Anais do Congresso Brasileiro de Iniciação Científica
(https://periodicos.unisanta.br/COB)

ISSN 2317-1316

ARQUITETURA PARA CADASTROS DE DOMÍNIOS PARA UMA PLATAFORMA LOW-CODE DE CRIAÇÃO DE CRUD

DOMAIN REGISTRATION ARCHITECTURE FOR A LOW-CODE CRUD GENERATION PLATFORM

Vitor Kenzo Martins Ochida, Bruno da Silva Rodrigues, Fabio Kawaoka Takase
Universidade Presbiteriana Mackenzie, Faculdade de Computação e Informática, Curso de Sistemas de Informação
E-mail para contato: vitor.ochida@mackenzista.com.br

RESUMO – O desenvolvimento de software enfrenta forte demanda por produtividade, impulsionando a adoção de plataformas low-code para automação de CRUDs. Contudo, abordagens atuais apresentam rigidez na modelagem de dados e lacunas na geração de interfaces. Este artigo relata a fase inicial do projeto AppBuilder, focando no desenvolvimento do DomainBuilder, módulo responsável pelo cadastro e gestão de domínios. A arquitetura integra uma Single Page Application em Vue.js, intermediada por camada BFF em Spring Boot conectada à AssetAPI e persistência em MariaDB. Como resultado, os cadastros e parametrização de domínios.

Palavras-chave: Low-code; Web Semântica; CRUD; Ontologias; Arquitetura de Software.

ABSTRACT – Software development faces a high demand for productivity, driving the adoption of low-code platforms for CRUD automation. However, current approaches present rigidity in data modeling and gaps in interface generation. This article reports on the initial phase of the AppBuilder project, focusing on the development of the DomainBuilder, a module responsible for domain registration and management. The architecture integrates a Single Page Application in Vue.js, mediated by a Spring Boot BFF layer connected to the AssetAPI, with persistence in MariaDB. As a result, domain registrations and parameterization.

Keywords: Low-code; Semantic Web; CRUD; Ontologies; Software Architecture.

1 INTRODUÇÃO

Na atualidade, o processo para desenvolver software está passando por uma crescente demanda por entregas rápidas e de qualidade, fomentando um movimento de digitalização acelerada nas empresas (RAFI, 2022). Contudo, os modelos tradicionais utilizados para

ISSN 2317-1316

desenvolver software se mostram morosos e ineficientes para reuso de componentes (JIN2008), resultando em projetos que não atendem às atuais expectativas de prazos de entrega (FADILLAH, 2025).

Considerado fundamental para qualquer aplicação que manipule um banco de dados, no qual é necessário criar, ler, atualizar e deletar, um CRUD (Create, Read, Update, Delete) faz parte das funcionalidades mais básicas da grande maioria das aplicações web. Diante disso, é inevitável sua implementação em sistemas em geral, contudo, a criação de CRUD pode ser automatizada para reduzir tempo de desenvolvimento aumentando a produtividade (FADILLAH, 2025). No cenário atual, é evidente a tendência de aumento de uso de plataformas low-code no mercado (VELÁSQUEZ, 2024) e como elas elevam a produtividade e a agilidade para entregas de projetos (RAFI, 2025).

Atualmente, é possível encontrar soluções para a automação de sistemas CRUD, com destaque para o framework desenvolvido por (Anuar, 2022). Nesta proposta se utilizou uma metodologia ágil, iterativa e incremental para automatizar a geração das operações CRUD e a camada de interface (front end), integrando funções de gestão de registro, como busca, arquivamento e trilhas de auditoria. A eficiência da solução foi validada pelos autores por meio de análise de funcionalidades de Determining an Evaluation Methodology for Software Engineering Methods and Tools (DESMET) e da aplicação da escala de usabilidade System Usability Scale (SUS), no qual o framework alcançou 88% de aprovação, superando a média de 34% obtida por outros geradores. Contudo, os autores pontuaram como ponto de evolução a necessidade de incluir recursos de restauração para registros e a flexibilização de regras de negócio engessadas e pouco personalizadas.

Outro exemplo de ferramenta low-code que evidencia a redução no tempo de desenvolvimento é uma pipeline apresentada por (Fadillah, 2025) que automatiza a criação de operações CRUD a partir de diagramas de classe UML (Unified Modeling Language), a metodologia utilizada consistiu em cinco estágios: modelagem visual no StarUML, extração de entidades via parser Python para converter esquemas JSON, integração na plataforma Budibase e implantação final validada por um estudo de caso de um Sistema de Gestão de Revendedores (DMS - Dealer Management System). Os resultados principais apresentaram uma redução de mais de 70% no tempo de desenvolvimento, 22 minutos contra 1 hora no método manual, além

ISSN 2317-1316

de uma consistência de 99% entre o modelo e o código e uma queda notável no número de erros e nas linhas de configuração necessárias. Todavia, o sistema apresenta ausência de um front end gerado automaticamente, sendo necessária a implementação manual da equipe de desenvolvedores. Segundo os autores, um dos pontos de melhoria da ferramenta seria a automação da camada de interface.

Em síntese, o cenário atual das ferramentas low-code voltadas à geração automatizada de CRUDs apresenta algumas carências. Entre elas, destacam-se a baixa flexibilidade na modelagem de dados e a rigidez na definição de requisitos de negócio, o que resulta em sistemas mais difíceis de personalizar, como (022) apontou em seu projeto. Somando a isso, observa-se a escassez de ferramentas que realizem a geração automatizada do front end, demandando o desenvolvimento manual dessa camada (FADILLAH, 2025).

Todavia, tais limitações, especialmente a rigidez na modelagem lógica e de dados, podem ser mitigadas por meio de tecnologias de Web Semântica. A utilização de padrões como Web Ontology Language (OWL) e Resource Description Framework (RDF) permite substituir a lógica de negócios codificada manualmente por uma especificação formal e explícita do domínio, conferindo, ao mesmo tempo, maior dinamismo para o front end (SILEGA et al., 2023).

Ademais, nota-se que a solução proposta por (Fadillah, 2025) não contempla a geração de interface. Já a aplicação de (Anuar, 2022), embora gere front end, não integra funcionalidades de Design System. Conforme aponta (Moore, 2020), um Design System oferece vantagens estratégicas ao estabelecer um conjunto de padrões que permitem o gerenciamento do design em larga escala via componentes reutilizáveis. Tal abordagem assegura a consistência visual e funcional do sistema, tirando divergências estéticas e poupando os desenvolvedores da necessidade de recriar componentes para cada nova página web ou nova aplicação.

Com base nesse contexto, este trabalho insere-se no projeto de concepção da plataforma low-code AppBuilder, voltada à aceleração do ciclo de desenvolvimento de aplicações CRUD mediante geração automatizada de interfaces e funcionalidades orientadas por ontologias. Como a geração e a consistência visual dessas aplicações dependem fundamentalmente da parametrização semântica e da delimitação do contexto de negócio, este artigo relata a fase

ISSN 2317-1316

inicial da plataforma, com ênfase na modelagem semântica dos dados e no desenvolvimento do DomainBuilder, módulo responsável pelo cadastro, estruturação lógica e gerenciamento dos domínios que servem de base para a posterior geração automatizada do front end e das operações CRUD pelo AppBuilder.

Para apresentar este projeto, o artigo está organizado da seguinte forma: na Seção 2 são descritos os Materiais e Métodos utilizados, detalhando a arquitetura global da plataforma e as decisões de implementação do módulo DomainBuilder, na Seção 3 apresentam-se os Resultados e Discussões decorrentes da validação desse módulo e, por fim, na Seção 4 são expostas as Conclusões do trabalho e os direcionamentos para a conclusão do ecossistema AppBuilder.

2 MATERIAIS E MÉTODOS

Este projeto caracteriza-se como uma pesquisa de desenvolvimento tecnológico aplicada à arquitetura de software. O objetivo principal compreende a estruturação dos alicerces conceituais e a implementação da interface Single Page Application (SPA) do módulo DomainBuilder, primeiro componente funcional da plataforma AppBuilder. Para atingir esse objetivo, inicialmente realizou-se uma pesquisa de caráter exploratório em bases de dados como IEEE Xplore e ACM para fundamentação teórica, acompanhada de consulta às recomendações e padrões formais estabelecidos pelo World Wide Web Consortium (W3C). Complementarmente, foram realizadas consultas com profissionais e pesquisadores da área de Engenharia de Software com experiência em modelagem ontológica, permitindo mapear os principais gargalos conceituais e práticos no projeto de geradores low-code.

A partir dessas diretrizes, delimitou-se a solução como uma arquitetura low-code orientada a ontologias, dividida em duas frentes complementares o DomainBuilder, módulo gestor de domínios e parametrizações semânticas, e o AppBuilder, módulo gerador de CRUDs e telas finais. No escopo deste artigo, priorizou-se o ciclo completo de construção do DomainBuilder. Para a modelagem semântica dos dados, adotou-se a linguagem formal OWL (Web Ontology Language) (W3C, 2012) e os padrões RDF (Resource Description Framework), estruturados no ambiente Protégé (PROTÉGÉ, 2026). A infraestrutura computacional utilizou o banco de dados MariaDB, framework Spring Boot para o desenvolvimento do backend e da camada BFF (Backend For Frontend), e Vue.js na construção da SPA do DomainBuilder. Os

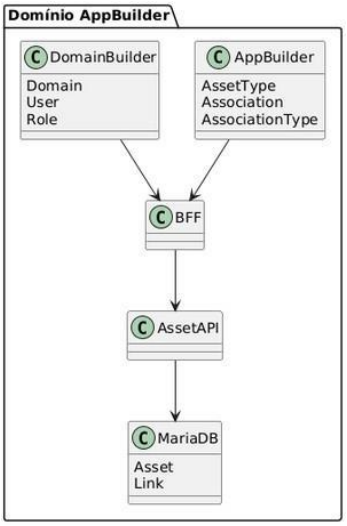
ISSN 2317-1316

detalhes da integração arquitetural são apresentados a seguir.

2.1 Arquitetura do Sistema

O desenvolvimento da plataforma orientou-se pelo conceito de domínio, definido como a área de negócio na qual um sistema de software é aplicado (Sommerville, 2018). O sistema adota uma arquitetura modular, conforme apresentado na Figura 1.

Figura 1: Diagrama de Módulos e Componentes da Plataforma.



Conforme ilustrado na Figura 1, a plataforma foi concebida sob uma visão modular integrada por duas interfaces em SPA: o DomainBuilder e o AppBuilder. Ambas se conectam ao ecossistema por meio do BFF (Backend For Frontend), que intermedeia o acesso à AssetAPI e a um banco de dados relacional MariaDB. A AssetAPI centraliza as classes genéricas responsáveis por processar as regras semânticas e orquestrar as operações CRUD de cada objeto instanciado sem demanda de código manual. O BFF atua gerenciando sessões e tokens de segurança, isolando o navegador de credenciais sensíveis. Enquanto o módulo DomainBuilder viabiliza a configuração administrativa dos domínios, tipos de dados e padrões de interface (Look & Feel), o AppBuilder consome essa base semântica para materializar as interfaces de CRUD para os usuários finais.

No âmbito da *AssetAPI*, todos os objetos nesse sistema são estruturados em torno de cinco entidades baseadas em triplas de dados RDF (*Resource Description Framework*), responsáveis por descrever semanticamente o domínio da aplicação. Essas entidades estabelecem o mapeamento direto entre as regras da ontologia em OWL (*Web Ontology*

ISSN 2317-1316

Language) e os objetos de manipulação em Java, refletindo a modelagem formal previamente realizada no ambiente Protégé. As cinco entidades compreendem:

AssetType: define as propriedades base de um determinado tipo de objeto, analogamente à definição de classes no paradigma de Orientação a Objetos. No contexto ontológico, corresponde diretamente ao construto owl:Class, servindo como molde para a criação de novas sub-classes.

Asset: constitui a instância concreta em banco de dados de um determinado *AssetType*, armazenando seus atributos dinâmicos e representando o sujeito ou objeto da aplicação. Semanticamente, equivale a um owl:NamedIndividual.

AssociationType: especifica os verbos conceituais e os predicados válidos no vocabulário do domínio, estabelecendo os tipos de relacionamentos admitidos entre classes. Sua correspondência ontológica é a owl:ObjectProperty.

Association: consolida as regras de negócio e restrições de integridade entre tipos de dados, definindo quais entidades de origem (sujeito) podem se conectar a entidades de destino (objeto). No modelo conceitual, expressa o papel de uma regra de restrição (owl:Restriction).

Link: materializa a conexão física entre duas instâncias concretas de *Asset* no banco de dados relacional, registrando os identificadores de origem, destino e a relação associada. Esse elemento concretiza a tripla RDF (*Resource Description Framework*).

2.2 DomainBuilder e AppBuilder

Os SPAs DomainBuilder e AppBuilder são as interfaces usadas pelos usuários. O DomainBuilder é o equivalente a uma tela de cadastros para domínios, onde o usuário pode adicionar informações básicas sobre um determinado domínio, como seu usuário, senha e role para acessar um determinado domínio, Look&Feel para definir um design pré-configurado e Domain, que identifica a qual domínio superior àquele subdomínio se vincula, caso não seja associado a um domínio pai explícito, recebe por padrão a constante “System". Por sua vez, o AppBuilder constitui a interface responsável por capturar as entradas do usuário, permitindo

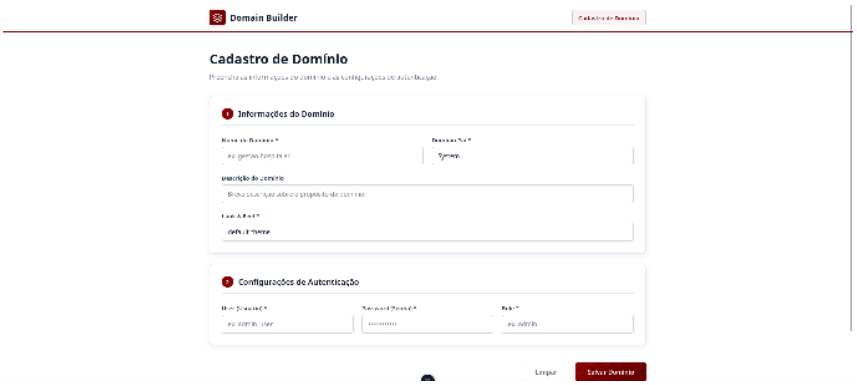
ISSN 2317-1316

registrar na AssetAPI as entidades e seus respectivos relacionamentos.

3 RESULTADOS E DISCUSSÃO

Com base no modelo apresentado na Figura 1, foi desenvolvido o SPA DomainBuilder, que tem o papel de cadastrar os domínios de cada aplicação na AssetAPI, que por sua vez irá armazenar as informações dentro do banco de dados MariaDB.

Figura 2: Tela de cadastro de domínios.



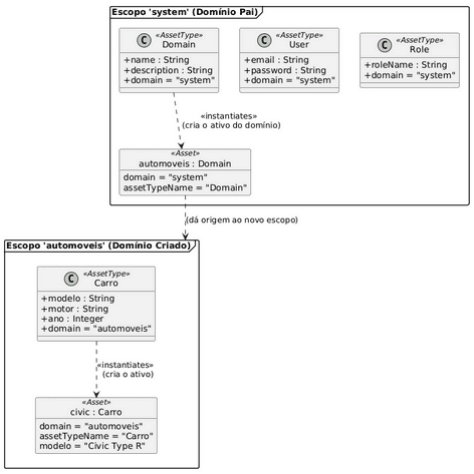
A Figura 2 ilustra os campos fundamentais para a criação de um domínio, com destaque para Look & Feel e Domínio Pai. O campo Look & Feel recebe um identificador textual associado a uma folha de estilo, responsável por definir o padrão visual das aplicações vinculadas àquele domínio. Por sua vez, o campo Domínio Pai estabelece uma relação hierárquica análoga ao conceito de herança, indicando a qual domínio superior a nova entidade pertence. Para viabilizar a instanciação do primeiro domínio nessa estrutura, adota-se o domínio System como nó-raiz autoinstanciável. Definido como uma constante na AssetAPI, o *System* é atribuído como pai padrão quando um domínio é criado sem um predecessor explícito, viabilizando a inicialização da hierarquia.

Na Figura 3 é possível observar em detalhes a estrutura das entidades ontológicas dentro do AssetAPI necessárias para criar um Asset para um determinado domínio. A imagem apresenta um exemplo prático aplicado ao contexto de automóveis, no qual se visualiza todo o escopo do domínio System, considerado o domínio pai. O domínio System é composto inicialmente por três AssetTypes fundamentais: User, Role e Domain. O AssetType User funciona como o identificador, registrando quem cria Assets, AssetTypes, Associations,

ISSN 2317-1316

AssociationTypes ou Links. O AssetType Role define o nível de permissão e os privilégios de acesso que cada User possui dentro de um domínio. Já o AssetType Domain serve como um molde para a criação de novos domínios. A partir da utilização do molde fornecido pelo AssetType Domain, realiza-se a criação de um domínio chamado “automoveis”. Esse domínio pertence ao domínio System e ao tipo Domain. Uma vez criado o domínio “automoveis”, torna-se possível modelar e conceber entidades específicas que fazem sentido dentro desse novo contexto veicular. Nesse nível, cria-se o AssetType Carro, que estabelece os atributos padronizados que qualquer carro cadastrado deverá possuir, como modelo, motor e ano de fabricação, vinculando esse esquema ao domínio “automoveis”. Por fim, ao instanciar esse molde, tem-se a criação de um Asset concreto, representado no diagrama pelo veículo civic do tipo Carro. Essa instância real herda toda a definição de atributos estipulada pela sua classe pai Carro, porém recebe valores particulares e independentes, tais como o modelo Civic Type R. Dessa maneira, ela mantém a estrutura semântica padrão ao mesmo tempo em que armazena dados próprios que a distinguem de quaisquer outras instâncias de carros existentes no sistema.

Figura 3: Esquema Conceitual de Instanciação de um domínio.



4 CONCLUSÃO

O presente trabalho apresentou a arquitetura e a implementação dos componentes iniciais de uma plataforma de desenvolvimento de software orientada a ontologias. A arquitetura modular proposta é estruturada em torno de uma API genérica (AssetAPI), um intermediário de apresentação e segurança (BFF) e interfaces em Single Page Application (SPA). Como

ISSN 2317-1316

resultado principal, destaca-se o desenvolvimento da aplicação de página única denominada DomainBuilder. Por meio dessa interface, viabilizou-se o cadastro parametrizado de domínios diretamente na AssetAPI com persistência estruturada no banco de dados MariaDB. Mediante da implementação do módulo DomainBuilder, validou-se a camada de gestão de domínios, permitindo a definição hierárquica a partir de um nó-raiz (System), a associação de identidades visuais (Look & Feel) e a inicialização segura dos esquemas fundamentais de permissões (Domain, User e Role).

A próxima etapa do presente trabalho é desenvolver o AppBuilder, que será o componente responsável da aplicação por criar os Assets e seus relacionamentos com outros Assets. Nessa Single Page Application, o usuário poderá definir AssociationTypes e Associations, ou seja, as regras semânticas da aplicação, visto que a ontologia já está modelada. Vale ressaltar que os componentes reais dentro do banco de dados são os Assets e Links, que representam as estruturas em triplas Resource Description Framework nas quais não possuem inteligência semântica, sendo apenas estruturas dentro do banco de dados.

5 DECLARAÇÃO DE USO DE IA

Durante a elaboração deste trabalho, utilizou-se a ferramenta Gemini para a revisão ortográfica e para o auxílio na estruturação do texto. O conteúdo foi revisado, editado e validado pelos autores, que assumem total responsabilidade pelo texto final.

6 REFERÊNCIAS

Anuar AW, et al. Revisiting Web Application Development with Integrated Records Management Important Aspect using Re-CRUD. Journal of Information and Knowledge Management. 2022 Apr;12(1):31-53.

Fadillah AR, et al. Automatic CRUD Code Generation from UML Models for Low-Code Platforms: A Model-Driven Engineering Approach. In: 2025 International Conference on Information Technology Research and Innovation (ICITRI). Jakarta, Indonesia: IEEE; 2025. p. 1-6.

Jin B, et al. Product design reuse with parts libraries and an engineering semantic web for small and medium-sized manufacturing enterprises. The International Journal of Advanced Manufacturing Technology. 2008 Oct;38(11-12):1075-84.

ISSN 2317-1316

Moore RJ, et al. Design Systems for Conversational UX. In: Proceedings of the 2nd Conference on Conversational User Interfaces. Bilbao, Spain: ACM; 2020. p. 1-4.
Protégé. Protégé. Disponível em: https://protege.stanford.edu/. Acesso em: 12 abr. 2026.

Rafi S, et al. DevOps Practitioners' Perceptions of the Low-code Trend. In: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. Helsinki, Finland: ACM; 2022. p. 301-6.

Silega N, et al. An ontology-based approach to support the knowledge management of software quality standards. Enfoque UTE. 2023 Jul;14(3):49-56.

Sommerville I. Engenharia de software. 10. ed. São Paulo: Pearson Education do Brasil; 2018.

Velásquez AP, et al. Systematic Literature Review of Low-Code and Its Future Trends. In: 2024 12th International Conference in Software Engineering Research and Innovation (CONISOFT). Puerto Escondido, Mexico: IEEE; 2024. p. 30-7.

W3C. OWL 2 Web Ontology Language Document Overview. 2. ed. [S. l.]: W3C; 2012. Disponível em: https://www.w3.org/TR/owl2-overview/. Acesso em: 12 abr. 2026.