, Bubble, FlutterFlow* e *Outsystems*, avaliadas por critérios como usabilidade, personalização, escalabilidade e facilidade de implementação. Os resultados indicam que o *Adalo* é mais adequado para projetos menores devido à sua simplicidade, enquanto o *Bubble* oferece flexibilidade, mas limitações quanto à exportação de código. O *FlutterFlow* destaca-se pela personalização e pela possibilidade de exportar o código-fonte, sendo ideal para projetos de maior escala. O *Outsystems* sobressai em grandes projetos corporativos, apesar de exigir uma curva de aprendizado mais acentuada. Conclui-se que a escolha da plataforma depende das necessidades específicas de cada projeto. Como trabalhos futuros, sugere-se ampliar a análise para incluir outras plataformas e explorar a integração dessas ferramentas com tecnologias emergentes, como inteligência artificial e automação de processos.

Palavras-chave: *low code*; MVP; comparação entre plataformas *low code*.

ABSTRACT

"Given the financial constraints and the shortage of IT specialists, many companies face difficulties in developing agile software solutions. In this context, low code development platforms emerge as an effective solution, enabling the rapid creation of software without requiring extensive programming knowledge. In addition to facilitating the development of MVPs, these platforms are widely used to create more robust applications, capable of scaling and meeting business demands in different contexts. However, even with a reduced scope, creating an MVP can be complex and require significant time from the development team. This work presents a comparative analysis of various development platforms, focusing on web application development. The methodology adopted in this work includes the implementation of an MVP in platforms such as Adalo, Bubble, FlutterFlow, and Outsystems, evaluated by criteria such as usability, customization, scalability, and ease of implementation. The results indicate that Adalo is more suitable for smaller projects due to its simplicity, while Bubble offers flexibility but has limitations regarding code export. FlutterFlow stands out for its customization and the possibility of exporting source code, making it ideal for larger-scale projects. Outsystems excels in large corporate projects, although it requires a steeper learning curve. It is concluded that the choice of platform depends on the specific needs of each project. Future work suggests expanding the analysis to include other platforms and exploring the integration of these tools with emerging technologies, such as artificial intelligence and process automation.

Keywords: low code, MVP, comparison between low code platforms.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
LCD	*Low Code Development*
LCDP	*Low Code Development Platform*
MVP	*Minimum Viable Product*
NCD	*No Code Development*
NCDP	*No Code Development Platform*
PaaS	*Plataform as a Service*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, o mercado de trabalho conta com um déficit de mão de obra na área de desenvolvimento de *software*, conforme especificado por Valença *et al.* (2023) explicando que "A problemática do déficit de capital humano especializado não é nova, o setor de TI nas últimas duas décadas enfrenta dificuldades para suprir a demanda por mão de obra" (Valença *et al.*, 2023, p. 2). Além disso, muitas vezes, os recursos para um projeto podem ser escassos, seja no quesito de tempo ou no de financiamento. É com esse paradigma que surgem as *Low Code Development Platform*s (LCDPs), traduzido como plataformas de desenvolvimento *low code*. Estas plataformas visam facilitar principalmente a construção dos aplicativos em menos tempo e podem ter como público alvo pessoas que não precisam, necessariamente, ter alguma formação na área de tecnologia da informação (TI) para utilizá-las.

Segundo uma pesquisa feita pelo Brasscom (2021) anseia-se que até 2025 o Brasil necessitará de 797 mil profissionais da área de TI. Diante disso, é possível perceber que quanto mais rápido acontecer a especialização desses profissionais, mais fácil essas vagas serão preenchidas. Visto que, o objetivo é que pessoas, conhecidas como desenvolvedor cidadão, denominadas por Bastos (2023) "[...] ser um funcionário responsável por criar recursos para aplicações para o próprio consumo ou de outros" (Bastos, 2023, p. 12) consigam utilizar uma LCDP para amenizar a demanda citada de profissionais.

Além disso, conforme a pesquisa de Vincent *et al.* (2019), que analisou 18 fornecedores diferentes, há uma vasta quantidade de LCDPs disponíveis no mercado. Essas plataformas se destacam por oferecer uma combinação entre produtividade e agilidade na entrega, atendendo tanto desenvolvedores profissionais quanto cidadãos, evidenciando a grande variedade de opções disponíveis atualmente no mercado.

De acordo com Alwis e Sedera (2022), muitas empresas de desenvolvimento de sistemas de informação focam nos aspectos técnicos e gerenciais, mas falham ao considerar o valor percebido pelos clientes, o que leva ao desperdício de tempo e recursos em produtos que não atendem às expectativas do mercado. Este é outro fator que impulsiona a utilização das LCDPs, já que é possível construir com agilidade um *Minimum Viable Product* (MVP), traduzido como produto mínimo viável, para validar a ideia e as necessidades do cliente.

No entanto, alguns pontos precisam ser observados sobre o uso de uma LCDP, como, por exemplo: a capacidade de personalização do *layout* e da responsividade, que afeta a adaptação a diferentes dispositivos e tamanhos de tela; a flexibilidade na integração com bancos

de dados; ou a possibilidade de bloqueio do fornecedor (*vendor lock-in*). Segundo Kumar e Mala (2022), o termo *vendor lock-in* refere-se ao uso forçado de todos os serviços de uma única nuvem, independente da qualidade do mesmo, fazendo com que a organização fique presa ao seu provedor.

Diante disso, é necessário a busca por plataformas que amenizem essas problemáticas, tendo em vista também a vasta gama de opções, é importante fazer a escolha certa da mesma, a fim de evitar problemas futuros, seja por restrições da plataforma ou financeiras.

Assim, o presente trabalho visa avaliar e comparar essas LCDPs com o intuito de definir: os recursos que podem ser utilizados; as limitações que possam impactar ao longo do desenvolvimento; e quais são as que atendem o máximo de requisitos estabelecidos para um MVP, que tem a finalidade de ser um dos parâmetros para a comparação.

1.1 Objetivos

Nesta seção, são descritos o objetivo geral e objetivos específicos do presente trabalho.

1.1.1 Objetivo Geral

Comparar as plataformas *low code* para definir limitações e recursos disponíveis, identificando as mais adequadas para diferentes cenários de uso, utilizando o MVP como recurso metodológico na realização da comparação.

1.1.2 Objetivos Específicos

- Especificar os requisitos do MVP que será criado em cada ferramenta selecionada neste trabalho;
- Selecionar as plataformas de desenvolvimento *low code* que serão objetos de comparação deste trabalho;
- Definir os critérios de avaliação que serão usados para a comparação das plataformas *low code*;
- Projetar e implementar o MVP em cada plataforma *low code* selecionada para comparação;
- Comparar as plataformas *low code* selecionadas segundo os critérios de avaliação estabelecidos.

1.2 Organização do texto

Os próximos capítulos estão organizados da seguinte maneira: O Capítulo 2, apresenta a fundamentação teórica e os conceitos que apoiam o ponto de vista proposto neste trabalho; o Capítulo 3 expõe os trabalhos relacionados, com descrição e comparação de pesquisas e projetos que possuem aspectos similares aos especificados neste trabalho; o Capítulo 4 introduz os procedimentos metodológicos, onde são descritos os passos para a realização deste trabalho; O Capítulo 5 apresenta os resultados obtidos a partir da metodologia usada; Por último, o Capítulo 6 descreve a conclusão e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para formar a fundamentação teórica deste trabalho, a Seção 2.1 dispõe os conceitos de *low code* e *no code*, destacando que, neste trabalho, o segundo é considerado um subconjunto do primeiro; já a Seção 2.2 explora o conceito de *Low Code Development Platform* (LCDP), destacando que essas plataformas podem ser implementadas tanto em ambientes *cloud*, pelo modelo *Plataform as a Service* (PaaS), quanto *on-premise*, possibilitando a criação e implantação de aplicativos de *software* totalmente funcionais; e, finalmente, a Seção 2.3 introduz o conceito de *Minimum Viable Product* (MVP), que será utilizado para gerar um produto nas plataformas selecionadas.

2.1 *Low code* e *no code*

De acordo com Metrolho *et al.* (2019), o termo *low code* é conceituado como um *software* que provê um ambiente de desenvolvimento usado para desenvolver aplicações através de interfaces gráficas de usuários e configurações, ao invés do método de programação tradicional. Além disso, é possível, também, escrever algumas linhas de código quando necessário, seja para customizar a interface, integrar com terceiros ou aprimorar alguma funcionalidade da aplicação que está sendo desenvolvida.

Ainda sobre o termo, segundo Rokis e Kirikova (2023), "[...] é uma abordagem que promove um desenvolvimento rápido, flexível e iterativo, permitindo a tradução rápida de requisitos de negócios por meio de programação visual com uma interface gráfica, abstração visual e mínimo uso de código manual" (Rokis; Kirikova, 2023, p. 72). Esses benefícios tornam esta tecnologia altamente atrativa, ao eliminarem a necessidade de profundo conhecimento técnico na área de TI. A possibilidade de criar aplicações de forma ágil e adaptável às demandas, utilizando recursos visuais simplificados, faz com que sejam uma excelente solução para atender às crescentes exigências do mercado atual.

Com base nos dados levantados pela pesquisa de Bucaioni *et al.* (2022), o *Low Code Development* (LCD) é amplamente utilizado em diversos setores de negócios no campo de *software*, abrangendo 21 áreas distintas, que vão desde o desenvolvimento de *websites* e aplicativos até indústrias mais especializadas, como a aeronáutica. Esta amplitude de uso demonstra a capacidade do LCD de se adaptar a diferentes cenários tecnológicos, atendendo tanto a demandas mais convencionais, como o desenvolvimento de soluções digitais para a *web* e

dispositivos móveis, quanto a setores de alta complexidade, como a manufatura e a aviação. Essa versatilidade reflete o potencial dessa abordagem em otimizar processos e promover a inovação em múltiplos contextos.

Bucaioni *et al.* (2022) destaca que as quatro principais áreas de aplicação do LCD incluem o desenvolvimento *web*, *mobile*, serviços empresariais e processos de negócios, evidenciando sua relevância em suportar tanto soluções corporativas quanto tecnológicas, facilitando a criação e atualização de sistemas de *software* em diferentes mercados.

Enquanto isso, o *no code* concentra-se na construção de aplicativos sem escrever nenhuma linha de código (Bloomberg, 2017; Vincent *et al.*, 2019), em que apenas algumas entradas de textos são necessárias para formar expressões, fórmulas e outros requisitos para o desenvolvimento de aplicativos mantidos em todo o visual, modelagem e configuração (Vincent *et al.*, 2019). Outra característica notória é a capacidade de arrastar e soltar os componentes para montar a interface e desenvolver lógicas na aplicação.

Apesar de o *low code* ser o foco do trabalho, é necessário destacar as semelhanças entre as nomenclaturas, mesmo que existam diferenças entre elas. Para tanto, Käss *et al.* (2022) cita uma discussão, em alta, na academia e na indústria sobre o uso dos termos LCD e *No Code Development* (NCD), alguns consideram totalmente distintos e outros como um subconjunto. Adota-se a segunda abordagem para este trabalho, sendo assim, as referências utilizadas serão nomeadas como LCD, tanto para os termos em NCD quanto para LCD.

2.2 *Low Code Development Platform* (LCDP)

De acordo com Sahay *et al.* (2020), as plataformas de desenvolvimento *low code* (em inglês, LCDP) podem ser baseadas em *cloud* (entregues pelo modelo PaaS) ou *on-premise*, permitindo a criação e implantação de aplicativos de *software* totalmente funcionais.

O modelo de serviço de computação em nuvem PaaS "[...] fornece uma estrutura ou meio aceitável para o desenvolvedor construir aplicativos e programas e distribuí-los na rede sem instalar ou gerenciar o ambiente de produção." (Mohammed; Zeebaree, 2021, p.20), possibilitando assim, economia no tempo e redução de erros por configuração do ambiente de desenvolvimento. Um exemplo de PaaS conhecido é o *Heroku*, a qual é uma plataforma em que é possível implantar, gerenciar e dimensionar aplicativos modernos, através da sua interface amigável. Além disso, é altamente independente e integra serviços de dados, bem como um ecossistema completo próprio.

As LCDP utilizam interfaces gráficas avançadas e abstrações visuais que exigem pouco ou nenhum código, facilitando o desenvolvimento de aplicações de forma simplificada. Entre os principais benefícios delas estão a simplicidade, reusabilidade, interoperabilidade, baixo custo e tempo reduzido de lançamento no mercado (Miyake *et al.*, 2023). Isso ocorre porque elas dispensam a necessidade de grandes investimentos em treinamentos ou na contratação de novos funcionários com conhecimentos especializados em tecnologias de desenvolvimento.

A pesquisa realizada por Richardson e Rymer (2016), identifica os cinco segmentos principais das plataformas de desenvolvimento *low-code*:

1. Propósito geral: Oferecem uma ampla variedade de funcionalidades, incluindo ferramentas para experiência do usuário, controle de acesso, integração e implantação, por exemplo: *OutSystems* e *Salesforce*;
2. Aplicativos de processo: Voltadas para a criação de aplicações de coordenação e colaboração, utilizando ferramentas visuais de modelagem de processos, por exemplo: *Appian* e *Ultimus*;
3. Aplicativos de banco de dados: Projetados para facilitar a criação de aplicativos que manipulam dados em bancos relacionais, oferecendo ferramentas para a construção de interfaces, lógica de aplicação, e gerenciamento de esquemas, por exemplo: *Oracle* e *Alphinat*;
4. Gerenciamento de requisições: Facilitam o processamento, manuseio e rastreamento de solicitações, por exemplo: *ServiceNow* e *Cherwell software*;
5. Desenvolvimento de aplicativos móveis: Especializadas no desenvolvimento de aplicativos para dispositivos móveis, com ferramentas dedicadas a essa finalidade, por exemplo: *Snappii* e *Appery.io*.

Embora essas plataformas ofereçam uma ampla gama de funcionalidades, há variações significativas entre as diferentes soluções disponíveis no mercado. No estudo conduzido por Rokis e Kirikova (2023), foi feita uma comparação das LCDPs por meio de diagramas e listas de recursos, identificando semelhanças e variações. Ao revisar a literatura existente e aplicar uma síntese temática, Rokis e Kirikova (2023) identificaram funcionalidades adicionais. Contudo, os autores ressaltam que nem todas suportam as mesmas funcionalidades da mesma forma ou com a mesma abrangência, o que torna desafiador definir um conjunto uniforme de características essenciais.

Portanto, para este trabalho, as plataformas analisadas enquadram-se principalmente

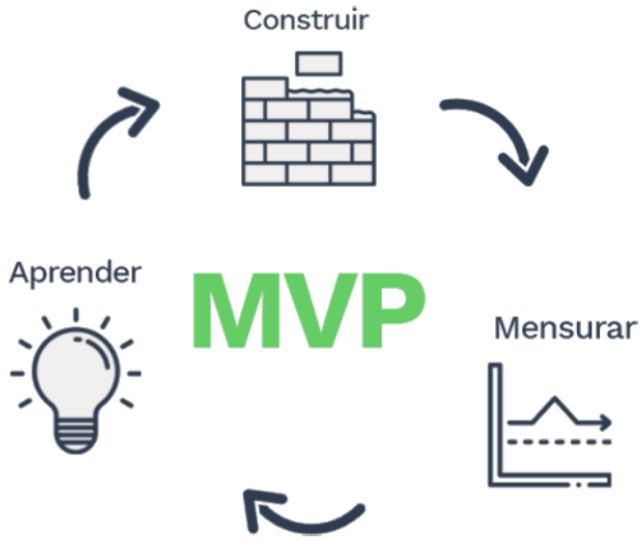
no segmento de desenvolvimento de propósito geral.

2.3 *Minimum Viable Product* (MVP)

Um estudo de mapeamento sistemático conduzido por Lenarduzzi e Taibi (2016) demonstrou que existem diversas definições de MVP. Deste modo, os autores concluíram, então, que a melhor definição para MVP é: a versão de um novo produto com o mínimo de funcionalidades que permita coletar o máximo de aprendizado, validado sobre os clientes com o mínimo esforço.

A Figura 1 representa o ciclo construir-medir-aprender, que de acordo com (Sankhya, 2023), é um processo iterativo de desenvolvimento que envolve criar um MVP, coletar opiniões dos usuários para medir seu desempenho e aprender com os dados para ajustar ou pivotar a ideia. Esse método é essencial para validar hipóteses e otimizar produtos conforme as necessidades do mercado.

Figura 1 – Passo a passo de como criar um MVP



Fonte: Sankhya (2023)

Podemos definir também que o produto mínimo viável é "[...] aquela versão do produto que permite uma volta completa do ciclo construir-medir-aprender, com o mínimo de esforço e o menor tempo de desenvolvimento." (Ries *et al.*, 2019, p. 44), em outras palavras, é a representação do núcleo do negócio, em que estarão as funcionalidades mínimas necessárias para caracterizar o produto e, principalmente, receber o retorno do cliente.

Para entender melhor, imagine o seguinte cenário: um cliente precisa de um aplicativo de gestão de tarefas para equipes, em que os membros podem criar, atribuir e monitorar o progresso delas, colaborativamente. A ideia é melhorar a comunicação e a eficiência na execução de projetos. O mínimo de funcionalidades para o MVP seria: registro e autenticação de usuários; criação de tarefas; atualizações em tempo real e uma interface simplificada. Outras funcionalidades como, por exemplo: notificações, priorização de tarefas e histórico de atividades fazem parte da solução, mas não são essenciais para a aplicação.

Neste trabalho, será especificado um MVP que posteriormente será implementado em cada LCDP para avaliar seus benefícios, limitações e impactos práticos.

3 TRABALHOS RELACIONADOS

Os trabalhos apresentados neste capítulo são de diferentes categorias, mas todos possuem em comum, o foco em plataformas de desenvolvimento *low code*. O primeiro, presente na Seção 3.1, se trata de uma comparação de LCDPs através de uma taxonomia fruto da análise e da utilização das mesmas. O segundo, descrito na Seção 3.2, apresenta estimuladores e inibidores para a adoção das LCDPs. Já o terceiro trabalho, exposto na Seção 3.3 trata-se de uma comparação entre quatro LCDPs para catalogar melhorias que podem ser implementadas na plataforma que é o foco de estudo do autor.

A seguir, será descrito de forma mais ampla o que são esses trabalhos e em seguida, a Seção 3.4 mostra como eles se relacionam com a pesquisa proposta.

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

No estudo conduzido por Sahay *et al.* (2020), foi realizada uma análise comparativa de oito LCDPs consideradas líderes de mercado, conforme relatórios das consultorias Gartner e Forrester. O objetivo foi identificar e organizar as funcionalidades mais relevantes dessas plataformas de forma sistemática.

Inicialmente, os autores introduzem o conceito de LCDPs e sua estrutura, abordando cada camada dessas plataformas com profundidade. Nesse estudo inclui como as plataformas são organizadas e, principalmente, o processo de desenvolvimento de *software* nelas, destacando o uso de interfaces visuais e a integração com serviços externos.

Posteriormente, as plataformas selecionadas, baseadas nos relatórios de Gartner e Forrester, são apresentadas. As plataformas analisadas foram escolhidas com base na quantidade de funcionalidades que poderiam agregar aos recursos, previamente, definidos pelos autores. Essas plataformas incluem: *OutSystem*, *Mendix*, *Zoho Creator*, *Microsoft PowerApps*, *Google App Maker*, *Kissflow*, *Salesforce App Cloud* e *Appian*.

A partir dessa análise, os autores propõem uma taxonomia para classificar e organizar as funcionalidades das plataformas, destacando as diferenças e semelhanças entre elas. Essa taxonomia é apresentada em uma tabela que detalha as principais características e capacidades das plataformas estudadas.

Por fim, as plataformas são comparadas utilizando a taxonomia criada. A análise considerou tanto informações obtidas nos sites das plataformas quanto a experiência prática dos

autores ao desenvolver uma aplicação de *benchmark* nessas plataformas. Com isso, foi possível identificar desafios e limitações que desenvolvedores e usuários podem enfrentar ao longo do processo de desenvolvimento dentro das LCDPs, oferecendo uma visão crítica sobre o uso dessas tecnologias em diferentes contextos.

3.2 *Drivers and Inhibitors of Low Code Development Platform Adoption*

Na revisão de literatura feita por Käss *et al.* (2022), foram identificados treze fatores que inibem e sete que impulsionam a adoção das LCDPs. Em seguida, estes foram estruturados com a teoria da difusão da inovação (do inglês, *Diffusion of Innovation Theory*, (DOI)), em que os usuários decidem se uma inovação é aceita ou rejeitada, segundo a crença sobre ela.

No início eles detalham sobre a questão da inovação digital ter se tornado mais urgente devido à pandemia, dando ênfase na falta de pessoas capacitadas para executar aplicações de qualidade e com orçamento razoável. Além disso, é apresentada a proposta da LCDP, pontuando as questões de redução de código escrito, tempo gasto e também o aumento da produção no desenvolvimento.

Como o foco dos autores é abordar os inibidores e impulsionadores, é alertado que o uso das LCDPs pode induzir aos riscos, tais como: as dificuldades com manutenção ou escalabilidade da aplicação, ou, até mesmo, a negligência nos padrões de segurança. Exposta essa questão, os estudiosos explicam que existem duas perspectivas ao analisar a adoção de uma tecnologia: a organizacional e a individual, que neste caso para este trabalho, optaram pela organizacional.

Para iniciar a revisão, os autores usaram banco de dados bibliográficos e serviços de indexação de citações, a fim de recuperar o máximo de informações relevantes, utilizando as palavras-chave "*low code*"e "*no code*". Após a pesquisa, foi feita uma filtragem para eliminar duplicações e remover aqueles que utilizam os termos em contextos diferentes, resultando ao final, um total de 73 artigos relevantes.

Por fim, O Quadro 1 destaca os resultados onde foram listados os impulsionadores e inibidores detalhadamente aplicados ao DOI *framework*, mostrando também o número de ocorrências na amostra da literatura selecionada. As cinco características analisadas: compatibilidade, complexidade, vantagem relativa, testabilidade e observabilidade, são elementos críticos na decisão de adoção de novas tecnologias nas organizações.

No caso da compatibilidade, não foram encontrados impulsionadores, mas vários

Quadro 1 – Impulsionadores e inibidores da adoção das LCDPs (número de ocorrências entre parênteses)

Impulsionador	**Inibidor**
Compatibilidade	
Nenhum impulsionador identificado no conjunto de literatura analisado	• Dificuldades em garantir a interoperabilidade com outras LCDPs (14) • Dificuldade de integrar com outros sistemas (13) • Limitações das funcionalidades das LCDPs (13) • Aumento dos riscos de segurança, privacidade e conformidade (11) • Relutância em mudar comportamentos (4) • Falta de estrutura de governança (3)
Complexidade	
• Barreiras de entrada reduzidas para desevolvimento de aplicações (48) • Fácil de desenvolver aplicações (35)	• Falta de escalabilidade (15) • Falta de flexibilidade e customização (15) • Falta de documentação (15) • Usabilidade limitada das LCDPs (8)
Vantagem relativa	
• Métricas de desempenho aprimoradas do processo de desenvolvimento de software (50) • Modo de trabalho aprimorado do processo de desenvolvimento de software (24) • Melhor resultado do processo de desenvolvimento de software (20) • Dependência reduzida do desenvolvedor de TI (19) • Reação mais rápida à demanda do mercado (7)	Nenhum impulsionador identificado no conjunto de literatura analisado
Testabilidade	
Nenhum impulsionador identificado no conjunto de literatura analisado	• Aprisionamento a um provedor da LCDP (13)
Observabilidade	
Nenhum impulsionador identificado no conjunto de literatura analisado	• Dificuldades em estimar o custo total (9) • Medo da Shadow IT (2)

Fonte: Adaptado de Käss *et al.* (2022)

inibidores, como dificuldades de interoperabilidade com outras plataformas e sistemas, além de preocupações com segurança e conformidade, sendo fatores que podem desencorajar as empresas.

Para a complexidade, um dos principais impulsionadores foi a redução das barreiras de entrada para o desenvolvimento de aplicações, além da facilidade de uso das plataformas. No entanto, a falta de escalabilidade e flexibilidade são inibidores significativos.

Na vantagem relativa, os impulsionadores foram bem expressivos, como a melhoria nas métricas de desempenho e na eficiência do desenvolvimento de software, bem como a redução da dependência de desenvolvedores de TI. Interessantemente, não foram identificados inibidores.

A testabilidade e a observabilidade apresentaram menos dados. Enquanto a testabilidade mostrou o risco de aprisionamento a um provedor específico como um inibidor, a

observabilidade destacou dificuldades em estimar o custo total e preocupações com Shadow IT.

Assim, o Quadro 1 ilustra uma análise clara de como esses fatores podem impactar a adoção das LCDPs, ajudando a compreender as barreiras e benefícios que as organizações enfrentam.

3.3 Análise comparativa entre o *Íris low-code* e outras plataformas *low-code*

Inicialmente, o autor Bastos (2023) discute o problema da escassez de mão de obra qualificada no setor de TI, um desafio que se intensificou com a pandemia, tornando urgente a necessidade de as empresas implementarem soluções de software capazes de conectar funcionários e clientes de maneira eficaz. Nesse contexto, o pesquisador apresenta as plataformas de desenvolvimento *low code* como uma alternativa viável, enfatizando suas vantagens: como a rápida implementação e a redução de custos, além da facilidade de uso por profissionais com pouca ou nenhuma experiência em programação. Por outro lado, também aponta desvantagens, como a possível limitação de personalização e a dependência de fornecedores, o que pode dificultar a adaptação à necessidades específicas de cada organização.

O estudo realiza uma comparação detalhada entre várias plataformas *low code* populares no mercado de TI, incluindo a *Íris Low Code* Certidão, uma plataforma específica utilizada pelo Governo do Estado do Ceará para emissão de certidões. O objetivo é identificar critérios que possam contribuir para a evolução dessa plataforma. O autor utiliza uma metodologia de análise criteriosa, avaliando aspectos como funcionalidade, compatibilidade, escalabilidade e flexibilidade das plataformas analisadas.

Entre os pontos fortes identificados, Bastos (2023) destaca a capacidade das plataformas *low code* de reduzir, significativamente, o tempo de desenvolvimento, principalmente, ao utilizar modelos prontos, facilitando a produção de aplicativos básicos sem a necessidade de programação extensa. Além disso, a compatibilidade com dispositivos móveis foi ressaltada como um fator crucial para melhorar a acessibilidade e a conveniência para os usuários.

No entanto, o estudo também identificou algumas limitações importantes que precisam ser abordadas. O autor conclui que a plataforma Íris precisa evoluir em dois aspectos essenciais: a integração com *Application Programming Interface* (API), fundamental para permitir a conexão com outros sistemas e expandir suas funcionalidades; e a criação de *workflows*, que permitiria automatizar processos internos de maneira mais eficiente, melhorando a produtividade das organizações que utilizam a plataforma.

O trabalho de Bastos (2023) oferece uma visão abrangente sobre o uso de plataformas *low code*, fornecendo percepções práticas para sua evolução futura, especialmente, em ambientes governamentais, onde a necessidade de agilidade e eficiência é elevada.

3.4 Comparação entre trabalhos

Diferente do estudo de Sahay *et al.* (2020), que utilizou relatórios do Gartner e Forrester para a escolha das plataformas, neste trabalho a seleção será realizada com base nos resultados de uma pesquisa que avaliará a capacidade das mesmas de atender aos requisitos, previamente, estabelecidos. Os pontos em comum entre os dois trabalhos incluem a comparação das plataformas, o uso delas para desenvolver uma aplicação e o suporte na tomada de decisão com base nos resultados obtidos.

De maneira similar ao estudo de Käss *et al.* (2022), será realizada uma pesquisa por palavras-chave, além de destacar as finalidades recomendadas para cada plataforma. No entanto, os aspectos que diferenciam os dois trabalhos incluem o foco em identificar os fatores que estimulam ou inibem a adoção do *low code*, sem a intenção de comparar diretamente as plataformas.

Assim como em Bastos (2023), o presente trabalho, utiliza as plataformas para construir uma aplicação com base em requisitos elicitados pelo autor, visando compará-las ao final. As principais diferenças entre os dois estudos são a forma de seleção das plataformas, o foco e os requisitos da aplicação desenvolvida.

Por conseguinte, no Quadro 2, estão listadas as características extraídas dos estudos anteriores e sua relação com o projeto apresentado neste trabalho. Os critérios para essa análise estão organizados da seguinte forma: (i) quantidade de plataformas de desenvolvimento comparadas; (ii) métodos de busca das plataformas; (iii) definição da lista de plataformas selecionadas; (iv) tipos de requisitos especificados); (v) definição dos critérios de avaliação.

Quadro 2 – Comparação entre os trabalhos relacionados e o proposto.

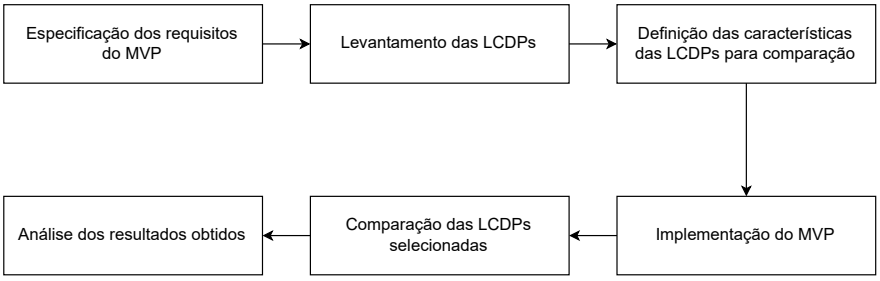
	Sahay *et al.* (2020)	**Bastos (2023)**	**Käss *et al.* (2022)**	**Trabalho proposto**
Quantidade de plataformas de desenvolvimento comparadas	*Appian, Google App Maker, Kissflow, Mendix, MS PowerApp, OutSystems, Salesforce App Cloud* e *Zoho Creator*	Íris *Low Code* Certidão, *Mendix*, *Outsystems* e *Zoho Creator*	-	*AppSheet, Adalo, Bubble, FlutterFlow* e *Outsystems*
Métodos de busca das plataformas	-	-	Pesquisa por palavras chaves em bancos bibliográficos e serviços de indexação	Pesquisa por palavras chaves em ferramentas de pesquisa online
Definição da lista de plataformas selecionadas	Relatório do Gartner e do Forrester	Relatório do Gartner e o Íris *Low-code* Certidão	Não possui lista de plataformas	Plataformas que atendam aos requisitos estabelecidos
Tipos de requisitos especificados	Funcionais	Funcionais	-	Funcionais e não funcionais
Definição dos critérios de avaliação	Proposta pelo autor após utilizar as plataformas	Proposta pelo autor após utilizar as plataformas	Revisão bibliográfica	Proposta pelo autor após utilizar as plataformas

Fonte: Elaborado pelo autor.

4 PROCEDIMENTOS METODOLÓGICOS

Este capítulo apresenta os procedimentos para a realização deste trabalho e os seguintes passos para sua execução, expostos na Figura 2: (i): Especificação dos requisitos funcionais e não funcionais do MVP; (ii): Levantamento das LCDPs; (iii): Definição das características das LCDPs para a comparação; (iv): Implementação do MVP; (v): Comparação das LCDPs selecionadas; (vi): Análise dos resultados obtidos.

Figura 2 – Fluxo das atividades que serão executadas



Fonte: Elaborado pelo autor.

4.1 Especificação dos requisitos do MVP

Nesta etapa, o foco é detalhar os requisitos funcionais e não funcionais do MVP. É essencial garantir que o escopo seja bem definido para evitar a inclusão de requisitos excessivos que comprometam o cronograma. O principal objetivo é utilizar o MVP como uma ferramenta prática para a avaliação comparativa de diferentes plataformas, inserindo-o em um contexto de uso real para analisar seu desempenho e usabilidade em relação às alternativas existentes.

4.2 Levantamento das LCDPs

O segundo procedimento consiste em pesquisar as plataformas no buscador de *internet Google*, com navegação anônima e em um navegador instalado, especificamente para isso, utilizando as palavras-chave: "*No Code Development Platform*"; "*Low Code Development Platform*".

4.3 Definição das características das LCDPs para a comparação

Este procedimento consiste em definir as características que serão consideradas para a comparação das plataformas. Para isso, é importante observar os inibidores apontados por Käss *et al.* (2022), como as limitações de funcionalidades, i aprisionamento a um provedor e a falta de flexibilidade e customização. Além disso, é necessário utilizar as plataformas para identificar novas possíveis características que possam ser comparadas.

4.4 Implementação do MVP

O objetivo deste procedimento é iniciar o desenvolvimento do MVP em cada uma das plataformas, conforme os requisitos estabelecidos na Seção 4.1. A implementação deste é uma etapa crucial para validar as principais funcionalidades e limitações da LCDP. Durante este processo, é fundamental observar e registrar uma série de fatores para garantir uma integração eficaz e a entrega de um produto funcional.

4.4.1 Dificuldades encontradas

Durante o desenvolvimento do MVP, é possível encontrar diversos contratempos que podem impactar o progresso do projeto. Entre os problemas comuns, pode-se destacar a interface da plataforma que, se não for intuitiva, pode dificultar a navegação e o uso adequado das ferramentas, resultando em atrasos e uma curva de aprendizado mais acentuada. Além disso, podem surgir problemas ao utilizar funcionalidades específicas, como incompatibilidades ou limitações inesperadas que podem exigir ajustes e soluções alternativas. É crucial armazenar e documentar todas essas dificuldades, juntamente com os detalhes sobre como foram resolvidas, para referência futura e para melhorar o processo de desenvolvimento.

4.4.2 Possíveis limitações da plataforma

Cada plataforma pode apresentar limitações que afetam a capacidade de atender completamente aos requisitos do projeto. Algumas podem não oferecer todos os recursos necessários, como suporte para funcionalidades avançadas ou integrações específicas, o que pode exigir a adoção de soluções alternativas ou o uso de serviços adicionais pagos. Além disso, podem ter restrições na personalização da aplicação, limitando a capacidade de atender a requisitos

específicos de design ou funcionalidade. É essencial avaliar essas limitações cuidadosamente para garantir que a plataforma escolhida suporte os requisitos do projeto e considerar soluções alternativas quando necessário.

4.5 Comparação das LCDPs selecionadas

Depois das considerações feitas na Seção 4.4, é feita a comparação entre as plataformas escolhidas na Seção 4.2, com base nos critérios definidos na Seção 4.3. O objetivo é examinar como os diferentes fatores impactaram o desenvolvimento e avaliar as características relevantes das plataformas. Esta etapa visa consolidar as informações para fornecer uma visão clara e detalhada de cada plataforma.

4.6 Análise dos resultados obtidos

Após realizar a comparação entre as LCDPs na Seção 4.5, procede-se com a análise dos resultados, destacando os pontos positivos e negativos de cada plataforma. Esta análise envolve a identificação das dificuldades encontradas e das funcionalidades que facilitaram o desenvolvimento.

5 RESULTADOS

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

5.1 Definição dos requisitos do MVP

Na definição dos requisitos do MVP, optou-se pela implementação de um sistema de controle financeiro, cuja função é gerenciar os gastos e despesas mensais dos usuários. Essa escolha foi feita devido à simplicidade na produção e ao fato de que o sistema oferece um nível básico de funcionalidade, incluindo acesso de usuários, relacionamento entre entidades no banco de dados e suporte a diferentes tipos de dados.

A seguir, estão listados todos os requisitos funcionais e não funcionais da aplicação, que foram escolhidos considerando, também, dois inibidores encontrados por Käss *et al.* (2022): as limitações das funcionalidades e a falta de flexibilidade e customização.

5.1.1 Requisitos funcionais

- **RF01 Manter usuário:** O sistema permitirá o cadastro e a autenticação do usuário.
- **RF02 Manter transações:** O sistema permitirá a adição, edição e exclusão de transações financeiras (receitas e despesas).
- **RF03 Visualizar transações:** O sistema permitirá a visualização de uma lista de transações financeiras.
- **RF04 Visualizar saldo:** O sistema permitirá a visualização do saldo total do usuário.
- **RF05 Visualizar balanço mensal:** O sistema permitirá a visualização do balanço mensal, considerando apenas as despesas e receitas do respectivo mês.
- **RF06 Visualizar total de receitas e despesas mensais:** O sistema permitirá a visualização do total de receitas e despesas do usuário do respectivo mês.

5.1.2 Requisitos não funcionais

1. **RNF01 Responsividade:** O sistema deve ser capaz de se adaptar às diversas dimensões de tela, sem que isso prejudique quaisquer funcionalidades.

5.2 Levantamento das plataformas de desenvolvimento *low code*

Para executar a pesquisa das plataformas, foi feita uma busca como dito na Seção 4.2, os resultados desta foram sites que tratavam do assunto como "listas de plataformas low-code indicadas para tentar em 2023" ou "as melhores plataformas low code de 2023". Os parâmetros de pesquisa que devem ser considerados neste trabalho são:

- Possuir plano de uso gratuito;
- Conseguir desenvolver aplicações *web*;
- Manipulação do esquema do banco de dados.

Vale ressaltar, que o terceiro ponto refere-se à flexibilidade ou rigidez na manipulação do esquema do banco de dados. Isso inclui a capacidade de ajustar a estrutura das tabelas, modificar os tipos de dados dos campos, definir ou alterar relacionamentos entre tabelas. O total de plataformas qualificadas inicialmente foi de 27, já que as mesmas apontavam ser promissoras no desenvolvimento de aplicações *web*. O Quadro 3 apresenta o resultado da pesquisa em contrates com os parâmetros.

Ao analisar o primeiro parâmetro da pesquisa, possuir plano gratuito, ocorreu uma redução de 10 plataformas, sete disponibilizavam apenas um período de teste (*Appy Pie, Appery.io, Pixpa, DronaHQ, Kintone, Quickbase* e *Skuid*) e três ofereciam apenas plano pago, com a opção de solicitar uma demonstração (*Kissflow*, *Appian* e *Betty Blocks*).

Já no segundo parâmetro, descartou-se mais quatro, o *Mockplus* por ser voltado apenas para prototipação (*UI/UX*), o *Peaka* por ser de integração de dados e as outras por focarem apenas em aplicativos *mobile* (*Crowbotics*, *Thunkable*).

Por último, o parâmetro de manipulação do esquema do banco de dados, removeu mais cinco: *Wix*, *Carrd*, *Wordpress*, *PageCloud* e *HubSpotCMS*. No caso específico do *Wordpress*, só é possível manipular o banco de dados através de *plugins*, esses por sua vez, só estão disponíveis em planos pagos. Vale ressaltar, também, que todas que foram descartadas pelo terceiro parâmetro, são plataformas de sistema de gestão de conteúdos (*CMS*). Um Sistema de Gestão de Conteúdo (CMS) é uma aplicação *Web* especializada que permite a criação e administração de conteúdo digital de forma simplificada e acessível. Por outro lado, uma aplicação *Web* refere-se a uma categoria abrangente de software acessado através de um navegador, com o propósito de realizar tarefas específicas e oferecer serviços interativos aos usuários. Embora um CMS possa ser classificado como uma aplicação *Web*, é importante notar que nem toda aplicação *Web* se enquadra na definição de um CMS.

Quadro 3 – Relação entre lista de plataformas e parâmetros de pesquisa

Plataforma	Planos	Tipo de desenvolvimento	Manipulação do esquema do banco de dados	Modelos de implementação
Carrd	Gratuito, período de teste (7 dias) e pago	Web	Não utiliza banco de dados	*Cloud*
FlutterFlow	Gratuito, período de teste (14 dias) e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*On-premise* e *cloud*
Bubble.io	Gratuito, período de teste (14 dias) e pago	*Web*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Wix	Gratuito, período de teste (14 dias) e pago	*Web e mobile*	Depende do banco conectado	*Cloud*
Zoho Creator	Gratuito, período de teste (15 dias) e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Mockplus	Gratuito, período de teste (15 dias) e pago	Prototipagem (UI/UX)	Não utiliza banco de dados	*Cloud*
Peaka	Gratuito, período de teste (30 dias) e pago	Integração de dados	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Outsystems	Gratuito e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*On-premise* e *cloud*
Adalo	Gratuito e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*On-premise* e *cloud*
Webflow	Gratuito e pago	*Web*	Criar e manipular coleções e atributos	*Cloud*
Mendix	Gratuito e pago	*Web e mobile*	Depende do banco conectado	*On-premise* e *cloud*
AppSheet	Gratuito e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
WordPress	Gratuito e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
PageCloud	Gratuito e pago	*Web*	Não utiliza banco de dados	*Cloud*
HubSpot CMS	Gratuito e pago	*Web*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Crowbotics	Gratuito e pago	Mobile	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Thunkable	Gratuito e pago	Mobile	Criar e manipular tabelas e atributos	*Cloud*
Appy Pie	Período de teste (7 dias) e pago	*Web e mobile*	Depende do banco conectado	*Cloud*
Appery.io	Período de teste (14 dias) e pago	*Web e mobile*	Criar e manipular coleções e construir relações (1:N) usando *pointers*	*Cloud*
Pixpa	Período de teste (15 dias) e pago	*Web*	Não utiliza banco de dados	*Cloud*
DronaHQ	Período de teste (30 dias) e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Kintone	Período de teste (30 dias) e pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Quickbase	Período de teste (30 dias) e pago	*Web*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Skuid	Período de teste (15 dias)	Mobile	Criar e manipular tabelas (objetos) e atributos (campos), construir relações (1:N)	*On-premise* e *cloud*
Kissflow	Pago	Automação de fluxo de trabalho	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Appian	Pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*
Betty blocks	Pago	*Web e mobile*	Criar e manipular tabelas e atributos, construir relações (1:N, N:N)	*Cloud*

Fonte: Elaborado pelo autor.

Embora oito plataformas tenham atendido a todos os parâmetros, três não foram continuadas como objeto de estudo. A primeira foi o *Webflow*, que se trata de um *CMS*, a limitação de só permitir o uso dos dados de formulários conforme o tipo definido na página e as entidades do banco de dados tornou a personalização complexa. Semelhantemente, o *Zoho Creator*, focado no desenvolvimento de sistemas internos como dashboards e ferramentas de gerenciamento, também apresentou dificuldades na criação de interfaces personalizadas, sendo muito baseado em formulários vinculados ao banco de dados. Já o *Mendix*, é voltado ao profissional que já trabalha na área de tecnologia, então ela acaba perdendo um pouco uma das

características principais do *no code*, sendo de ter interfaces simples, intuitivas e que acelerem o desenvolvimento, até mesmo para fazer uma página de autenticação é preciso muito esforço.

Portanto, restaram apenas cinco plataformas para desenvolver o MVP: *Adalo*, *Appsheet*, *Bubble*, *FlutterFlow* e *OutSystems*.As únicas plataformas que disponibilizaram a versão utilizada foram o *Bubble*, que está na versão 29, e o *Outsystems*, que está na versão 11. Já as plataformas *Adalo*, *AppSheet* e *FlutterFlow* não revelaram a versão, porém a data de uso foi agosto de 2024.

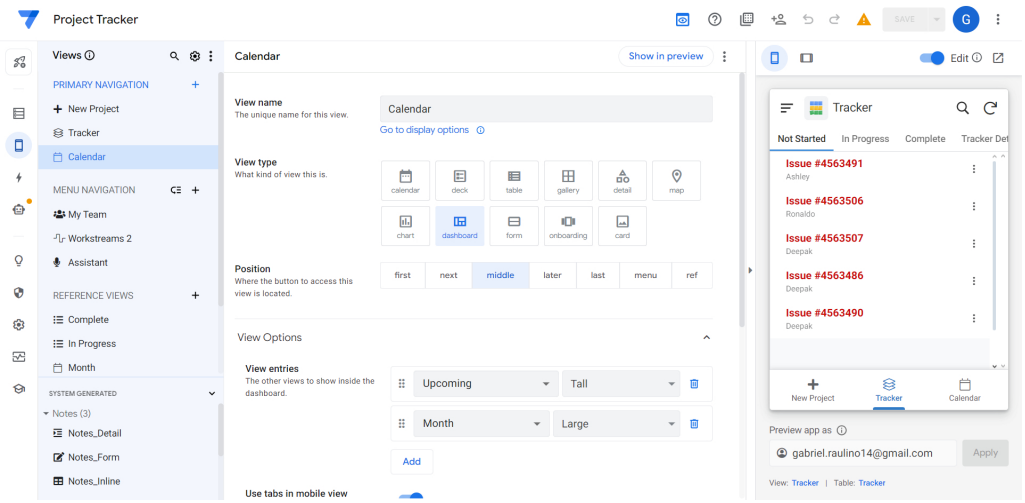
5.3 Plataformas Selecionadas

Nesta seção, são apresentadas breves descrições das LCDPs selecionadas na Seção 5.2. Todas essas plataformas permitem ir além da criação de protótipos ou MVPs, fornecendo um conjunto robusto de ferramentas que viabilizam o desenvolvimento de soluções completas e escaláveis.

5.3.1 AppSheet

O *AppSheet* é uma plataforma desenvolvida pelo *Google* que permite criar aplicativos *web* e *mobile* sem a necessidade de programação. A plataforma se destaca pela integração com diversas fontes de dados, como *Google* Planilhas, *Excel* e *SQL*, facilitando o desenvolvimento rápido de soluções empresariais completas. Além disso, ela oferece uma interface intuitiva e recursos avançados, como automações e IA embutida, oferecendo suporte para soluções empresariais em larga escala. A Figura 3 ilustra um exemplo de interface do *AppSheet*

Figura 3 – Exemplo de interface do *AppSheet*

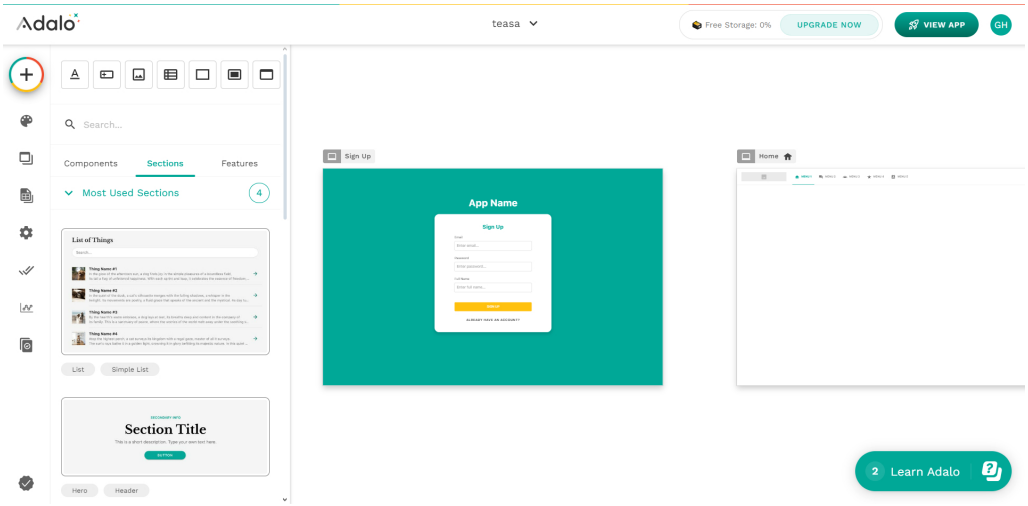


Fonte: Elaborada pelo autor.

5.3.2 *Adalo*

O *Adalo* permite a criação de aplicativos *web* e *mobile* com foco na simplicidade, utilizando um editor visual de arrastar e soltar. Sua versatilidade pode ir desde prototipagem até o lançamento de aplicativos completos, com integrações com serviços como *Stripe* e *Zapier*, permitindo a criação de produtos viáveis para o mercado. A Figura 4 mostra um exemplo de interface do *Adalo*

Figura 4 – Exemplo de interface do *Adalo*



Fonte: Elaborada pelo autor.

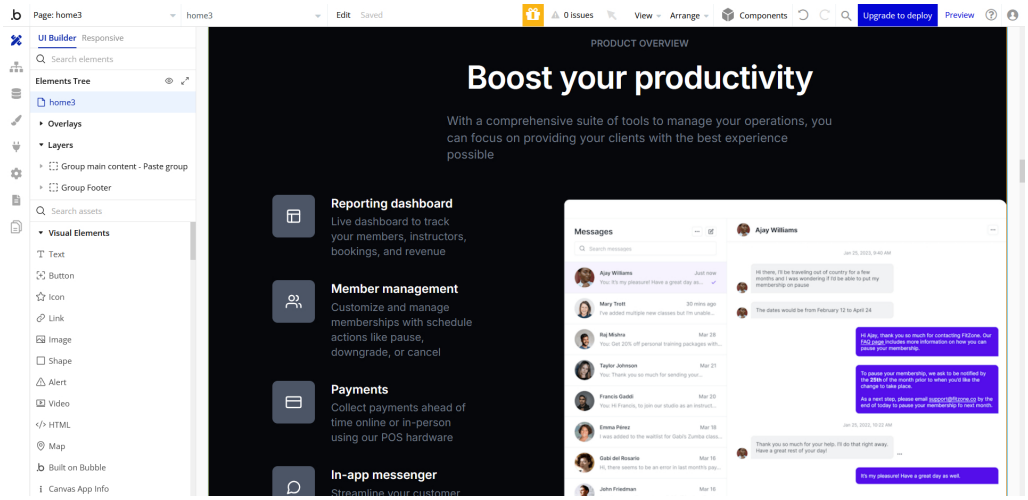
5.3.3 *Bubble*

O *Bubble* permite a criação de aplicativos *web* robustos, com foco em flexibilidade e escalabilidade. Oferece controle total sobre a lógica de negócios e a base de dados, permitindo que a criação de aplicações personalizadas. Além disso, se diferencia por suas capacidades avançadas de integração com APIs e personalização, permitindo o desenvolvimento de aplicativos sofisticados e escaláveis. A Figura 5 ilustra um exemplo de interface do *Bubble*

5.3.4 *FlutterFlow*

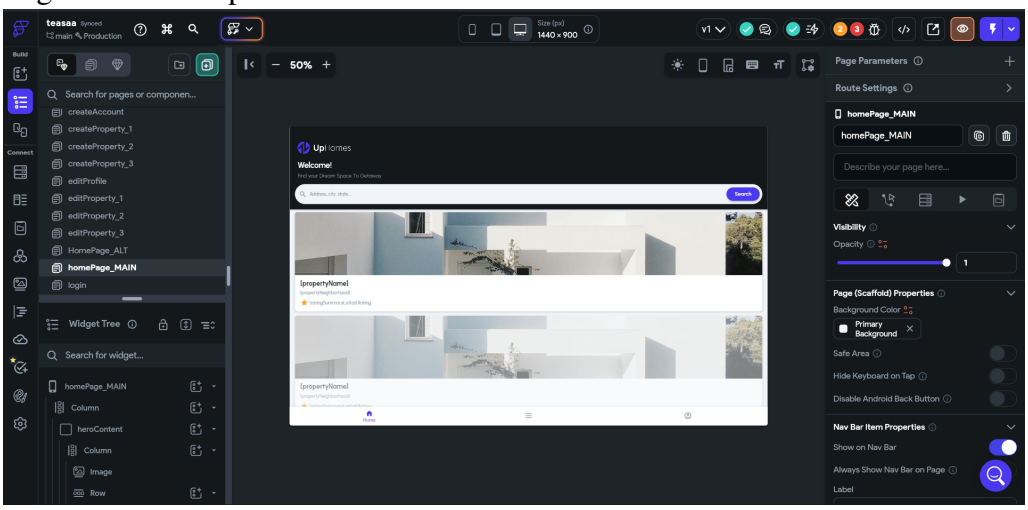
O *FlutterFlow* é uma plataforma baseada no *Flutter*, *framework* da *Google* para desenvolvimento de aplicativos *mobile* e *web*. A ferramenta possibilita a criação de interfaces de usuário atraentes com funcionalidades nativas do *Flutter*, como animações e integração com *Firebase*. A Figura 6 mostra um exemplo de interface do *FlutterFlow*

Figura 5 – Exemplo de interface do *Bubble*



Fonte: Elaborada pelo autor.

Figura 6 – Exemplo de interface do *FlutterFlow*



Fonte: Elaborada pelo autor.

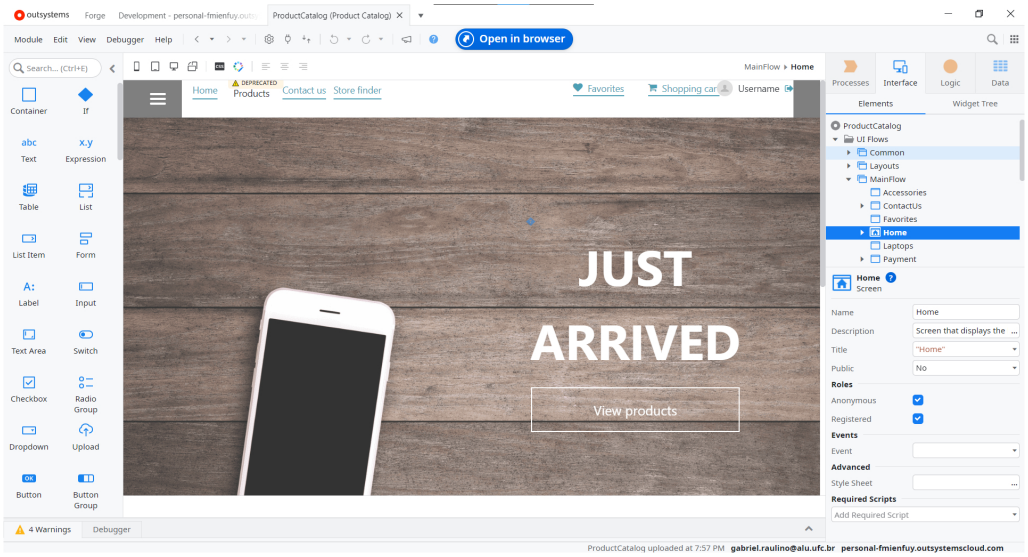
5.3.5 Outsystems

O *Outsystems* é uma plataforma empresarial focada em soluções robustas e escaláveis, com um extenso conjunto de ferramentas para desenvolvimento ágil de aplicativos *web* e *mobile*. A plataforma se destaca por suas capacidades de integração com sistemas legados e *APIs*, além de oferecer suporte a arquiteturas empresariais complexas, como grandes volumes de dados e fluxos de trabalho personalizados. A Figura 7 ilustra um exemplo de interface do *Outsystems*

5.4 Critérios de comparação

Esta seção apresenta todas as características que serão consideradas para comparação das LCDPs. Para melhor organização, tais atributos foram separados em dois grupos, os critérios

Figura 7 – Exemplo de interface do *Outsystems*



Fonte: Elaborada pelo autor.

quantitativos e os qualitativos:

5.4.1 Critérios Quantitativos

Esta categoria inclui atributos que podem ser mensurados numericamente, como o número de bancos de dados suportados e a quantidade de recursos nativos de interface. Para este trabalho, são considerados recursos nativos todos os elementos, componentes e *templates* disponíveis por padrão na plataforma. Esses fatores são importantes para avaliar a eficiência e o potencial de cada plataforma no desenvolvimento.

Bancos de dados suportados

Para este fator, consideram-se todos os tipos de bancos de dados que a plataforma consegue integrar e gerenciar, abrangendo tanto os relacionais quanto os não-relacionais. A análise da compatibilidade da plataforma com esses diferentes tipos é essencial, pois a escolha adequada impacta diretamente na eficiência e na escalabilidade da aplicação desenvolvida. Portanto, a capacidade de suportar uma ampla variedade de bancos de dados é crucial para avaliar a versatilidade e a adaptabilidade da plataforma às diversas necessidades do projeto.

Recursos nativos de interface

Neste critério, a avaliação se concentra nos recursos nativos que a plataforma oferece para a construção de interfaces de usuário, abrangendo elementos como botões, campos de

entrada de texto, caixas de seleção e menus *dropdown*; além de componentes como barras de navegação, rodapés, painéis laterais e formulários; e até mesmo páginas completas como cadastro, login, perfil e relatórios, que já vêm prontas para uso ou exigem mínima customização, o que pode acelerar significativamente o desenvolvimento de projetos.

É importante destacar a diferença entre elementos e componentes da interface: elementos são os blocos mais básicos de construção usados para interações específicas, como um botão ou um campo de texto. Eles são essenciais para criar a interface e capturar interações do usuário. Por outro lado, componentes são estruturas mais complexas, compostas por múltiplos elementos agrupados para desempenhar funções mais amplas, como uma barra de navegação ou um formulário completo, que já integram diversas interações e fluxos lógicos.

Em plataformas *low code*, a disponibilidade destes, facilita o desenvolvimento ao permitir a ágil construção de interfaces complexas, com funcionalidades pré-configuradas, aumentando a produtividade, a padronização do projeto e reduz o retrabalho.

Assim, ao avaliar esses recursos de interface, é possível entender o grau de maturidade da plataforma e como ela otimiza o ciclo de desenvolvimento com base nos elementos e componentes nativos que oferece.

5.4.2 Critérios Qualitativos

Este grupo abrange atributos mais subjetivos, que não podem ser expressos numericamente, mas que têm ainda assim um impacto significativo na comparação das plataformas selecionadas.

Capacidade de Customização

Para este critério, é necessário avaliar a capacidade de customização da página que está sendo desenvolvida e dos elementos utilizados. Esse aspecto é crucial por influenciar diretamente a flexibilidade no design e na funcionalidade dos aplicativos, garantindo que o produto final atenda às necessidades específicas de cada projeto. Avaliar os recursos de customização permite medir o quanto a plataforma oferece controle sobre aspectos visuais e interativos, sem comprometer a simplicidade e a velocidade de desenvolvimento, características essenciais nessas ferramentas. Os critérios levantados para este aspecto incluem:

- Tipos de *layouts* suportados;
- Nível de controle sobre o espaçamento entre elementos;

– Variedade de opções de posicionamento dos elementos.

Esses critérios abrangem a avaliação de como são gerenciados posicionamento de elementos na interface, diferentes configurações de layout e espaçamento. A flexibilidade desses aspectos permite que os desenvolvedores adaptem a interface às necessidades do projeto, garantindo que o design seja responsivo e atenda às exigências visuais e funcionais, sem exigir habilidades avançadas de codificação.

Capacidade de Migração dos Dados

Este atributo avalia a eficácia e a flexibilidade da plataforma em termos de migração de dados, tanto na própria plataforma quanto para sistemas externos. A capacidade de migração de dados é fundamental para garantir que as informações possam ser transferidas, convertidas e integradas de maneira eficiente entre diferentes ambientes ou formatos. Isso inclui não apenas a transferência de dados entre diferentes contas ou ambientes na mesma plataforma, mas também a integração com sistemas externos e a adaptação dos dados a novos formatos quando necessário.

A importância da migração de dados reside na sua capacidade de suportar a continuidade dos negócios e a evolução dos sistemas sem perda de informações ou funcionalidades. Uma plataforma que oferece uma migração de dados eficaz permite que as organizações mantenham a integridade e a acessibilidade dos dados ao longo do tempo, adaptando-se às mudanças de requisitos ou de tecnologia sem enfrentar desafios significativos. As métricas que serão avaliadas são estas:

– Formatos de exportação suportados;
– Formatos de importação suportados;

Esses critérios avaliam a versatilidade da plataforma em lidar com diferentes tipos de dados e formatos, o que é essencial para garantir a flexibilidade na integração com outros sistemas e a continuidade dos processos de negócios. Avaliar essas métricas ajuda a entender como a plataforma pode acomodar mudanças e assegurar que os dados possam ser transferidos e utilizados eficazmente em diversos contextos.

Capacidade de Migração da Aplicação

Neste atributo, é avaliada a capacidade de migrar os aplicativos gerados na plataforma. Esse aspecto abrange não apenas a facilidade de transferir um aplicativo para outras contas dentro da mesma plataforma, como também a possibilidade de migrar para outras completamente

diferentes.

A migração interna pode ser necessária em situações como, a troca de proprietário do projeto, mudanças na estrutura organizacional ou até mesmo a necessidade de segmentação de ambientes de produção e de desenvolvimento. Já a migração externa, envolve um processo mais complexo, que requer compatibilidade entre diferentes tecnologias e formatos de dados, além de uma adaptação eficiente do código e das funcionalidades existentes.

A capacidade de migrar aplicações sem perda de informações ou funcionalidades é um fator crítico para garantir a flexibilidade e a longevidade dos projetos, especialmente, em ambientes corporativos dinâmicos, onde a mudança de provedores ou a adoção de novas tecnologias pode ser frequente. Avaliar a facilidade e as limitações dessa migração é essencial para assegurar que os aplicativos possam evoluir conforme as necessidades do negócio e não fiquem presos a um único ecossistema. A lista abaixo apresenta as métricas considerados para este critério:

- Tipo de exportação do aplicativo (interna e externa);
- Capacidade de exportar e transferir fluxos de trabalho automatizados, regras de negócio e lógica sem a necessidade de desenvolvimento manual;
- Compatibilidade com plataformas externas.

Suporte à Colaboração

Neste fator, é avaliada a possibilidade de colaboração entre desenvolvedores em um mesmo projeto, seja através do uso de uma única conta que permita múltiplos acessos simultâneos ou do compartilhamento do projeto entre contas diferentes.

O suporte à colaboração é fundamental para garantir que equipes de desenvolvimento possam trabalhar de forma eficiente e sincronizada, especialmente em projetos complexos ou em larga escala. A colaboração eficaz não apenas acelera o desenvolvimento, mas também melhora a qualidade do projeto, promovendo a integração de diferentes habilidades e conhecimentos. Os aspectos críticos a serem avaliados incluem:

- Compartilhamento de projetos entre contas;
- Permite edição simultânea em tempo real;
- Possui ferramentas de versionamento e controle de mudanças;

A capacidade de colaborar efetivamente é essencial para a agilidade e sucesso de projetos de desenvolvimento, permitindo que equipes coordenem seus esforços e integrem suas

contribuições eficientemente.

5.5 Desenvolvimento do MVP

Esta seção detalha as facilidades, dificuldades e limitações encontradas no desenvolvimento do MVP em cada um das plataformas. O resultado das interfaces, tanto para *desktop* quanto para *mobile*, pode ser visto no Apêndice A

5.5.1 Facilidades encontradas

O *Adalo* simplificou significativamente a criação de novas despesas e receitas. O componente de formulário requer a seleção do tipo de coleção a ser modificada, gerando automaticamente os campos e preenchendo algumas informações. Além disso, quando a tabela não contém dados, é possível exibir uma mensagem informativa para o usuário sobre o estado vazio. Em relação à exibição de valores numéricos, o *Adalo* oferece suporte para múltiplas moedas.

O *Bubble* forneceu muitas opções para customizar a disposição dos elementos, como: a capacidade de ajustar o *layout* de forma flexível, definir margens e alinhamentos precisos, e criar estilos personalizados para os componentes. Além disso, permite a criação de elementos como *dropdowns* com o conteúdo desejado, sem a necessidade de utilizar tabelas do banco de dados. O desenvolvimento tornou-se mais eficiente devido à possibilidade de criar estados para os componentes, facilitando a manipulação, validação e personalização das ações.

O *FlutterFlow*, por possibilitar codificar e ainda por cima ter um *code copilot*, facilitou a construção de comportamentos customizados como alterar a data de um componente que recebe um estado do aplicativo. Além disso, o interessante dessas funções é que elas possuem testes unitários, diminuindo assim a probabilidade de *bugs* e caso ocorram, sejam facilmente detectados. Outro fato interessante sobre ele, é que existem diversas formas de lidar com a responsividade dos elementos e até mesmo o posicionamento deles consoante as dimensões da tela.

O *Outsystems* proveu alguns elementos e componentes nativos que auxiliaram na construção das funcionalidades, entre eles: *Month picker* (para seleção do mês), Tabela (com a possibilidade de adicionar colunas customizadas, sem necessariamente ser parte de uma coleção) e o *popup*. Existem muitas funções para lidar com os dados, por exemplo, caso tenha uma

variável com uma lista de uma entidade, pode-se realizar filtrar, adicionar, remover ou limpar a lista.

No *Appsheet*, ao selecionar o tipo de dado a ser exibido em uma *view* (que define como os dados são apresentados, como tabelas, formulários, ou gráficos), o sistema gera automaticamente as funcionalidades de CRUD (Criar, Ler, Atualizar e Excluir) correspondentes. A tabela gerada inclui, por padrão, opções para ordenação e permite a personalização dos nomes e tipos das colunas, incluindo uma ampla gama de opções para formatação de moedas. Para iniciar um projeto, é possível descrever o objetivo e a estrutura do aplicativo para o *Gemini*, a inteligência artificial do *Google* integrada à plataforma. O *Gemini* apresentará as tabelas do banco de dados, oferecendo a opção de editá-las e, em seguida, criar o aplicativo com alguns dados já preenchidos.

5.5.2 Dificuldades encontradas

A responsividade no *Adalo* apresenta limitações, já que a ausência de ferramentas como *grid* ou *flex-box* dificulta o ajuste no crescimento e distanciamento entre elementos agrupados. Também, não é possível colocar *padding*, *margin* ou *gap* nos componentes. Ainda, no banco de dados, não se consegue criar *Enums*, então para o caso do tipo de categoria, para definir se é receita ou despesa, foi criado o atributo *boolean* com o nome "Receita", caso o valor seja falso, é uma "despesa", do contrário "receita".

Quanto ao componente de *dropdown* dessa plataforma, só consegue mostrar opções de uma coleção, ou seja, não é possível colocar opções textuais manualmente, também não se conseguiu atribuir um valor inicial padrão que não fosse vazio. Ao preencher os dados do elemento tabela, houve dificuldade para criar um intervalo entre o primeiro e o último dia do mês selecionado no elemento utilizado. Acredita-se que essa manipulação não seja viável, já que as opções oferecidas são dias, semanas, meses e anos antes ou depois da data atual, ou seja, intervalos pré-estabelecidos.

Devido à impossibilidade de tornar a fonte de dados da tabela dinâmica, conforme a seleção no *dropdown*, optou-se por criar três tabelas distintas, cada uma com seus respectivos dados, alternando a visibilidade conforme a opção selecionada. Além disso, a adição de uma coluna customizada que comportasse a edição ou deleção do registro não foi viável, pois o conteúdo exibido está restrito exclusivamente aos dados da coleção selecionada. Contudo, aproveitou-se a opção nativa de definir uma ação ao clicar na linha da tabela, para implementar

um modal que executasse as ações necessárias de edição e deleção dos registros.

No *Bubble* não foi viável lidar com a disposição dos elementos, como esperado da responsividade, para alternar o alinhamento dos elementos de coluna para linha ou vice-versa, conforme o tamanho da tela, foi necessário copiar o componente e organizá-lo de forma que fosse coerente com o comportamento esperado em telas menores, alterando a visibilidade condicionalmente. Além disso, não foi possível mudar a cor das opções do *dropdown*, apenas dar para alterar o fundo do elemento todo, então, se quiser deixar a lista de opções de uma cor e o botão de outro, não é viável. Outro ponto é que, quando não há dados na tabela, ela não exibe um estado vazio para indicar a falta de conteúdo, ficando exposto somente o cabeçalho.

O *FlutterFlow* demonstrou muita lentidão na hora do desenvolvimento em alguns navegadores, como, no *Firefox* e no *Opera GX* (no *Edge* o desempenho foi normal), ocasionando até mesmo o travamento da página. Além disso, é preciso montar o projeto, ou seja, a plataforma demora de dois a três minutos para apresentar o aplicativo criado, sempre que quiser testar os fluxos e ações dos componentes, sendo que cada sessão fica aberta por 15 minutos, caso o usuário renove a mesma, precisará passar por todo o processo de montagem novamente. Na parte de autenticação, foi utilizado um *template* nativo de cadastro, o qual foi necessário fazer a tradução dos textos. A desvantagem deste, é que o componente que engloba os elementos não conseguiam fazer validações, portanto foi adicionado um formulário para tal.

Embora a documentação forneça detalhes sobre a configuração do *Firebase* para o uso do *Firestore* e do *Authentication*, o processo revelou-se um pouco desafiador. É necessário criar as coleções na plataforma exatamente como estão no banco de dados, observando que essas informações são sensíveis a maiúsculas e minúsculas. Além disso, a falta de automação no esquema do banco de dados da plataforma exige uma atualização constante em relação ao *Firestore*.

Por último, a plataforma em questão mostrou-se limitada na manipulação das listas de documentos do *Firebase* com código customizado. Por exemplo, ao solicitar uma lista de funcionários, não foi possível calcular a soma dos salários diretamente. Além disso, ao tentar filtrar uma lista de transações com base no campo categoria (que é uma referência a um documento *Firebase*), a filtragem não ofereceu a opção de verificar se um item estava contido em outra lista, tornando necessário o uso de código customizado para realizar essa operação.

No início, a interface do *Outsystems* revelou-se bastante desafiadora para entender, como navegar para os elementos e adicionar ações, mas depois foi fluindo conforme o uso. Ao

deparar com erros que quebravam a aplicação no navegador, um leigo talvez não conseguisse resolver com tanta facilidade, pois o que ocorre fica registrado ali no console do próprio navegador. No *Outsystems*, houve certa dificuldade em lidar com ações relacionadas ao usuário, como o login com credenciais, que apresentou erros mesmo quando as informações estavam corretas. A única forma de acessar a plataforma foi utilizando o ID do usuário obtido a partir das credenciais para efetuar o login. Além disso, ao tentar criar transações manualmente no banco de dados, enfrentou-se a limitação de não conseguir preencher o atributo de chave estrangeira do usuário, uma vez que esse campo estava protegido pelo sistema. Assim como no *FlutterFlow*, a obtenção direta dos dados da tabela para calcular valores como saldo, receita, despesa e balanço mensal não era viável. Foi necessário criar variáveis locais e manipulá-las com base nos dados retornados pela consulta ao banco de dados.

O *Appsheet* mostrou ser um dos menos flexíveis para customização, primeiro por só ser possível criar telas a partir de uma lista disponível na plataforma (*views*), preenchidas com uma coleção do banco ou parte dela, a única que não precisa de dados é obrigatoriamente a opção de *dashboard*, que no caso ela é para juntar outras *views*. Segundo que alguns elementos que vêm na tela não podem ser removidos ou renomeados, muito menos alterar as ações que eles executam. Por fim, como a responsividade dos componentes é gerida não é muito clara, por exemplo, um botão que aparece apenas como um ícone no modo móvel exibe um rótulo adicional quando visualizado em *desktop*. Não é evidente em que momento essa mudança ocorre ou como pode ser editada. Outro fato é que não é possível enviar parâmetros customizados entre as ações executadas.

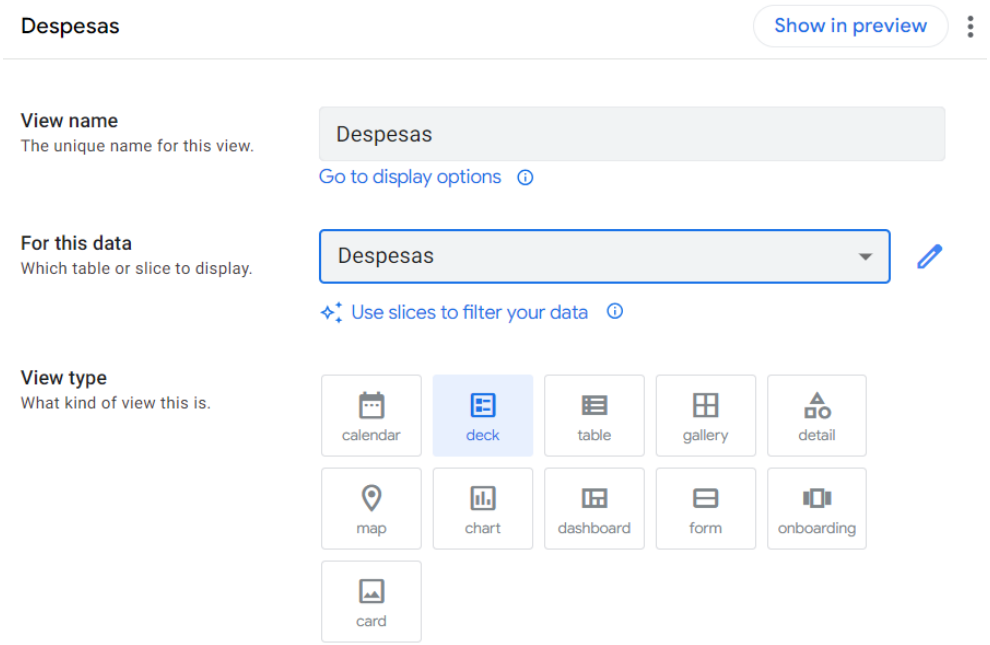
5.5.3 Limitações

No *Outsystems*, não foi possível excluir usuários cadastrados, pois essa ação parece estar protegida. Além disso, a conexão com bancos de dados externos não foi viável, uma vez que o plano gratuito não suporta integração com bancos que não estejam na nuvem, limitando assim o uso dos mesmos.

O *AppSheet* não permite implementar ações condicionais, como abrir um modal específico ao clicar no botão de adicionar transação quando a tabela exibe despesas. Outra limitação encontrada foi a incapacidade de filtrar a categoria de uma transação no momento da edição, com base no tipo (receita ou despesa). Também não foi possível remover algumas ações pré-definidas nas interfaces da *view*, o que restringiu o controle sobre as mesmas. Ainda, a

Figura 8 destaca que é obrigatório o uso de um tipo específico de dado para a *view*, impedindo assim, a criação de páginas com textos estáticos. A configuração de uma tela de login e cadastro no plano gratuito não foi viável.

Figura 8 – Configurando a *view* no *AppSheet*



Fonte: Elaborada pelo autor.

Por conta dessas limitações, a análise desta plataforma foi restrita ao seu suporte a bancos de dados, uma vez que ela não conseguiu atender às necessidades para o desenvolvimento pleno do MVP.

5.6 Comparação entre as LCDPs

Esta seção compara as plataformas selecionadas na Seção 5.2, utilizando as todas as características definidas na Seção 5.4, exceto o *AppSheet*, como especificado na Seção 5.5.3.

5.6.1 Critérios Quantitativos

Esta subseção apresenta dois critérios quantitativos fundamentais para a avaliação das plataformas low-code: o gerenciamento de bancos de dados e a disponibilidade de recursos nativos de interface. Essas subseções oferecem uma análise detalhada das capacidades de cada plataforma em lidar com dados e da variedade de elementos, componentes e *templates* que elas oferecem para o desenvolvimento de aplicativos.

Banco de dados

Esta seção examina as capacidades das plataformas em relação aos tipos e ao gerenciamento de bancos de dados. O objetivo é avaliar quais recursos e tipos cada plataforma oferece, bem como suas capacidades de conexão com diferentes bancos de dados e as limitações de armazenamento nos planos gratuitos.

O Quadro 4 apresenta um panorama das plataformas analisadas, destacando os tipos de bancos de dados suportados, a presença de bancos de dados próprios, as opções de conexão direta e as restrições de armazenamento. Essa análise é crucial para entender como cada plataforma lida com dados e quais são suas limitações e capacidades em termos de gerenciamento de banco de dados.

Quadro 4 – Informações sobre os bancos de dados das plataformas

	Bubble	**Outsystems**	**Adalo**	**FlutterFlow**	**AppSheet**
Tipos de banco de dados	Relacional	Relacional	Relacional	Não relacional	Relacional
Possui banco de dados próprio	Sim	Sim	Sim	Não	Sim
Bancos de dados escolhido	Próprio	Próprio	Próprio	Firestore	Próprio
Bancos de dados disponíveis para conexão	PostgreSQL, MySQL e Microsoft SQL	Azure SQL, MongoDB Aurora PostgresSQL, MySQL, Azure PostgresSQL, Oracle, PostgreSQL, SQL Server	-	-	SQLServer, MySQL, DynamoDB, Postgres, MariaDB, Oracle, Redshift, BigQuery.
Outras fontes de dados	Airtable, Box e Salesforce	SalesForce, SAP OData, Sharepoint, Aurora PostgresSQL, Microsoft Dataverse, iDB2, Oracle, Microsoft Dataverse, Microsoft Dynamic 365	Airtable, Google Planilhas, Xano, Firebase, Supabase e Backendless	Firebase, Supabase	Google Drive Documents, OData, Box, Oracle, Redshift, Apigee, Smartsheet, Dropbox, Microsoft Salesforce, Airtable, BigQuery, Google Planilhas, OpenAPI, External services, REST API's, Apigee connector, BigQuery connector
Limite de armazenamento no plano gratuito	200 registros	2GB	200 registros	1GB	2500 registros

Fonte: Elaborado pelo autor.

Observa-se que a maioria das plataformas analisadas utiliza bancos de dados relacionais, à exceção do *FlutterFlow*, que opta por um banco não relacional, o *Firestore*. Isso pode indicar que ela é mais indicada para aplicações que lidam com dados não estruturados ou que exigem escalabilidade. Além disso, quatro das cinco plataformas (*Bubble*, *Outsystems*, *Adalo* e *AppSheet*) oferecem seus próprios bancos de dados. Essa característica pode simplificar o desenvolvimento e otimizar o gerenciamento de dados. No entanto, o *FlutterFlow* se diferencia por não possuir um banco próprio, o que pode exigir maior atenção em relação à integração com fontes de dados externas.

No que diz respeito às opções de conexão, o *Outsystems* e o *AppSheet* destacam-se por oferecerem uma grande variedade de integrações com bancos de dados externos e outras fon-

tes de dados, como *Salesforce, SAP*, *Google* Planilhas e APIs. Isso proporciona uma flexibilidade considerável para quem deseja construir soluções mais complexas e com múltiplas fontes de dados. O *Adalo* e o *FlutterFlow*, no entanto, não possuem conexões diretas com bancos de dados, só sendo possível através de conexões por serviços de *backend*, como *Firebase* e *Supabase*, por exemplo.

Por fim, ao analisar os limites de armazenamento nos planos gratuitos, nota-se que o *Bubble* e o *Adalo* oferecem uma capacidade inicial mais limitada (200 registros), enquanto plataformas como *Outsystems* e *FlutterFlow* proporcionam mais espaço, tornando-se opções mais interessantes para projetos que demandam um maior volume de dados logo no início. O *AppSheet*, por exemplo, eleva significativamente esse limite, permitindo até 2500 registros.

Recursos nativos

Como descrito na Seção 5.4.1, foram contabilizados todos os recursos nativos de interface (elementos, componentes e *templates*) disponíveis nas plataformas, essa ação foi feita utilizando a ferramenta em si e consultando a documentação da mesma. Vale ressaltar que elementos são as unidades básicas e atômicas usadas para construir a interface do usuário, enquanto componentes são unidades mais complexas que podem incluir vários elementos e encapsular lógica e funcionalidades mais abrangentes. Os *templates* são páginas completas, como, por exemplo, uma página de perfil do usuário ou de registro de um cliente, que possuem vários componentes (podendo ou não ser algum dos disponíveis na plataforma). O Quadro 5, lista a quantidade de recursos encontrados de cada plataforma.

Quadro 5 – Quantidade de recursos nativos de interface em cada plataforma

	Elementos nativos	**Componentes nativos**	**Modelos de páginas**
Bubble	27	29	2
Outsystems	120	0	38
Adalo	41	201	30
FlutterFlow	80	173	20

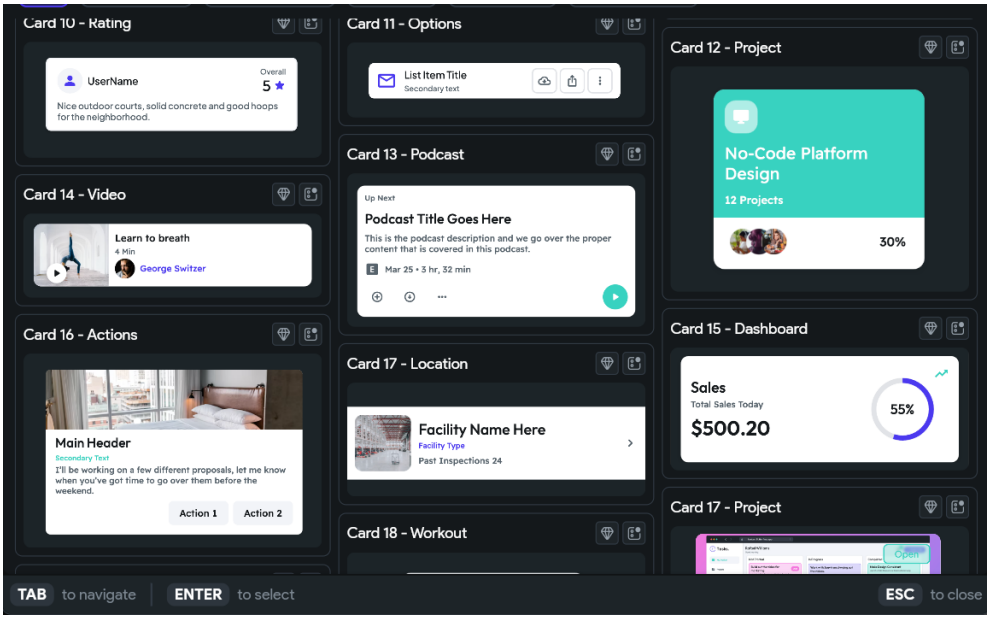
Fonte: Elaborado pelo autor.

Embora o *Outsystems* e o *Bubble* possuam um número consideravelmente menor de componentes nativos em comparação com o *Adalo* e o *FlutterFlow*, eles se diferenciam por permitir que a comunidade contribua com a criação de *plugins* e módulos próprios. No caso

do *Outsystems*, existe a biblioteca *Forge*[1], que disponibiliza uma vasta coleção de mais de sete mil recursos, incluindo componentes reutilizáveis, módulos, integrações e temas, facilitando e acelerando o desenvolvimento de aplicativos.

Por outro lado, o *Adalo* e o *FlutterFlow* se destacam pela ampla quantidade e diversidade de componentes nativos disponíveis. Vale ressaltar que diferentes variações de um mesmo componente são contabilizadas separadamente. Por exemplo, a Figura 9 ilustra que no *FlutterFlow*, existe uma diversidade de componentes do tipo *card* mas são contados como componentes distintos.

Figura 9 – Lista de componentes nativos do tipo *card* no *FlutterFlow*



Fonte: Elaborada pelo autor.

Já no *Adalo*, é possível encontrar listas detalhadas e simples, além de estados vazios (usados em casos como uma busca sem resultados), refletindo a flexibilidade e a variedade de personalizações oferecidas por essas plataformas.

5.6.2 Critérios Qualitativos

Esta subseção analisa diferentes aspectos qualitativos das plataformas, com foco em três critérios principais: customização visual, capacidade de migração de aplicativos e de dados. Cada subseção explora as possibilidades e limitações oferecidas por cada plataforma, abordando os desafios técnicos e funcionais envolvidos em cada um desses processos.

[1] https://www.outsystems.com/forge/

Capacidade de Customização

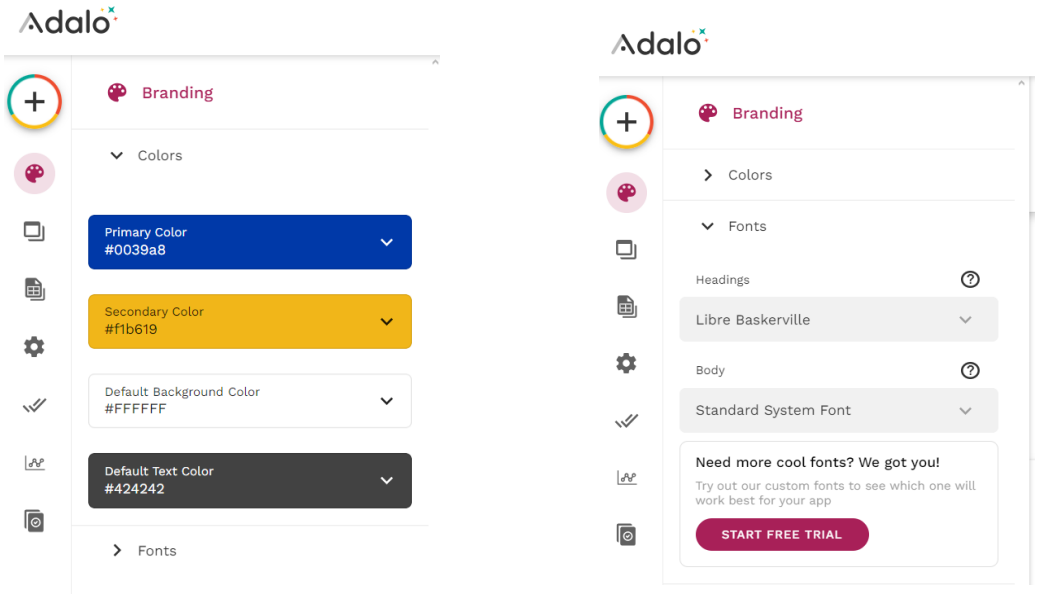
Esta seção apresenta as opções de customização visuais oferecidas por cada plataforma. A análise abrange ajustes de propriedades visuais dos elementos, como cores, fontes, bordas e dimensões, além da adaptação a diferentes dispositivos. Destacam-se as flexibilidades e limitações de cada plataforma no que se refere à personalização da interface e à responsividade visual.

Adalo

A plataforma permite customizar diversas propriedades visuais dos elementos da aplicação, como altura, largura, cor e imagem de fundo, estilo das bordas (incluindo espessura e arredondamento), opacidade e sombra. Também é possível ajustar o comportamento visual em diferentes dispositivos, garantindo que o componente se adapte a telas de *mobile, tablet* e *desktop*. Além disso, permite definir a visibilidade dos elementos com base no dispositivo utilizado.

A Figura 10 demonstra ser possível escolher as fontes para cabeçalhos e corpo do texto. No entanto, não é possível criar novas variáveis de cores, e o número de fontes disponíveis é limitado no plano gratuito.

Figura 10 – Opções de estilização global no Adalo



Fonte: Elaborada pelo autor

Bubble

A plataforma permite customizar diversos aspectos visuais dos elementos da aplicação, como cor de fundo, estilo das bordas (permitindo ajustes individuais), opacidade e sombra. As opções de customização variam conforme o tipo de elemento. Por exemplo, em uma página, é possível alterar o texto exibido na aba do navegador, enquanto em uma tabela, pode-se definir a disposição dos dados.

Além disso, é possível aplicar ajustes visuais condicionais, como alterar a altura de um componente com base na largura da tela ou modificar sua visibilidade conforme necessário. A plataforma também oferece a possibilidade de personalizar elementos e salvar essas personalizações para reutilização em outros componentes, de maneira semelhante ao uso de classes no CSS. Para garantir consistência visual, podem ser definidas variáveis de fontes e cores que se aplicam a diferentes elementos da aplicação.

No que se refere ao layout, é possível controlar a organização dos componentes, ajustar margens, espaçamentos internos (*padding*), altura e largura, podendo ser definidos em px ou porcentagem. Em contêineres que contêm outros elementos, há opções de organização semelhantes ao flexbox, permitindo o alinhamento dos elementos horizontalmente (*row*) ou verticalmente (*column*), além da aplicação de espaçamento entre eles.

FlutterFlow

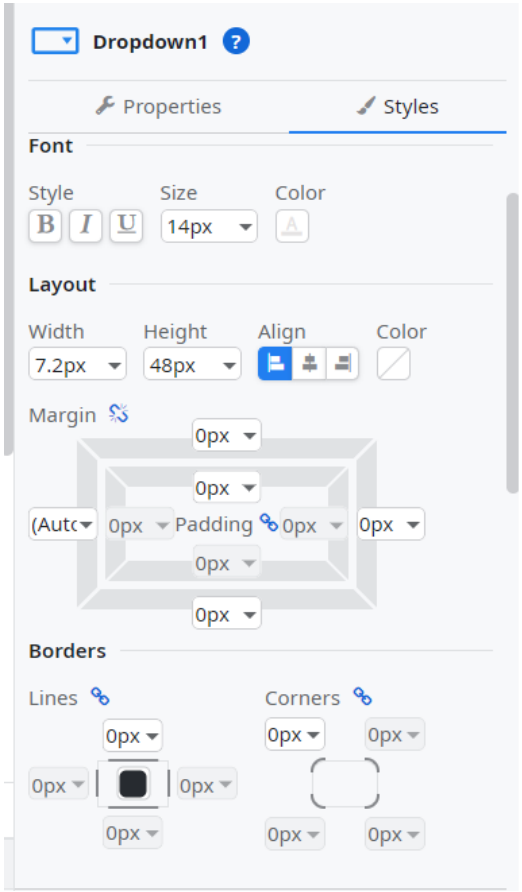
A plataforma permite a personalização de cores, disponibilizando a criação de paletas tanto para temas claros quanto escuros, além de adicionar novas cores manualmente, extraídas de imagens ou geradas por IA. Para tipografia, é possível ajustar o espaçamento entre letras, cor, família, tamanho e peso das fontes, com a opção de torná-las responsivas para diferentes dispositivos como *mobile, tablet* e *desktop*. Também é possível definir as fontes primária e secundária, além de adicionar ícones e fontes personalizadas.

Outros recursos incluem a personalização de *breakpoints*, indicadores de carregamento e o tema da barra de rolagem. A plataforma ainda permite criar temas personalizados para diferentes elementos visuais, além de ajustar propriedades como visibilidade condicional, *padding*, alinhamento, altura, largura, bordas, opacidade e sombra. Para elementos de responsáveis pelo *layout*, é possível definir o alinhamento horizontal e vertical, o espaçamento entre os componentes e configurar a rolagem.

Outsystems

A Figura 11 demonstra que no *Outsystems* foi possível lidar com o estilo das fontes (tamanho, cor, itálico, sublinhado ou negrito), o alinhamento, a largura e altura do componente, bem como definir individualmente os valores para *paddings*, *margins* e bordas, sejam a espessura da linha ou o arredondamento dos cantos.

Figura 11 – Customização no *Outsystems*



Fonte: Elaborada pelo autor.

A estilização é livre caso o usuário possua conhecimento em *CSS*, já que a plataforma permite declaração de classes para serem reutilizadas, possuindo inclusive algumas já por padrão, como ".flex-direction-column"e ".background-primary". A classe ".flex-direction-column"reorganiza os elementos de um *container* para serem dispostos em uma coluna, facilitando o controle de layouts responsivos. Já a classe ".background-primary"aplica a cor de fundo primária do tema do projeto ao componente, garantindo uma consistência visual com o restante da aplicação, sem a necessidade de definir manualmente a cor em cada elemento.

Essas classes facilitam a personalização e permitem uma padronização rápida de estilos no projeto, oferecendo flexibilidade tanto para desenvolvedores menos experientes quanto

para aqueles que desejam um controle mais avançado sobre o design visual.

Capacidade de Migração da Aplicação

Esta seção avalia a capacidade de migrar os aplicativos desenvolvidos na plataforma, tanto internamente, entre contas ou ambientes, quanto externamente, para outras plataformas. A migração interna pode ser necessária por razões como troca de proprietário ou segmentação de ambientes. A migração externa, por sua vez, envolve desafios como compatibilidade tecnológica e adaptação de código.

Adalo

Para migração externa, o *Adalo* não oferece uma forma de exportar o código-fonte ou o design do projeto, o que é uma limitação amplamente discutida nos fóruns da comunidade[2]. No entanto, a plataforma permite duas formas de reutilização interna da aplicação: copiar ou clonar[3] o aplicativo, ambas disponíveis no plano gratuito.

Cada abordagem tem suas vantagens específicas. Ao copiar um aplicativo, as coleções de dados e registros são compartilhadas entre o original e a cópia, de modo que quaisquer alterações feitas em um dos aplicativos se refletem em ambos. Essa é a melhor opção quando se deseja manter um conjunto de dados sincronizado entre múltiplas versões do mesmo aplicativo.

Por outro lado, ao clonar um aplicativo, as coleções de dados do aplicativo original são duplicadas, mas a nova versão possui um banco de dados completamente independente. Assim, as alterações feitas no banco de dados de um dos aplicativos não afetam o outro, o que é ideal quando se deseja separar as instâncias do aplicativo sem interferência mútua.

Bubble

No *Bubble*, as aplicações são exclusivamente executadas na própria plataforma, não havendo a possibilidade de exportar o código-fonte para execução em outro ambiente. Caso seja necessário migrar externamente, a lógica da aplicação deverá ser reconstruída, no entanto, o design visual pode ser exportado com o auxílio do suporte da mesma. É importante destacar que, em caso de descontinuação da plataforma, o código-fonte será disponibilizado como código

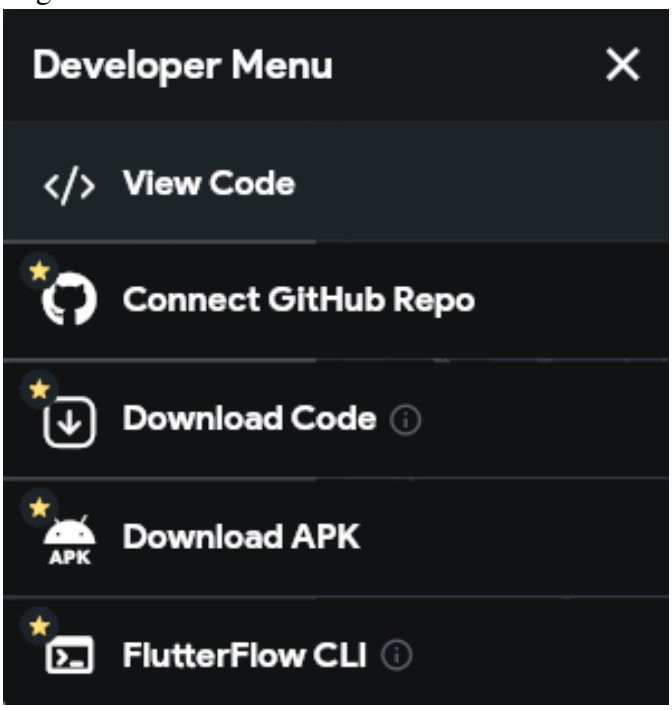
[2] https://help.adalo.com/resources/general-questions
[3] https://help.adalo.com/settings/copying-vs-cloning-your-app

aberto, possibilitando que a aplicação seja mantida em um servidor auto-hospedado.

FlutterFlow

No *FlutterFlow*, a Figura 12 apresenta duas formas de exportar o código-fonte: baixando diretamente da plataforma ou utilizando a ferramenta *FlutterFlow CLI*[4]. Também são disponibilizadas as opções de baixar a APK do projeto ou conectá-lo a um repositório do *GitHub*. Esses recurso só estão disponíveis nos planos padrão ou PRO, a única ação em relação ao código-fonte que pode ser feita no plano gratuito, é a de visualizá-lo.

Figura 12 – Menu do desenvolvedor no *FlutterFlow*



Fonte: Elaborada pelo autor

Para migração interna, é disponibilizada a duplicação da aplicação, garantindo todas as configurações feitas do *Firebase*, estilos e lógica dos elementos.

Outsystems

No caso do *Outsystems* a decisão de exportar a aplicação deve ser tomada com muito cuidado, já que essa é uma ação irreversível, uma vez que esse código é exportado e alterado em outra IDE, não será possível inserir ele novamente na plataforma. Além disso, esse processo exige que todos os aplicativos na plataforma sejam igualmente removidos ou migrados, já que

[4] https://docs.flutterflow.io/exporting/ff-cli/

não há uma opção para isolar e desativar apenas alguns aplicativos enquanto outros permanecem na mesma infraestrutura da plataforma.

Capacidade de Migração dos Dados

Esta seção avalia a capacidade de importar e exportar dados na plataforma. A importação pode ser utilizada para facilitar a inserção de dados, enquanto a exportação pode ser necessária para troca de banco de dados ou migração para outro sistema. Os principais desafios envolvem a compatibilidade de formatos, integridade dos dados, adaptação de esquemas e manutenção da segurança durante todo o processo.

Adalo

O *Adalo* permite tanto a exportação quanto a importação através de arquivos CSV, sendo fundamental então, formatar o arquivo corretamente para garantir a interpretação correta dos dados. Vale ressaltar que, cada coleção deve ser importada ou exportada separadamente, ou seja, não é possível executar essa ação com o banco completo.

Bubble

O *Bubble* possui uma funcionalidade integrada para importar dados através de arquivos CSV, porém, só está disponível nos planos pagos. Para importá-lo, é crucial que o arquivo seja formatado corretamente para a plataforma interpretar os dados adequadamente. A primeira linha do arquivo deve conter os nomes dos campos, e apenas uma tabela pode ser carregada por vez. Alguns tipos de dados complexos, como intervalos de datas, podem necessitar de formatação específica para serem reconhecidos corretamente. Já para exportar os dados, os formatos suportados são: CSV, JSON e *Newline-Delimited JSON (NDJSON)*.

FlutterFlow

O *FlutterFlow* ainda não oferece uma funcionalidade robusta para exportar dados diretamente, talvez por não possuir uma integração nativa completa com sistemas de gerenciamento de banco de dados externos. No entanto, a importação é possível através do uso de arquivos CSV, este processo permite importar dados para coleções vinculadas ao *Firebase*.

Ao importar um CSV, o sistema realiza uma validação dos tipos de dados para

garantir a consistência durante o processo, no entanto, é importante observar que o processo de importação não substitui dados existentes, apenas adiciona novas entradas. Para gerenciar dados armazenados no *Firebase*, *FlutterFlow* oferece também ações para leitura, atualização e exclusão de documentos através do *Firestore*, permitindo a manipulação direta de coleções e documentos específicos. Essa limitação quanto à exportação e a dependência do *Firebase* pode restringir as opções de migração de dados para outras plataformas.

Outsystems

Na documentação do *Outsystems* não consta formas de exportar nativamente os dados que estão no banco local, que foi utilizado neste estudo. No entanto, várias participações da comunidade nos fóruns apontam uma diversidade recursos para executar essa ação, pelo menos para casos em que envolvem o ambiente *Cloud* da plataforma, como conectores disponíveis na biblioteca *Forge*.

A importação para este banco de dados local também mostrou ser ineficiente, uma vez que ao tentar importar um arquivo *Excel*, a plataforma utilizou apenas a estrutura do mesmo, deixando as informações fora do resultado final.

Suporte à Colaboração

Esta seção avalia os mecanismos de colaboração oferecidos pelas plataformas, com foco em aspectos como edição simultânea, controle de permissões e gerenciamento de versões.

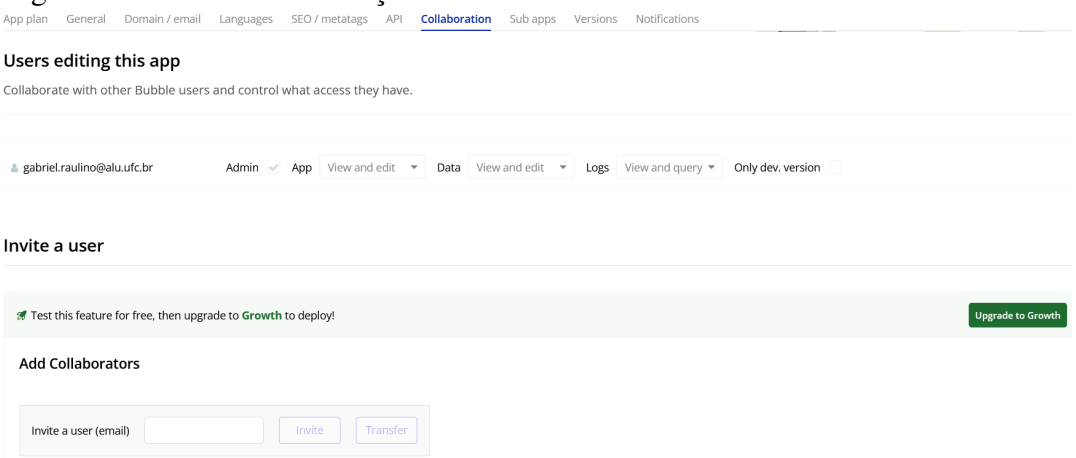
Adalo

O *Adalo* permite colaboração, porém, não de forma simultânea. A plataforma alerta que, se múltiplos usuários acessarem e editarem o aplicativo ao mesmo tempo, há o risco de que alterações não sejam salvas, e qualquer trabalho perdido não poderá ser recuperado. Além disso, usuários adicionados a um projeto têm acesso total, incluindo a permissão para excluir o aplicativo, sem possibilidade de restauração. Por isso, é fundamental garantir que apenas pessoas de confiança tenham essa permissão. Embora não haja limites para adicionar ou remover usuários, se um membro da equipe for removido ou sua conta for rebaixada, ele perderá o acesso aos aplicativos criados por ele, ficando o controle nas mãos do proprietário da equipe.

Bubble

O *Bubble* também permite a colaboração em tempo real, mas é necessário assinar o plano *Growth*, que permite apenas dois editores ao mesmo tempo, até o momento o máximo é cinco com o plano *Team*. A Figura 13 mostra a aba de colaboração no Bubble, onde administradores podem convidar usuários e definir seus níveis de acesso.

Figura 13 – Aba de colaboração no *Bubble*



Fonte: Elaborada pelo autor

O nível de Aplicativo possui duas permissões: "somente visualização" e "visualização e edição", sendo esta última limitada ao aplicativo sem afetar o banco de dados. Em relação aos dados, as permissões incluem desde "sem acesso" até "visualização e edição", permitindo a modificação completa dos dados. Para Logs, as opções variam de "sem acesso" a "visualizar e consultar", que concede acesso a informações de banco de dados e fluxos de trabalho agendados.

FlutterFlow

O *FlutterFlow* tem várias formas de colaboração, a primeira delas é compartilhar o projeto com um usuário, atribuindo para ele um dos três papéis: *Manager* que tem permissão de gerenciar o projeto, *Editor* que pode editar o projeto e por fim, *Read-Only* que pode apenas visualizar o projeto. Vale ressaltar que essa modalidade de colaboração é limitada pelas assinaturas baseadas em assentos. Isso significa que, apesar dos papéis atribuídos aos usuários, cada um terá acesso apenas às funcionalidades disponíveis conforme o plano de assinatura individual que possuem. Portanto, a capacidade de um usuário em interagir com o projeto pode ser restrita pela própria conta, independentemente do papel que lhe foi atribuído.

Também conta com o mecanismo de ramificações (*branches*), no momento, suporta

mesclas (*merges*) apenas na *branch* principal. Durante o processo de *merge*, somente o usuário que o iniciou possui acesso tanto à principal quanto à nova. As alterações que não apresentam conflitos são integradas automaticamente. Não há possibilidade de reverter uma *merge* já concluído, no entanto, é possível cancelá-lo e reiniciá-lo, caso necessário. Ademais, se o usuário sair do projeto durante o *merge* e retornar posteriormente, o progresso será preservado. Por fim, a colaboração em tempo real só está disponível nos planos *Teams* ou *Enterprise*.

Outsystems

O *OutSystems* facilita a colaboração em equipes de desenvolvimento ao permitir atividades simultâneas através de uma arquitetura escalável e ambientes isolados (*sandboxes*), evitando interferências entre desenvolvedores. A plataforma oferece um console centralizado para gerenciar controle de versões, *releases* e dependências, além de um sistema integrado de versionamento que armazena e gerencia diferentes versões de aplicativos e componentes. Adicionalmente, implementa um mecanismo de mesclagem automática para integrar alterações sem conflitos e disponibiliza ferramentas de comparação e resolução destes, garantindo uma colaboração eficiente e sincronizada entre as equipes.

5.7 Análise dos resultados obtidos

Esta seção apresenta a análise dos resultados obtidos na comparação apresentada na Seção 5.6. Cada plataforma foi analisada com base em diferentes critérios relevantes para o desenvolvimento ágil e eficiente de aplicações *web*. As principais categorias de análise incluem as facilidades e dificuldades encontradas durante o uso, o suporte a diferentes tipos de banco de dados, a oferta de recursos nativos de interface, as opções de customização de layout e estilo, e as capacidades de migração de dados e de aplicações. Também foi considerado o suporte à colaboração, essencial em equipes de desenvolvimento que trabalham simultaneamente em um mesmo projeto.

O Quadro 6 organiza e detalha esses resultados para permitir uma visão clara das capacidades e limitações de cada plataforma. Ao analisá-lo, é revelado que cada uma das LCDPs possui características específicas que as tornam mais ou menos adequadas para determinados tipos de projetos. O *Adalo* destaca-se pela simplicidade de uso e pela agilidade na criação de aplicações com funcionalidades básicas, sendo ideal para projetos menores ou com baixa

complexidade. No entanto, sua limitação na personalização e a falta de suporte à exportação de código podem restringir sua aplicação em projetos que precisem de maior escalabilidade ou flexibilidade.

Quadro 6 – Comparação entre as LCDPs objeto de estudo

Categoria	Adalo	Bubble	FlutterFlow	Outsystems
Facilidades	Simples para criar novas transações, suporte a múltiplas moedas	Muitas opções de customização para layout e estilo	Permite codificação personalizada, code copilot, várias opções de responsividade	Muitos elementos e funções nativas para manipulação de dados e tabelas
Dificuldades	Limitações de responsividade, dificuldade com enums e dropdowns	Problemas com responsividade e personalização do dropdown	Lentidão no desenvolvimento, problemas com autenticação e integração Firebase	Interface desafiadora para iniciantes, erros no login de usuários
Banco de Dados	Relacional, próprio, 200 registros no plano gratuito	Relacional, próprio, 200 registros no plano gratuito	Não-relacional (Firestore), 1 GB no plano gratuito	Relacional, próprio, 2 GB no plano gratuito
Recursos Nativos	41 elementos, 201 componentes, 30 modelos	27 elementos, 29 componentes, 2 templates	80 elementos, 173 componentes, 20 modelos	120 elementos, 0 componentes (suplementado pela biblioteca Forge), 38 modelos
Customização	Falta suporte a grid/flex-box, padding, margin, gap ausentes; dificuldade em ajustar componentes em telas menores; apenas quatro cores principais; fontes limitadas no plano gratuito	Flexível com margens, espaçamentos e opções de visibilidade; controle fino sobre o layout via regras condicionais; suporte a variáveis de cores e fontes; pode reutilizar personalizações salvas (semelhante a classes de CSS)	Oferece personalização detalhada de breakpoints, espaçamento e alinhamento com base no dispositivo; suporta temas escuros e claros; cores podem ser geradas por IA ou extraídas de imagens; fontes personalizadas disponíveis	Suporte a CSS avançado e flexbox-like; classes reutilizáveis para padronização de layout e responsividade; flexibilidade completa em personalizar cores e fontes; suporte a temas predefinidos e customização via CSS
Migração de Dados	Suporta exportação e importação via CSV; cada coleção deve ser exportada/importada separadamente	Suporta importação de dados via CSV (apenas em planos pagos); exportação disponível nos formatos CSV, JSON e NDJSON	Suporta importação via CSV para coleções Firebase; não há suporte robusto para exportação de dados	Importação limitada para banco de dados local; suporte a múltiplas fontes de dados através da biblioteca Forge
Migração de Aplicação	Não suporta exportação de código-fonte; pode clonar ou copiar apps internamente	Não suporta exportação completa de código; pode exportar design visual com assistência	Suporta exportação de código-fonte via CLI; conexão com GitHub para versionamento contínuo disponível	Suporta exportação de código, mas é irreversível; migração para outras plataformas exige cuidados técnicos
Suporte à Colaboração	Não permite edição simultânea; todos os colaboradores têm acesso total ao projeto, sem limitação de permissões	Colaboração em tempo real disponível em planos pagos; suporte para até cinco editores simultâneos com controle granular de permissões	Colaboração via papéis, suporte a branches e merges; colaboração em tempo real disponível em planos pagos	Suporte robusto à colaboração simultânea com versionamento automático e controle de versões

Fonte: Elaborado pelo autor.

Por outro lado, o *Bubble* oferece um nível de customização mais robusto, permitindo maior controle sobre o design e o comportamento dos elementos. Embora seja eficiente para projetos que permanecem na plataforma, sua dificuldade em lidar com migrações de dados e a ausência de exportação completa de código limitam sua utilização em contextos onde seja necessária uma migração futura. Ainda assim, para equipes pequenas que valorizam a flexibilidade visual e a colaboração em tempo real, o *Bubble* se mostra uma solução prática.

O *FlutterFlow*, com sua capacidade de integrar código personalizado e a flexibilidade na exportação do código-fonte, é uma escolha interessante para projetos que exigem funcionalidades mais complexas ou que possam crescer com o tempo. A plataforma se adequa bem a cenários onde a escalabilidade e a personalização são requisitos essenciais, embora sua lentidão durante o desenvolvimento possa ser um fator limitante, especialmente para equipes menores ou com orçamentos mais restritos.

Já o *Outsystems* se posiciona como a solução mais completa para projetos de grande porte ou corporativos. A ampla gama de recursos nativos de interface, a capacidade de integração com bancos de dados externos e a flexibilidade oferecida pela plataforma a tornam ideal para empresas que buscam uma solução de desenvolvimento robusta e que permita a colaboração eficiente entre equipes. No entanto, essa robustez vem acompanhada de uma curva de aprendizado mais acentuada, o que pode ser desafiador para quem busca uma implementação mais rápida ou intuitiva.

Em suma, a escolha da plataforma ideal dependerá das necessidades específicas de cada projeto. Para aplicações de menor complexidade e equipes reduzidas, *Adalo* e *Bubble* apresentam-se como opções viáveis. No entanto, para projetos que demandem maior personalização e escalabilidade, *FlutterFlow* e *Outsystems* se mostram alternativas mais adequadas. A decisão deve considerar, portanto, tanto os requisitos técnicos quanto as expectativas de crescimento e manutenção do projeto.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma análise comparativa entre plataformas de desenvolvimento *low code*, focada na criação de MVP. A análise revelou que as plataformas possuem características distintas que influenciam diretamente no processo de desenvolvimento, como a personalização, integração com bancos de dados e capacidade de exportação de código.

O *Adalo* e o *Bubble* mostraram-se eficientes para o desenvolvimento rápido de aplicações simples, principalmente para equipes reduzidas ou projetos com requisitos menos complexos. No entanto, a limitação em funcionalidades avançadas, como exportação de código e maior personalização, coloca essas plataformas em desvantagem para projetos que exigem escalabilidade.

Por outro lado, o *Flutterflow* e o *Outsystems* destacaram-se por suas capacidades robustas de personalização e exportação de código, sendo mais indicados para projetos maiores ou que demandem integração com sistemas externos. O *Outsystems*, em especial, mostrou-se adequado para aplicações corporativas de grande porte, oferecendo uma ampla gama de recursos nativos de interface, embora com uma curva de aprendizado mais acentuada.

Em suma, a escolha da plataforma ideal depende das necessidades específicas do projeto. Projetos de pequena complexidade se beneficiam da simplicidade e agilidade oferecidas pelo *Adalo* e *Bubble*, enquanto o *Flutterflow* e *Outsystems* são mais adequados para cenários que exijam maior flexibilidade e escalabilidade.

Embora este trabalho tenha fornecido uma análise detalhada de plataformas de desenvolvimento *low code*, algumas lacunas permanecem. Primeiramente, ampliar o estudo para incluir plataformas adicionais, uma vez que o mercado de LCDPs está em constante evolução com novas ferramentas sendo lançadas regularmente.

Outro aspecto relevante é a exploração da migração de aplicações e dados entre plataformas. Embora algumas delas ofereçam suporte básico para essa funcionalidade, ainda há desafios significativos, especialmente em contextos de alta complexidade, que exigem uma avaliação mais aprofundada.

Além disso, trabalhos futuros podem focar em casos de uso em setores específicos, como saúde, educação ou governos, onde os requisitos podem ser bastante diferenciados, exigindo customizações específicas que não foram abordadas neste estudo. Outra abordagem poderia ser explorar diferentes tipos de plataformas, focando em inteligência artificial ou automação, em vez de apenas desenvolvimento *web*. Também seria relevante investigar como essas

plataformas lidam com APIs e integrações com outros sistemas, ampliando sua aplicabilidade em cenários mais complexos.

Por fim, um estudo mais aprofundado sobre colaboração em equipes distribuídas e o suporte dessas plataformas a esse tipo de ambiente de trabalho também seria valioso. A pandemia acelerou o trabalho remoto e esse fator deve ser considerado uma demanda crescente nos próximos anos.

REFERÊNCIAS

ALWIS, I. D.; SEDERA, D. Minimum viable product in information systems development context: Systematic mapping study. 2022. Disponível em: https://aisel.aisnet.org/acis2022/83/. Acesso em: 27 out. 2023.

BASTOS, J. d. S. **Análise comparativa entre o Iris low-code certidão e outras plataformas low-code**. 2023. Disponível em: https://repositorio.ufc.br/handle/riufc/73961. Acesso em: 10 set. 2023.

BLOOMBERG. **The Low-Code/No-Code Movement: More Disruptive Than You Realize**. 2017. Disponível em: https://www.forbes.com/sites/jasonbloomberg/2017/07/20/the-low-codeno-code-movement-more-disruptive-than-you-realize/?sh=69eae281722a/. Acesso em: 26 set. 2023.

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: 10 set. 2023.

BUCAIONI, A.; CICCHETTI, A.; CICCOZZI, F. Modelling in low-code development: a multi-vocal systematic review. **Software and Systems Modeling**, v. 21, n. 5, p. 1959–1981, Outubro 2022. Disponível em: https://doi.org/10.1007/s10270-021-00964-0. Acesso em: 8 out. 2024.

KÄSS, S.; STRAHRINGER, S.; WESTNER, M. Drivers and inhibitors of low code development platform adoption. In: IEEE. **2022 IEEE 24th Conference on Business Informatics (CBI)**. [*S. l.*], 2022. v. 1, p. 196–205. Acesso em: 14 set. 2023.

KUMAR, P. H.; MALA, G. A. H2run: An efficient vendor lock-in solution for multi-cloud environment using horse herd runge kutta based data placement optimization. **Transactions on Emerging Telecommunications Technologies**, Wiley Online Library, v. 33, n. 9, p. e4541, 2022. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4541. Acesso em: 12 out. 2023.

LENARDUZZI, V.; TAIBI, D. Mvp explained: A systematic mapping study on the definitions of minimal viable product. In: IEEE. **2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)**. [*S. l.*], 2016. p. 112–119. Disponível em: https://ieeexplore.ieee.org/abstract/document/7592786/. Acesso em: 14 set. 2023.

METROLHO, J.; ARAúJO, R.; RIBEIRO, F.; CASTELA, N. An approach using a low-code platform for retraining professionals to ict. In: **EDULEARN19 Proceedings**. [*S. l.*]: IATED, 2019. (11th International Conference on Education and New Learning Technologies), p. 7200–7207. ISBN 978-84-09-12031-4. ISSN 2340-1117. Disponível em: https://doi.org/10.21125/edulearn.2019.1719. Acesso em: 17 out. 2023.

MIYAKE, T.; MASUDA, Y.; OGUCHI, A.; ISHIDA, A. Strategic risk management for low-code development platforms with enterprise architecture approach: Case of global pharmaceutical enterprise. In: SPRINGER. **International KES Conference on Innovation in Medicine and Healthcare**. [*S. l.*], 2023. p. 88–100. Disponível em: https://link.springer.com/chapter/10.1007/978-981-99-3311-2_9. Acesso em: 13 out. 2023.

MOHAMMED, C. M.; ZEEBAREE, S. R. Sufficient comparison among cloud computing services: Iaas, paas, and saas: A review. **International Journal of Science and Business**, IJSAB International, v. 5, n. 2, p. 17–30, 2021. Disponível em: https://ideas.repec.org/a/aif/journl/v5y2021i2p17-30.html. Acesso em: 23 out. 2023.

RICHARDSON, C.; RYMER, J. R. Vendor landscape: The fractured, fertile terrain of low-code application platforms. **Forrester, Janeiro**, p. 12, 2016. Disponível em: https://informationsecurity.report/Resources/Whitepapers/0eb07c59-b01c-4399-9022-dfc297487060_Forrester%20Vendor%20Landscape%20The%20Fractured,%20Fertile%20Terrain.pdf. Acesso em: 15 nov. 2023.

RIES, E.; CALADO, A.; IMMELT, J. **A startup enxuta: Como usar a inovação contínua para criar negócios radicalmente bem-sucedidos**. [*S. l.*]: Editora Sextante, 2019. ISBN 9788543108629.

ROKIS, K.; KIRIKOVA, M. Exploring low-code development: A comprehensive literature review. **Complex Systems Informatics and Modeling Quarterly**, n. 36, p. 68–86, 2023. Disponível em: https://csimq-journals.rtu.lv/article/view/csimq.2023-36.04. Acesso em: 12 set. 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. Disponível em: https://ieeexplore.ieee.org/abstract/document/9226356/. Acesso em: 7 set. 2023.

SANKHYA, R. **O que é um MVP (Produto Mínimo Viável) e como criar um?** 2023. Acesso em: 12 nov. 2023. Disponível em: https://www.sankhya.com.br/blog/como-criar-um-mvp/.

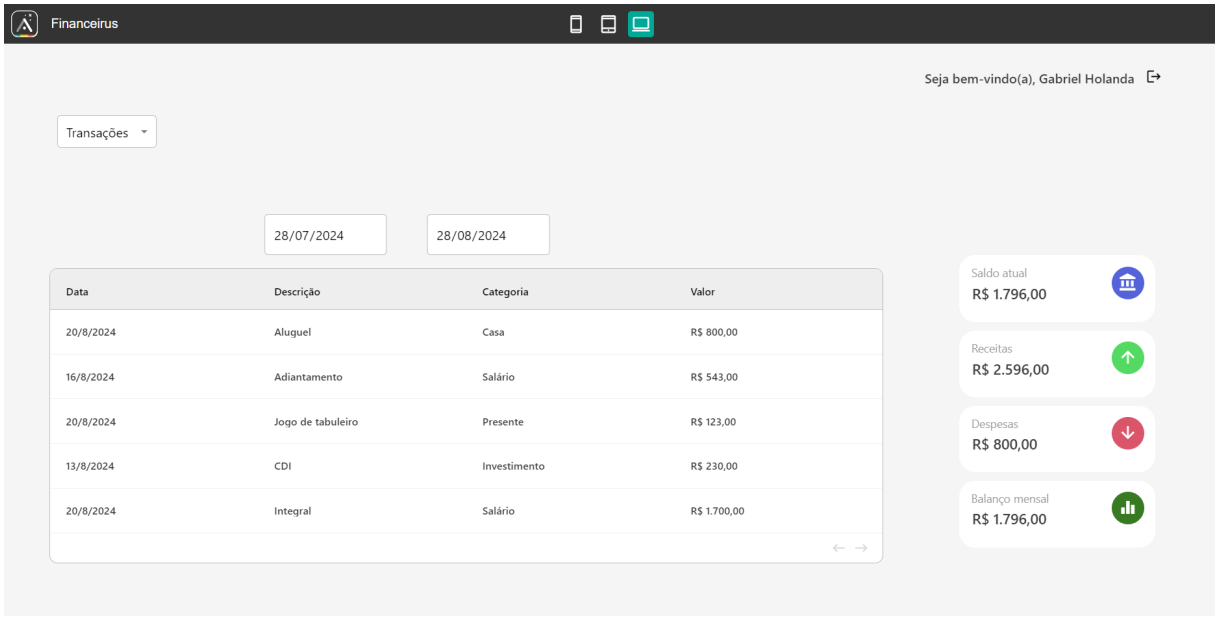
VALENÇA, M.; DINIZ, W.; PINCOVSKY, M.; FRANÇA, C.; CABRAL, G. Mercado de trabalho em tecnologia da comunicação e informação (ti): análise de um experimento de aproximação entre academia e indústria no porto digital. In: SBC. **Anais do VIII Workshop sobre Aspectos Sociais, Humanos e Econômicos de Software**. [*S. l.*], 2023. p. 1–10. Disponível em: https://sol.sbc.org.br/index.php/washes/article/view/24770/24591. Acesso em: 23 out. 2023.

VINCENT, P.; IIJIMA, K.; DRIVER, M.; WONG, J.; NATIS, Y. Magic quadrant for enterprise low-code application platforms. **Gartner report**, v. 120, 2019. Disponível em: https://scholar.google.com/scholar_lookup?title=Magic+Quadrant+for+Enterprise+Low-Code+Application+Platforms&author=Vincent,+P.&author=Iijima,+K.&author=Driver,+M.&author=Jason,+W.&author=Natis,+Y.&publication_year=2016&journal=Gart.+Rep.. Acesso em: 22 set. 2024.

APÊNDICE A – INTERFACE DO MVP NAS PLATAFORMAS

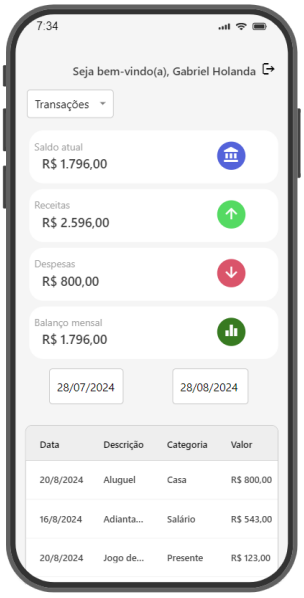
As figuras 14 a 21 apresentam os resultados obtidos com a implementação do MVP nas quatro plataformas analisadas: *Adalo*, *Bubble*, *FlutterFlow* e *Outsystems*. As imagens mostram as interfaces desenvolvidas tanto nas versões *desktop* quanto *mobile*, destacando as diferenças visuais entre as plataformas para diferentes dispositivos.

Figura 14 – Página inicial do MVP no Adalo - *Desktop*



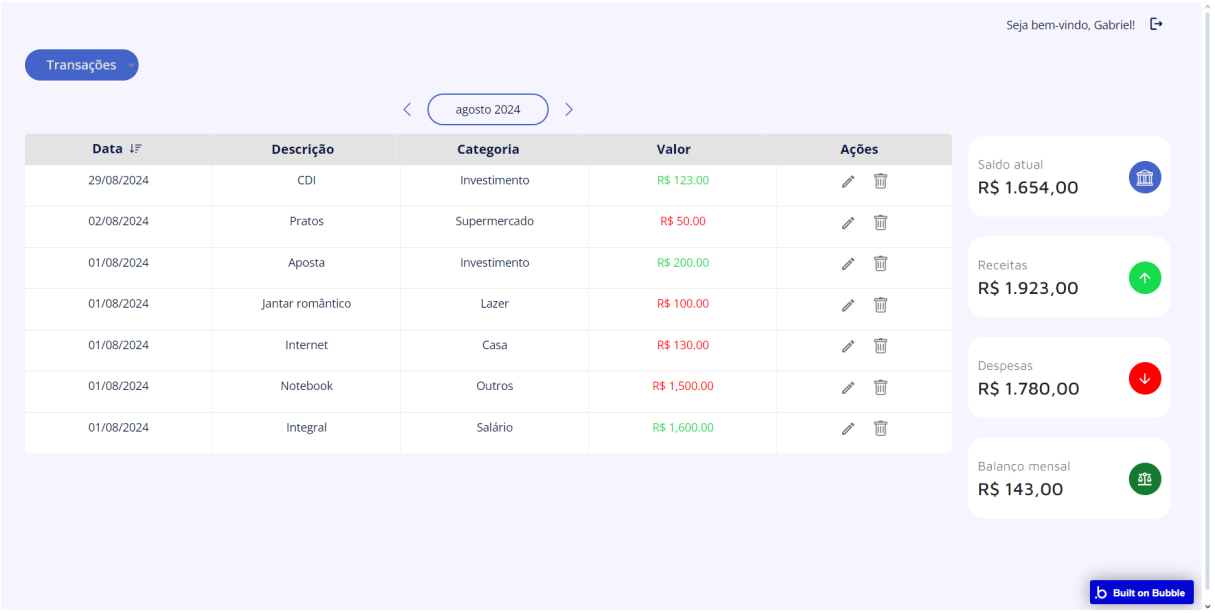
Fonte: Elaborada pelo autor

Figura 15 – Página inicial do MVP no Adalo - *Mobile*



Fonte: Elaborada pelo autor

Figura 16 – Página inicial do MVP no Bubble - *Desktop*

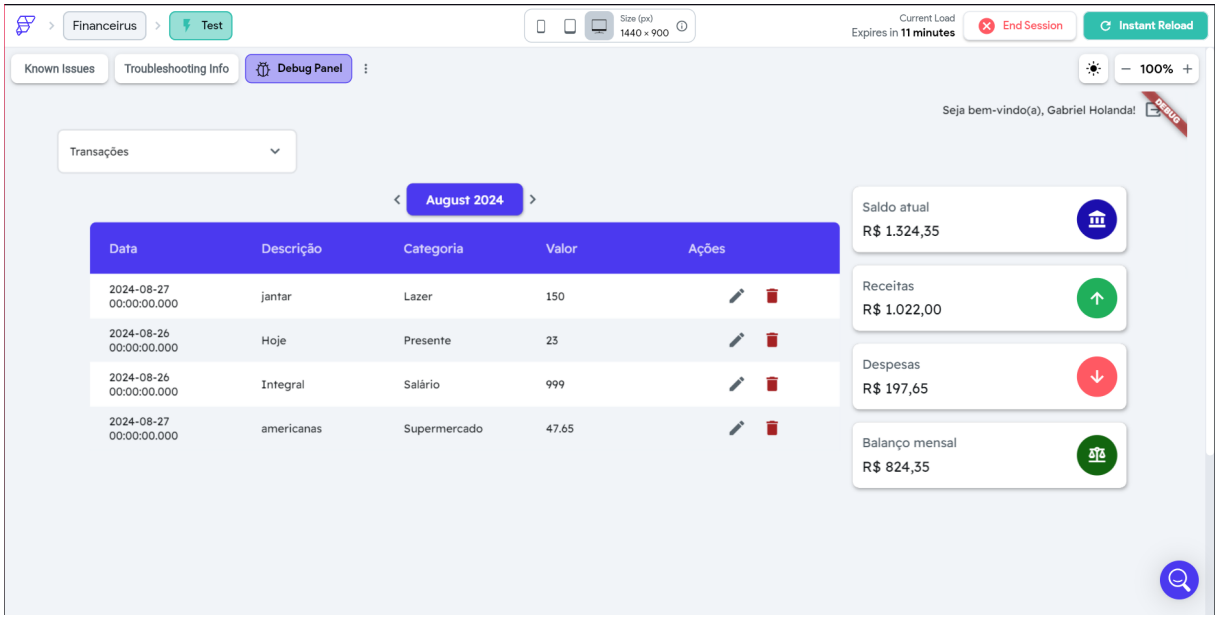


Fonte: Elaborada pelo autor

Figura 17 – Página inicial do MVP no Bubble - *Mobile*

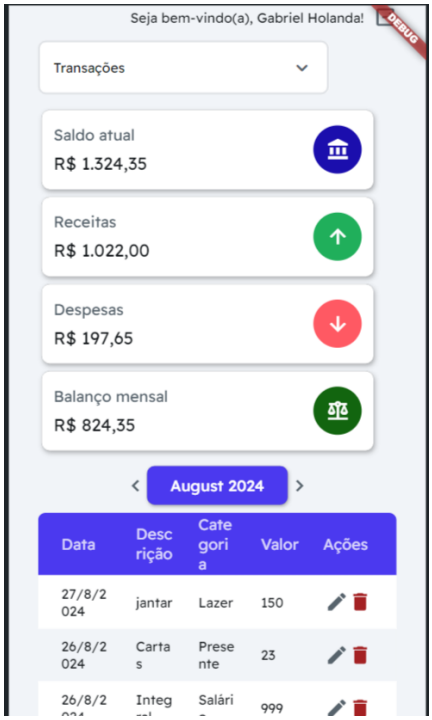
Fonte: Elaborada pelo autor

Figura 18 – Página inicial do MVP no FlutterFlow - *Desktop*



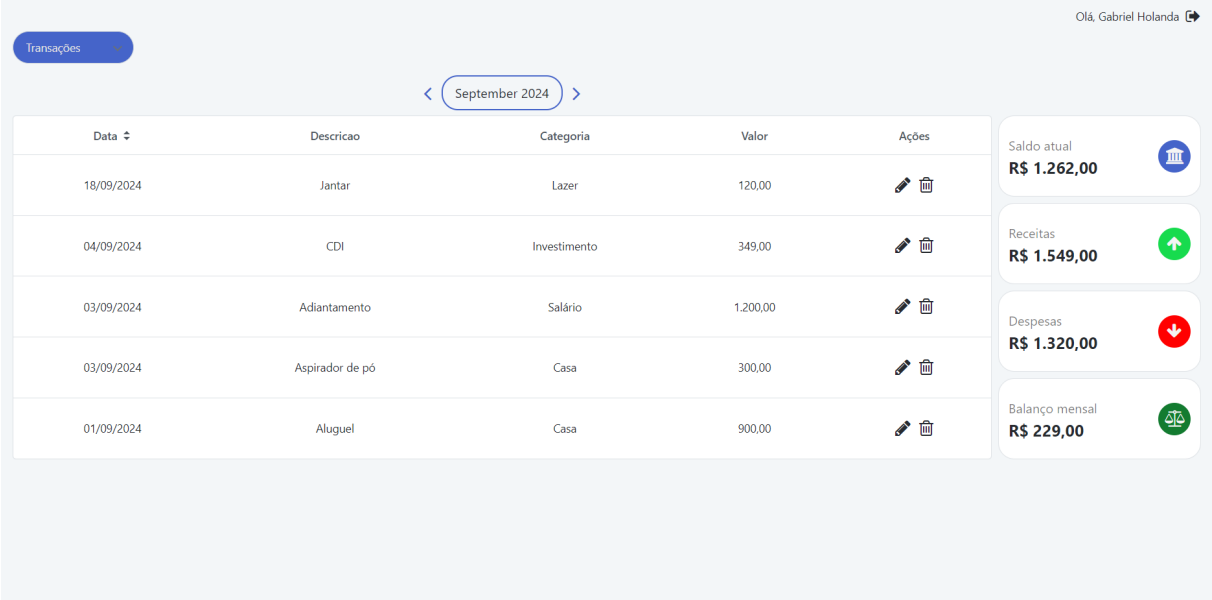
Fonte: Elaborada pelo autor

Figura 19 – Página inicial do MVP no FlutterFlow - *Mobile*



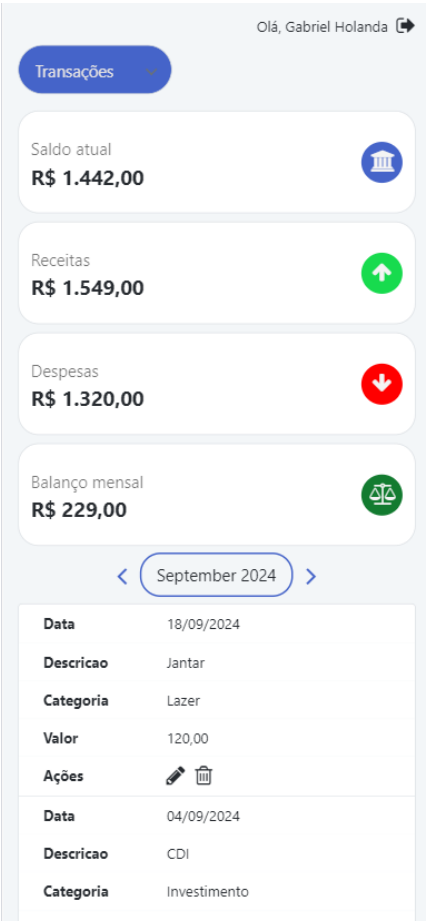
Fonte: Elaborada pelo autor

Figura 20 – Página inicial do MVP no Outsystems - *Desktop*



Fonte: Elaborada pelo autor

Figura 21 – Página inicial do MVP no Outsystems - *Mobile*



Fonte: Elaborada pelo autor