# Desenvolvimento de uma API REST para Manipulação de Ontologias utilizando Python e OWLReady2: Um Relato de Experiência

**Rafael Silva da Silva**[1], **João Pablo Silva da Silva**[1], **Alice Fonseca Finger**[1]

[1]Laboratory of Intelligent Software Engineering (LabISE)
*Campus* Alegrete – Universidade Federal do Pampa (UNIPAMPA)
Av. Tiarajú, 810 – 97546-550 – Alegrete – RS – Brasil

{rafaelsds3.aluno, joaosilva, alicefinger}@unipampa.edu.br

***Abstract.*** *The Semantic Web enables new ways of organizing and intelligently retrieving information but still lacks accessible mechanisms for ontology manipulation. This report presents the development of a RESTful Web API in Python, using the Owlready2 library, which supports the creation and classification of individuals, the application of semantic rules (SWRL), and logical inferences. The results indicate improvements in interoperability, content reuse, and query accuracy, confirming the potential of this integration to enhance automated information retrieval.*

***Resumo.*** *A Web Semântica possibilita novas formas de organizar e recuperar informações de maneira inteligente, mas ainda carece de mecanismos acessíveis para manipulação de ontologias. Este relato descreve o desenvolvimento de uma API Web RESTful em Python, com a biblioteca Owlready2, que viabiliza criação e classificação de indivíduos, aplicação de regras semânticas (SWRL) e inferências lógicas. Os resultados indicam avanços em interoperabilidade, reutilização de conteúdos e precisão nas consultas, confirmando o potencial dessa integração para aprimorar a recuperação automatizada de informações.*

## 1. Introdução

A evolução da Web tem impulsionado a busca por soluções capazes de organizar e recuperar informações de maneira mais inteligente e eficiente. Nesse cenário, a Web Semântica surge como uma abordagem que permite estruturar dados de forma interpretável por máquinas, permitindo a formalização de conceitos e relações por meio de ontologias [Berners-Lee et al. 2001]. Tais recursos favorecem a interoperabilidade entre sistemas, a integração de dados heterogêneos e a construção de aplicações capazes de executar raciocínios automáticos, resultando em maior precisão e relevância na entrega de informações [Breitman 2005].

As ontologias ocupam papel central na pilha tecnológica da Web Semântica, uma vez que permitem a formalização explícita de conceitos e relações dentro de um domínio de conhecimento [Gruber 2009]. Por meio delas, informações deixam de ser representadas apenas como dados isolados e passam a incorporar significado semântico, tornando-se passíveis de interpretação por máquinas. Esse processo viabiliza a interoperabilidade entre sistemas, o reuso de informações em diferentes contextos e a aplicação de mecanismos de raciocínio automático, como classificação hierárquica e inferências lógicas [Berners-Lee et al. 2001].

Apesar de seu potencial, a adoção prática da Web Semântica enfrenta barreiras significativas. A ausência de ferramentas acessíveis e padronizadas para manipulação de ontologias limita o uso dessas tecnologias em sistemas distribuídos e aplicações do dia a dia. Além disso, a necessidade de integrar mecanismos de inferência lógica em arquiteturas modernas de software exige soluções que conciliem formalismo semântico com praticidade de implementação. Nesse contexto, torna-se fundamental explorar formas de encapsular ontologias em interfaces padronizadas, favorecendo tanto desenvolvedores quanto usuários na construção de sistemas inteligentes e interoperáveis.

Este trabalho tem como objetivo relatar a experiência de desenvolvimento de uma API Web RESTful implementada em Python, utilizando a biblioteca Owlready2. A proposta busca disponibilizar uma camada de serviços capaz de realizar operações sobre ontologias, incluindo a criação e classificação de indivíduos, a aplicação de regras semânticas e a execução de raciocínios lógicos com o auxílio de motores de inferência. A metodologia adotada seguiu princípios de arquitetura de software baseada em serviços, utilizando boas práticas de modularização e camadas de abstração, provendo uma interface simples e padronizada, voltada à interoperabilidade, ao reuso de informações e à integração com sistemas externos.

A experiência demonstrou que a integração entre APIs RESTful e tecnologias semânticas pode contribuir significativamente para ampliar a interoperabilidade entre repositórios, facilitar a reutilização de conteúdos e aumentar a eficiência das consultas semânticas. Além disso, observou-se que o uso da Owlready2 reduziu a complexidade no acesso e manipulação das ontologias, tornando a implementação mais intuitiva e produtiva para desenvolvedores. Assim, este trabalho contribui ao apresentar um relato prático que aproxima os conceitos da Web Semântica de aplicações reais, oferecendo uma solução que combina escalabilidade, padronização e inteligência na recuperação de informações.

O restante do trabalho está organizado da seguinte forma: a Seção 2 apresenta os conceitos importantes que fundamentam o trabalho; a Seção 3 discute trabalhos que também desenvolveram API Web RESTful em Python, com a biblioteca Owlready2; a Seção 4 apresenta um relato da experiência no desenvolvimento da API com uso da Owlready2; a Seção 5 expõe os resultados obtidos no desenvolvimento do trabalho; e, por fim, a Seção 6 apresenta as conclusões, destacando as contribuições, as limitações e os trabalhos futuros.

## 2. Fundamentação

O desenvolvimento de soluções baseadas em Web Semântica depende de uma sólida fundamentação em ontologias, responsáveis por estruturar e dar significado às informações. Essa combinação permite a criação de sistemas capazes de raciocinar e inferir novos conhecimentos. Para tornar esse processo acessível ao desenvolvimento de software, a biblioteca OWLReady2 oferece recursos que simplificam a integração de ontologias em aplicações Python, sendo um componente central para este trabalho.

A Web Semântica foi Idealizada por Tim Berners-Lee como uma extensão da Web atual. Ela tem como objetivo tornar os conteúdos publicados não apenas legíveis, mas também processáveis por máquinas. Sua proposta central é estruturar as informações de maneira padronizada e formal, permitindo que sistemas sejam capazes de interpretar e relacionar dados de forma inteligente. Dessa forma, agentes de software podem raciocinar

sobre o conteúdo disponibilizado, atribuindo-lhe significados bem definidos. Isso possibilita avanços como mecanismos de busca mais precisos, recomendações personalizadas e uma Web efetivamente interoperável [Berners-Lee et al. 2001].

As ontologias são estruturas formais que definem entidades, suas propriedades e os relacionamentos existentes em um determinado domínio, com o objetivo de descrever semanticamente o conhecimento presente nesse contexto [Gruber 2009]. Elas fornecem um vocabulário compartilhado e regras explícitas que permitem a interpretação consistente de informações por humanos e sistemas computacionais. Além disso, ontologias possibilitam a integração de dados heterogêneos, a inferência automática de novos conhecimentos e o suporte a aplicações inteligentes, como sistemas de recomendação, agentes semânticos e Web Semântica [Carlan 2006].

A Web Ontology Language (OWL) é um padrão do W3C projetado para representar de forma formal conceitos, classes e relações em um domínio de conhecimento. Baseada em lógica descritiva, a OWL permite não apenas descrever estruturas conceituais, mas também realizar raciocínios automáticos, como verificação de consistência, classificação hierárquica e inferência de novos fatos. Sua expressividade e compatibilidade com outros padrões da Web Semântica a tornam fundamental para a construção de sistemas inteligentes e interoperáveis [McGuinness and Harmelen 2004].

A OWLReady2[1] é uma biblioteca Python que propõe o paradigma de programação orientada a ontologias (ontology-oriented programming), realizando uma "tradução" dinâmica entre os elementos de uma ontologia OWL e objetos/classe em Python. Em termos práticos, classes OWL se tornam classes Python, indivíduos são instâncias dessas classes e propriedades OWL são acessadas como atributos. Esse mapeamento de alto nível reduz a distância semântica entre o formalismo lógico e o código, tornando as operações de criação, consulta e atualização de ontologias mais naturais para quem programa em Python [Lamy 2017].

No plano conceitual, a OWLReady2 trabalha sobre um mundo que carrega uma ou mais ontologias, expondo uma API que preserva os princípios da OWL 2 e, ao mesmo tempo, oferece operações idiomáticas de Python para manipular classes, indivíduos e propriedades. Assim, a definição de propriedades de dados (DataProperty) e propriedades de objetos (ObjectProperty), bem como o uso de restrições (por exemplo, minQualifiedCardinality, only, some) podem ser expressos tanto no arquivo OWL (em RDF/XML ou TTL) quanto diretamente em Python, mantendo a consistência com o modelo formal [Lamy 2017].

Um dos principais diferenciais da biblioteca está na sua capacidade de integrar-se a raciocinadores baseados em lógicas descritivas, como HermiT e Pellet, por meio de funções específicas, por exemplo, a sync_reasoner(). Esse recurso possibilita não apenas a verificação de consistência da ontologia, identificando contradições ou incoerências na modelagem conceitual, mas também a classificação automática das classes, organizando-as em hierarquias mais precisas de acordo com os axiomas definidos. Além disso, permite a inferência de novos conhecimentos, como a atribuição de tipos a indivíduos, a propagação de restrições e a descoberta de relações implícitas que não estavam explicitamente descritas no modelo [Lamy 2017].

[1]Disponível em: `https://owlready2.readthedocs.io/en/v0.48/`

## 3. Trabalhos Relacionados

Diversos estudos exploram a integração de ontologias e serviços Web. Por exemplo, [Wang et al. 2020] apresenta uma API voltada ao desenvolvimento de um sistema de recomendação em saúde, fornecendo materiais educacionais personalizados a pacientes com doenças crônicas na China, utilizando Owlready2 para manipulação da ontologia que sustenta as recomendações.

Outros trabalhos avançam na incorporação de semântica em sistemas de e-learning. O estudo de [Rabahallah and Ahmed-Ouamer 2015] propõe uma arquitetura de serviços Web semânticos, reutilizando funcionalidades como autenticação, gerenciamento de cursos e avaliações, e viabilizando a descoberta e composição automática de serviços. De forma complementar, [Júnior et al. 2022] apresenta uma abordagem híbrida para recomendação personalizada de Objetos de Aprendizagem Digitais (OADs), integrando algoritmos bioinspirados e técnicas de Web Semântica para combinar diferentes fontes (AVAs, YouTube, Wikipédia), representando metadados por meio de ontologias e promovendo recomendações alinhadas ao perfil do estudante. Apesar da relevância dessas propostas, muitas permanecem conceituais ou não apresentam avaliações empíricas robustas.

Diante das iniciativas identificadas na literatura, este trabalho se diferencia por apresentar uma experiência prática de implementação que vai além de propostas conceituais. Essa abordagem oferece uma camada de serviços reutilizável e padronizada, que integra mecanismos de inferência semântica e classificação automática, ampliando a interoperabilidade entre repositórios e a eficiência na recuperação de informações. Assim, o trabalho avança em direção à aplicabilidade efetiva da Web Semântica em sistemas distribuídos, apoiado por resultados empíricos que demonstram benefícios concretos em termos de desempenho e reuso de informações.

## 4. Relato de Experiência

Nesta seção é relatado o processo de concepção e implementação da API, enfatizando a metodologia adotada, as ferramentas empregadas e as estratégias de design utilizadas. A descrição contempla a configuração do ambiente de desenvolvimento, a escolha de bibliotecas e *frameworks*, bem como a forma como a integração com raciocinadores semânticos foi incorporada ao sistema. Esse panorama fornece a base para compreender os resultados obtidos e as contribuições apresentadas nas seções seguintes.

A OntoObADi [2]. Sua adoção possibilitou aplicar na prática os recursos da API desenvolvida, como a criação e classificação de indivíduos, a definição de propriedades e a execução de inferências semânticas. Para um melhor entendimento da ontologia, é apresentado na Figura 1 o modelo conceitual da OntoObADi.

O desenvolvimento da API foi pautado pela adoção de ferramentas modernas que promovem produtividade, reprodutibilidade e alto desempenho. Para o gerenciamento de dependências, empacotamento e isolamento de ambiente, utilizou-se o *Poetry*, que garantiu que todas as bibliotecas estivessem em suas versões corretas e compatíveis. A implementação da interface RESTful foi realizada com o *FastAPI*, escolhido por sua alta performance (baseada no *framework* ASGI Starlette), pela tipagem automática com

[2]Disponível em: `https://zenodo.org/communities/ontoobadi/`

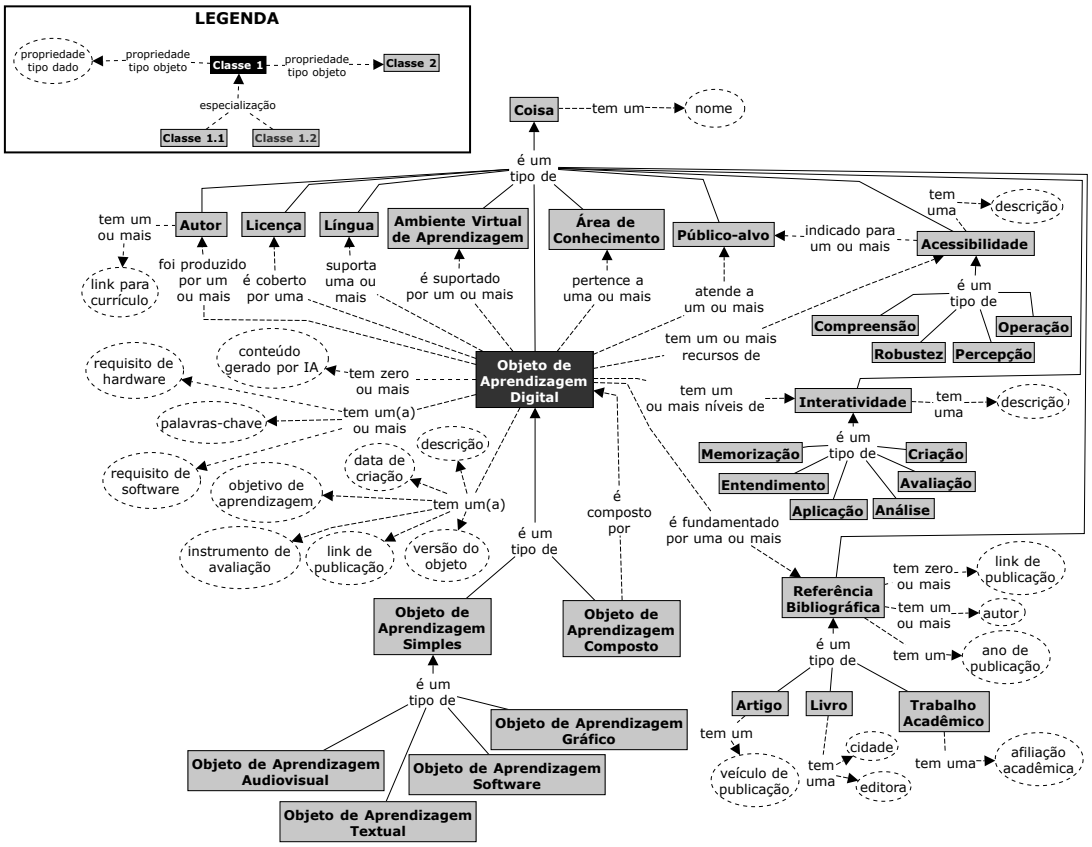


**Figura 1. Modelo Conceitual da ontologia OntoObADi.**

Pydantic e pela geração nativa de documentação interativa via Swagger/OpenAPI. Já a manipulação da OntoObADi foi viabilizada pela biblioteca *OWLReady2*, considerada o núcleo da solução, uma vez que permitiu realizar inferências, persistir informações e adotar um paradigma de programação orientada a ontologias diretamente em Python.

## 4.1. Arquitetura da API

API foi desenvolvida seguindo os princípios do estilo arquitetural *Representational State Transfer* (REST), um estilo amplamente consolidado em sistemas distribuídos, valorizado por sua simplicidade, escalabilidade e compatibilidade nativa com os protocolos da Web [Xiao-Hong 2014]. Essa opção arquitetural permite que as operações sejam realizadas por meio dos métodos convencionais do protocolo *Hypertext Transfer Protocol* (HTTP), facilitando a interação de aplicações externas com a OntoObADi de forma intuitiva e padronizada. A troca de dados é realizada integralmente no formato *JavaScript Object Notation* (JSON), tanto para entradas quanto para saídas.

Do ponto de vista de implementação, a arquitetura da API segue o padrão MVC (Model-View-Controller), ampliado com uma camada de serviços (Service Layer) dedicada. A estrutura completa dessa arquitetura é ilustrada na Figura 2.

A camada *Models* é responsável por encapsular as entidades centrais do domínio do problema. Nela são definidas as classes e estruturas de dados que mapeiam diretamente os conceitos da ontologia, como *Individual*, *BaseOntologyModel*, *Rule* e *ManagerOntology*. Essa organização promove um desenvolvimento mais limpo, modular e alinhado

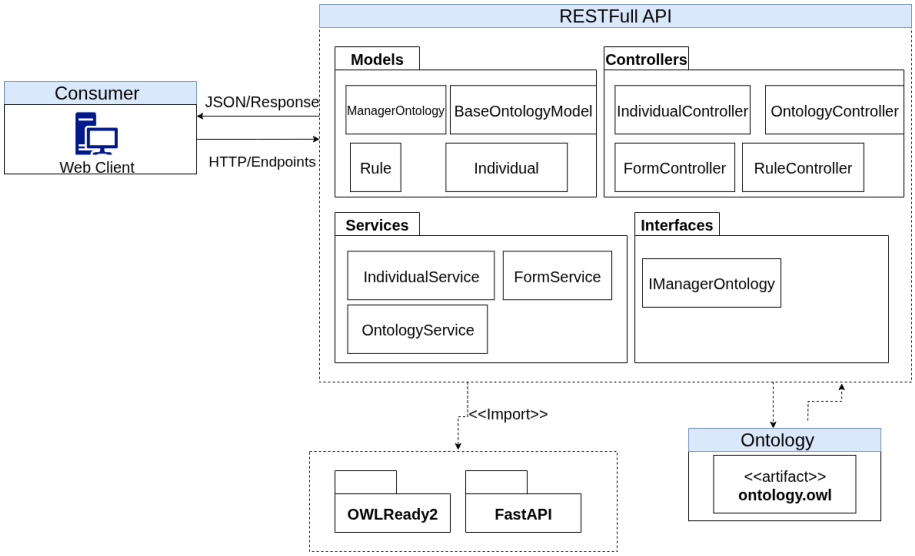


**Figura 2. Diagrama de pacotes da *Application Programming Interface* (API).**

com os princípios de orientação a objetos. A camada *Controllers* atua como intermediária entre as requisições recebidas pela API e as demais camadas. Sua principal responsabilidade é direcionar o fluxo das ações, processar entradas, invocar a camada de serviços adequada e retornar as respostas correspondentes no formato esperado, garantindo assim uma separação clara de responsabilidades.

A camada *Services*, encapsula a lógica de negócio complexa e específica do domínio, orquestrando operações, aplicando regras e coordenando interações entre modelos. Para manter os *controllers* enxutos e focados em sua função de mediação, utiliza-se a injeção de dependências (*Dependency Injection*). Essa técnica permite que os *controllers* recebam instâncias de serviços como dependências em seus contrutores. Com isso, a responsabilidade de gerenciar a lógica de negócio é delegada aos serviços. A camada *Interfaces* implementa o princípio de Inversão de Dependência, um dos pilares do SOLID, com o objetivo de abstrair completamente o acesso à OntoObADi Ela promove o desacoplamento entre os componentes de alto nível da aplicação (que definem o que precisam fazer) e os módulos de baixo nível (que implementam como as operações ontológicas são realizadas).

Esse desacoplamento é concretizado por meio de um contrato formal, assim estabelecendo os métodos essenciais para interação com a ontologia. Dessa forma, os componentes consumidores (como *Services* ou *Controllers*) dependem apenas dessa abstração e não de implementações concretas. Isso possibilita que a fonte ou a tecnologia de persistência da ontologia (ex.: trocar arquivos OWL locais por um endpoint SPARQL ou uma API remota) seja substituída sem que seja necessário alterar qualquer código nos módulos de alto nível que a consomem.

### 4.2. Utilização da OWLReady2

Logo nas fases iniciais do desenvolvimento, tornou-se evidente uma das mais significativas vantagens proporcionadas pela adoção da OWLReady2: a notável facilidade para interagir e manipular a ontologia de forma programática. Essa facilidade se manifesta concretamente através de uma sintaxe intuitiva e pythonica, permitindo realizar operações

complexas com comandos concisos e de alta legibilidade. Por exemplo, para carregar uma ontologia, bastam poucas linhas de código:

```
from owlready2 import get_ontology

# Carregando a ontologia
onto = get_ontology("caminho/para/ontologia.owl").load()
```

Com esse simples trecho, temos uma instância completa da ontologia, incluindo todas as classes definidas no arquivo *.owl* ilustradas na Figura 1. A OWLReady2 também facilita a criação de indivíduos e a manipulação de propriedades de forma direta e intuitiva. Por exemplo, é possível criar um indivíduo da classe *Author* e atribuir valores às suas propriedades com poucas linhas:

```
# Instanciando individuo
with onto:
    individual = Author("autor_teste")
    individual.hasResumeLink = ["http://lattes.cnpq.br
       /7433133067872365"]
    onto.save(file="caminho/para/ontologia.owl", format="rdfxml"
       )
```

Além da criação de indivíduos, a biblioteca oferece suporte nativo para explorar a estrutura ontológica, permitindo listar de forma simples tanto as *data properties* quanto as *object properties* definidas no modelo. Isso é extremamente útil para compreender e manipular o conteúdo da ontologia de maneira programática:

```
#Listando data_properties e object_properties
with onto:
     for data_properties in onto.data_properties():
        print(data_properties)
     for object_properties in onto.object_properties():
        print(object_properties)
```

Por fim, um dos recursos mais poderosos da OWLReady2 é a integração com raciocinadores automáticos, como o *HermiT* e o *Pellet*. Esses mecanismos permitem a inferência de novos fatos a partir das definições existentes na ontologia, enriquecendo o conhecimento representado sem necessidade de intervenção manual. O exemplo a seguir demonstra a execução de um raciocinador para atualizar a hierarquia de classes e os relacionamentos entre indivíduos:

```
from owlready2 import sync_reasoner
#Executando raciocinador
with onto:
    sync_reasoner()
```

Após a exploração prática dos recursos oferecidos pela *OWLReady2*, tais funcionalidades foram incorporadas à camada de *Models*, sendo encapsuladas pela classe `ManagerOntology`. Essa classe implementa a interface `IManagerOntology`, a qual estabelece um contrato bem definido para todas as operações de acesso e

manipulação da ontologia. Com esse design, a complexidade inerente à OWLReady2 permanece isolada, garantindo que os demais módulos da aplicação não dependam diretamente da biblioteca ou da estrutura interna da ontologia. Dessa forma, a lógica de negócio interage apenas com a interface, de maneira desacoplada e transparente, preservando os princípios de modularidade e manutenibilidade do sistema. O diagrama de classes apresentado na Figura 3 a seguir ilustra claramente essa organização.

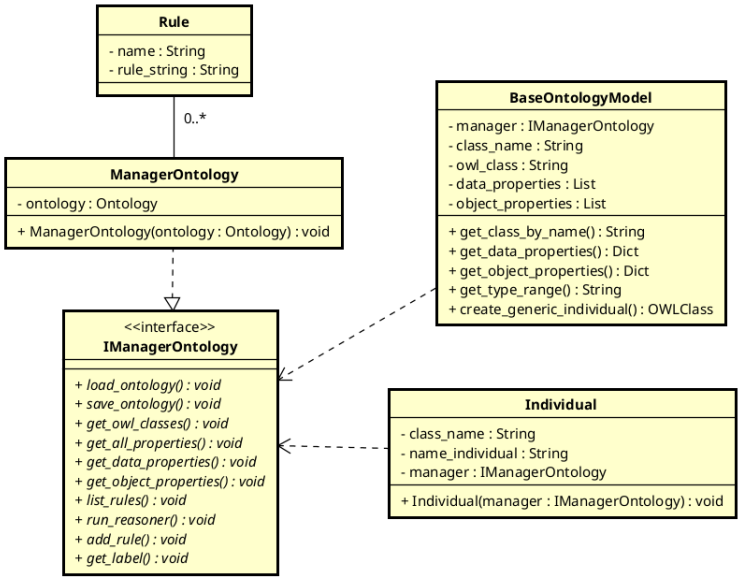


**Figura 3. Diagrama de classe da *Application Programming Interface* (API).**

## 5. Resultados

A implementação da API possibilitou encapsular a OntoObADi de forma a permitir sua manipulação via serviços RESTful. Foram desenvolvidos *endpoints* como por exemplo o *endpoint GET/createSchema*, que retorna um *JavaScript Object Notation* (JSON) estruturado contendo *data_properties* e *object_properties* assim como suas labels definidas na OntoObADi,como mostra a Figura 4.

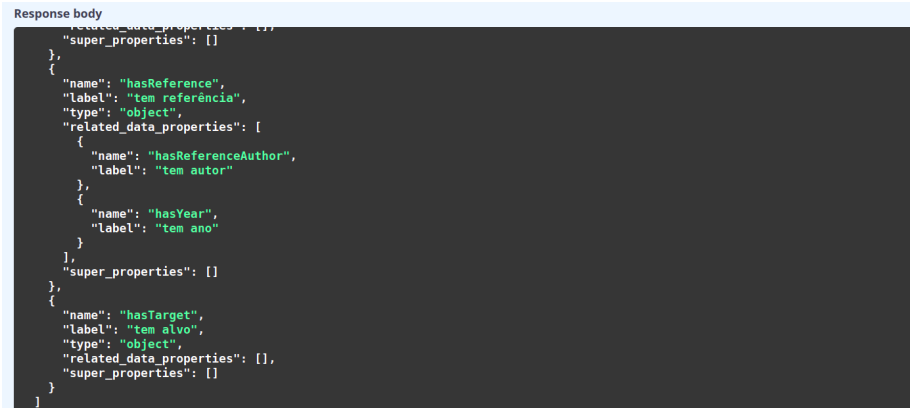


**Figura 4. Retorno do Endpoint.**

Esse esquema dinâmico serve como base para a construção automática de formulários, permitindo que sistemas clientes criem interfaces de cadastro e edição de conteúdo educacional de forma padronizada. A Figura 5 exemplifica o consumo desse *endpoint* em um formulário multi-step.

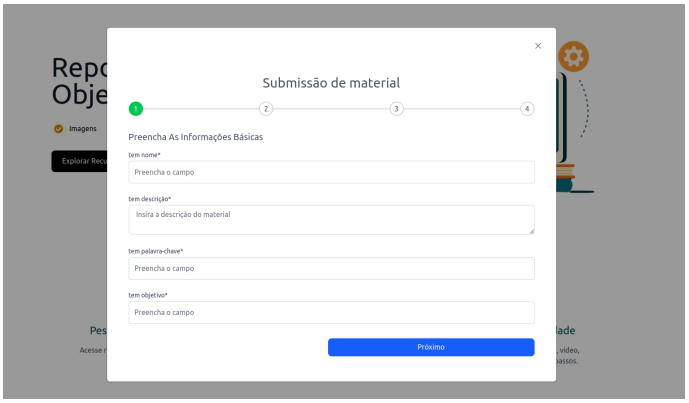


**Figura 5. Web Client Consumindo o Endpoint.**

Outro cenário relevante é a criação e classificação automática de OADs. A *Application Programming Interface* (API) instancia dinamicamente indivíduos na OntoObADi, por meio da classe `Individual` e do *endpoint* `POST/createindividual`, que espera os seguintes parâmetros: o campo `name`, que indica o nome do indivíduo a ser criado; e dois blocos de propriedades: `data_properties`, destinado a valores literais, e `object_properties`, conforme ilustrado na Figura 6.

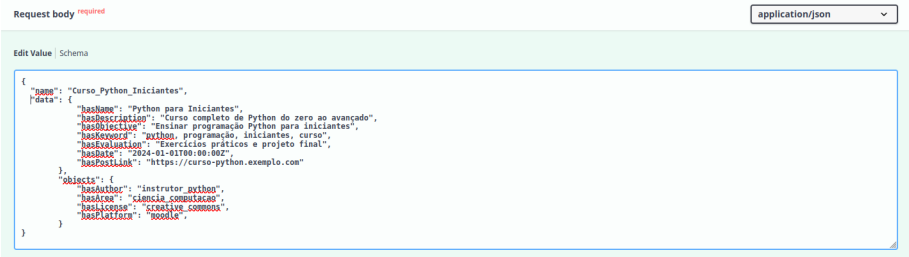


**Figura 6. Parametros do Enpoint.**

Dessa forma, o processo garante o retorno de um indivíduo criado, instanciado e classificado dinamicamente de forma automática na ontologia.

## 6. Conclusão

Durante o desenvolvimento, a utilização da OWLReady2 se mostrou um diferencial, permitindo a abstração de tarefas complexas relacionadas ao acesso e manipulação da ontologia. A experiência prática revelou tanto benefícios como a simplicidade na definição de propriedades e a flexibilidade na execução de consultas e inferências, quanto desafios, como a necessidade de tratar limitações de desempenho em raciocínios mais extensos e a adaptação da estrutura ontológica ao modelo relacional de uma API REST. Os exemplos de uso obtidos até o momento indicam que a API, aliada à OWLReady2, é capaz de apoiar desde a construção dinâmica de interfaces educacionais até a recomendação personalizada

de conteúdos, evidenciando seu potencial em contribuir com soluções pedagógicas mais inteligentes e centradas no aluno.

Como trabalho futuro, propomos a criação de um repositório educacional baseado em Web Semântica, voltado à classificação e recomendação de OADs. Acredita-se que essa iniciativa contribuirá significativamente para a organização e o compartilhamento de conteúdos educacionais, potencializando a reutilização e a interoperabilidade entre diferentes plataformas. Com este trabalho, espera-se não apenas validar a aplicação prática da OADs, mas também incentivar a adoção da OWLReady2 e de tecnologias semânticas no contexto educacional, promovendo buscas mais eficientes e o desenvolvimento de aplicações educacionais inteligentes.

## Agradecimentos

Os autores agradecem à FAPERGS pelo apoio financeiro (Projeto ARD/ARC - processo 23/2551-0000761-4) e ao CNPq pela concessão de bolsa de Iniciação Científica.

## Referências

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):35–43. Disponível em: `https://www.lassila.org/publications/2001/SciAm.html`. Acesso em: 01 set. 2024.

Breitman, K. K. (2005). *Web semântica*. LTC, Rio de Janeiro. recurso online. Disponível em: `https://app.minhabiblioteca.com.br/reader/books/978-85-216-1958-1`. Acesso em: 25 maio 2023.

Carlan, E. (2006). Ontologia e web semântica. *Brasília: Universidade de Brasília*.

Gruber, T. (2009). *Ontology*, page 3748. Springer-Verlag.

Júnior, C. P., Araújo, R., and Dorça, F. (2022). Uma abordagem híbrida apoiada por algoritmo bioinspirado e tecnologias de web semântica para recomendação personalizada de objetos de aprendizagem. In *Anais Estendidos do XI Congresso Brasileiro de Informática na Educação*, pages 35–46, Porto Alegre, RS, Brasil. SBC.

Lamy, J.-B. (2017). Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28.

McGuinness, D. L. and Harmelen, F. V. (2004). Owl web ontology language overview. Dispon{í}vel em: `{https://goo.gl/p0g6aq`. Acesso em: 30 de junho de 2017.

Rabahallah, K. and Ahmed-Ouamer, R. (2015). Creating e-learning web services towards reusability of functionalities in creating e-learning systems. In *2015 Global Summit on Computer Information Technology (GSCIT)*, pages 1–6.

Wang, Z., Huang, H., Cui, L., Chen, J., An, J., Duan, H., Ge, H., and Deng, N. (2020). Using natural language processing techniques to provide personalized educational materials for chronic disease patients in china: Development and assessment of a knowledge-based health recommender system. *JMIR Med Inform*, 8(4):e17642.

Xiao-Hong, L. (2014). Research and development of web of things system based on rest architecture. In *2014 Fifth International Conference on Intelligent Systems Design and Engineering Applications*, pages 744–747.