AgroGraphAPI: uma API para armazenamento e recuperação de dados agrícolas

Anderson dos Santos da Rosa[1], **Ana Paula Lüdtke Ferreira**[1]

[1]Bacharelado em Ciência da Computação – Campus Alegrete
Universidade Federal do Pampa

[2]Programa de Pós-graduação em Computação Aplicada – Campus Bagé
Universidade Federal do Pampa

{andersonrosa.aluno,anaferreira}@unipampa.edu.br

Resumo. *A evolução da Agricultura Digital exige a construção de sistemas que atendam às necessidades dos* stakeholders *dos sistemas produtivos. Interfaces para Programação de Aplicações (API) são serviços* web *que oferecem funcionalidades que podem ser adicionadas a quaisquer sistemas, independente das tecnologias usadas em sua construção. Este trabalho apresenta uma API para registro e recuperação eficiente de dados provenientes dos sistemas produtivos, usando um banco de dados orientado a grafos. Os texto descreve o modelo de dados, a arquitetura da API, a descrição dos* endpoints *e o formato das requisições. A API pode ser usada por qualquer interessado em manter um banco de dados de valores coletados, sem necessidade de implementação dos procedimentos de criação e consulta do banco de dados.*

Abstract. *The evolution of Digital Agriculture requires information systems that meet the needs of production systems' stakeholders. Application Programming Interfaces (API) are web services that offer functionalities that can be added to other systems, regardless of their building technologies. This work presents an API for efficient agricultural data recording and retrieval, using a graph database as the supporting technology. The text describes the data model, the API's architecture, the available endpoints, and the expected format of requests. The AgroGraphAPI is useful to anyone interested in maintaining set of collected data without implementing the database's creation and query procedures.*

1. Introdução

A evolução dos sistemas produtivos agropecuários acompanhou os processos industriais: do maquinário movido a tração animal ou humana, passou-se ao trabalho mecanizado, informatização dos processos e, hoje, tem-se à disposição toda a miríade de soluções

ligadas à Indústria 4.0 que, no meio rural, chama-se de Agricultura Digital ou Agricultura 4.0 [Massruhá and Leite 2017].

A possibilidade do uso da tecnologia disponibilizou aos proprietários rurais uma variedade de aplicativos, que apresentam diferenças nas interfaces, formas de organização da informação, linguagens de programação e sistemas gerenciadores de bancos de dados, sem qualquer comunicação entre si [Bernardi et al. 2014].

A evolução da Agricultura 4.0 exige que a tecnologia seja integrada em dispositivos físicos, com interfaces que façam sentido para quem delas fará uso [Shepherd et al. 2018]. Sistemas produtivos diferentes têm necessidades distintas e a integração dos sistemas tecnológicos é requisito para um controle completo da atividade. Contudo, são muito poucos os produtores, no contexto nacional, que possuem condições de custear sistemas integrados e customizados de gestão de processos e produtos [Bolfe et al. 2020].

Interfaces de Programação de Aplicações (ou simplesmente API, do termo em Inglês *Application Programming Interfaces*) podem contribuir para um processo de desenvolvimento de sistemas mais efetivos e com menor custo. API são serviços *web* que disponibilizam funções por meio de endereços únicos no formato *Uniform Resource Identifier* (URI) que podem ser acessadas por meio do protocolo HTTP [Arcuri 2019]. A vantagem do uso de API é o fraco acoplamento existente entre funcionalidade e aplicação, permitindo que sistemas construídos com diferentes tecnologias possam acessar um conjunto de recursos sem preocupação com detalhes de implementação [Subramanian and Raj 2019]. Funcionalidades existentes em diferentes API podem ser incorporadas a uma aplicação, por meio da interface disponibiizada pela API, agilizando seu desenvolvimento. Exemplos de API correntemente usadas são os processos de *login* através do *Google* ou do *Facebook* e os serviços de localização do *Google Maps*, fornecidos pela API da Google.

O objetivo deste trabalho é apresentar uma API para registro e recuperação eficiente de dados provenientes dos sistemas produtivos, que disponibiliza as funcionalidades do sistema AgroGraph [Rosa et al. 2022]. O sistema AgroGraph opera com chamadas processadas em PHP para conexão direta ao banco de dados de grafos que armazena as informações agrícolas de produtores, de forma flexível. As operações de *login*, armazenamento e busca passaram a ser executadas pela API (denominada AgroGraphAPI) e são disponibilizadas aos interessados em integrá-las aos seus próprios sistemas de informação usando as requisições da API, por meio de sua interface.

O restante do texto está estruturado como se segue: a Seção 2 discute os fundamentos teóricos das funcionalidades disponibilizadas pela AgroGraphAPI, a Seção 3 apresenta a arquitetura e o funcionamento da API, a Seção 4 discute os resultados e a Seção 5 apresenta as conclusões do trabalho.

2. Modelos e tipos de dados

A AgroGraphAPI opera com um banco de dados de grafos [Robinson et al. 2015]. A escolha cnsiderou características dos dados produzidos em sistemas produtivos agropecuários: os dados têm propriedades espaciais e/ou temporais, exigindo georreferenciamento e registro de tempo de coleta; o conjunto de atributos de uma entidade não é fixo

e nem limitado; um dado necessita de várias chaves de acesso e informações importantes podem não ser únicas; valores são coletados em diferentes pontos de tempo e espaço; novos atributos podem ser criados a qualquer momento; o acesso aos dados é bidirecional; o valor das variáveis coletadas em determinado local e quais são os locais que possuem um determinado valor para um conjunto de variáveis são perguntas frequentes.

Os bancos de dados relacionais podem ser adaptados para lidar com esses requisitos, mas a adaptação resulta em uma coleção dispersa de tabelas com poucas colunas, em uma estrutura chave-valor ou similar [Fiss et al. 2020]. A realização de consultas usualmente requer operações de junção entre tabelas, com alta complexidade computacional [Silberschatz et al. 2006]. Os modelos de dados baseados em grafos possibilitam consultas eficientes de relações, sem necessidade de operações adicionais no banco.

A partir dos anos 1990, bases de dados fora do padrão relacional foram propostas, buscando flexibilizar seus esquemas rígidos de descrição de dados. Conhecidas como *Not Only SQL* (NOSQL), essas bases permitem que novos tipos de dados possam ser associados com registros já existentes. Em um banco de dados de grafos, nodos representam os valores e arestas representam relações, que podem ser rotulados com propriedades ou valores. A inserção de nodos, arestas ou propriedades não requer alteração nos dados já armazenados [Robinson et al. 2015].

O banco de dados usado para implementar as funcionalidades da API foi o Neo4J 4.4.7 [Neo4J, Inc. 2020], escolhido por existir uma versão gratuita que suporta as propriedades ACID (*atomicidade*, *consistência*, *isolamento* e *durabilidade*) para as transações do banco, além de já ser usado em diferentes domínios de aplicação. O sistema AgroGraph [Rosa et al. 2022] faz uso desse banco, por causa da forte estrutura relacional existente entre os elementos do Sistema Solo-Planta-Animal-Atmosfera, então foi uma escolha natural.

A organização do modelo de dados considerou as características dos dados coletados a respeito da produção vegetal e de dados podológicos e meteorológicos. Especificamente, os dados coletados possuem uma *designação* (nome da variável), um *valor*, uma *unidade de medida*, o *local* e o *período* da coleta. Considera-se que todos os dados coletados apresentam um mínimo de quatro atributos: sua localização georreferenciada, a data de captura do dado, seu valor e a unidade de medida. Em alguns casos, o horário da captura do dado também é relevante. Por exemplo, no caso de dados meteorológicos que podem variar ao longo do dia. Nesse caso, a representação do horário é feita como uma propriedade do nodo medição, permitindo que várias coletas ao longo de um dia sejam armazenadas. A Tabela 1 sumariza os tipos e as representações dos dados aqui descritos.

Tabela 1. Atributos dos dados armazenados no banco

Aspecto	**Tipo**	**Representação**
Espacial	Ponto	latitude e longitude
Temporal	Data	dd/mm/aaaa hh:mm:ss
Dado	Variável	tipo (*string*), valor (*float*) e unidade de medida (*string*)

Os três tipos de armazenamento são traduzidos em nodos e arestas para a inserção e recuperação na base de dados. Para a criação do modelo de representação utilizou-se a premissa de que podem ser realizadas consultas por localização, variável observada, e data

da observação. Assim, optou-se pela modelagem utilizando três nodos para representar cada um dos tipos: localização, data, variável. Como o sistema prevê controle de acesso aos dados, é necessário armazenar informações sobre o usuário para garantir limitações de visibilidade. A informação de propriedade do dado é feita em um nodo do banco, com os atributos necessários. Como o Neo4j não implementa o conceito de hiperarco (ou hiperaresta) [Habel 1992] que pode ser usado para ligar múltiplos nodos simultaneamente, foi definido o nodo *medição*, que sumariza a ideia de um ponto de coleta no tempo e no espaço. A Figura 1 apresenta o modelo de dados desenvolvido para armazenar os dados descritos anteriormente. Note-se que o modelo de dados é um grafo que atua como a tipagem dos demais nodos e arestas que representam os valores do banco, à semelhança dos grafos tipados em gramáticas de grafos [Ehrig et al. 1996].

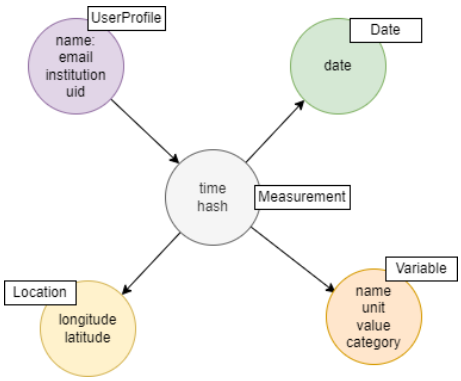

Figura 1. Modelo definido para armazenar os dados no AgroGraphAPI

A data de cada medição foi modelada como um nodo próprio devido à propriedade de adjacência livre de índice do Neo4j, em que cada nodo contém uma referência direta para os seus nodos vizinhos [Ian Robinson and Eifrem 2015]. Dessa forma, quando uma pesquisa por data ocorre todas as medições feitas na data especificada estarão acessíveis diretamente através do nodo Data sem a necessidade de percorrer todos os nodos Medição comparando suas datas com a data buscada .

Para identificar cada novo nodo Medição é gerada uma *hash* usando como entrada: o identificador do usuário, horário e data da medição, latitude e longitude da medição. Quando uma medição é inserida no banco de dados é checado previamente se já existe um nodo para o mesmo local, momento e usuário através da *hash*, caso já exista a variável é inserida no nodo existente se não um novo nodo é criado. Isso não seria necessário na versão paga da ferramenta, em que elementos podem ser identificados como únicos.

Todas as operações do banco são implementadas com a linguagem de consulta Cypher [Neo4J 2020] e estão listadas na documentação do sistema.

3. AgroGraphAPI

A API e seu funcionamento são apresentados a partir das perspectivas de gerenciamento de usuários, da coleta dos dados e das tecnologias usadas no seu desenvolvimento.

A AgroGraphAPI foi desenvolvida usando o *web framework Representational State Transfer* (REST) *Django* [Django Software Foundation 2023]. O acesso ao banco de dados *Neo4j* é feito por meio do *Object Graph Mapper Neomodel* que mapeia nodos e arestas do banco de dados para objetos da linguagem de programação Python

[Edwards 2019]. Para realizar as operações no plano de coordenadas georreferenciadas é usado o pacote para Python *shapely* [Gillies 2023]. A AgroGraphAPI não atende totalmente ao padrão RESTful, já que não segue a restrição REST da interface uniforme, visto que também disponibiliza métodos (filtragem de dados e krigagem, futuramente) além dos recursos usuais do padrão.

A arquitetura do *framework* Django chama-se MVT (*Model, Views, and Template*). Nesta arquitetura, os elementos relacionados ao modelo, às visões e ao desenho das páginas estão logicamente separados. A conexão com o banco de dados é definida e operada dentro do elemento *Model*. A parte *View* contém as regras de negócio e não abriga o *design* da página, somente organiza a lógica e chama a renderização do *Template*. O *Template*, por sua vez, possui o código de marcação com a parte visual que será apresentada ao usuário. Arquitetura MVT às vezes é referenciada como MTV (*Model, Template, and Views*), mas são termos intercambiáveis.

A AgroGraphAPI não faz uso do módulo *Template*, visto que nossa API não objetiva renderizar páginas – eventuais páginas serão renderizadas pelos sistemas construídos usando as funcionalidades disponibilizadas. A AgroGraphAPI usa o formato *JavaScript Object Notation* (JSON) para retornar os dados requisitados e as mensagens de controle. O formato JSON (`http://json.org/`) usa texto puro para troca de informações e tornou-se popular por ter uma legibilidade maior para humanos e processos de reconhecimento (*parsing*) mais simples do que os modelos de troca de informação baseados em *Extensible Markup Language* (XML). A arquitetura da API está ilustrada na Figura 2.

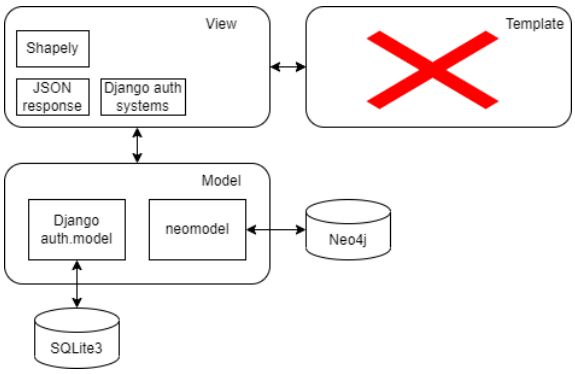

Figura 2. Arquitetura do AgroGraphAPI

O sistema de autenticação usado na API é o sistema nativo do *Django*, que armazena usuários e *tokens* no banco de dados *SQLite3*. Para cada usuário criado pelo sistema de autenticação é criado novo um perfil de usuário no banco de dados *Neo4j*. Para executar a API é necessário instalar o banco de dados *Neo4j* 4.4 e os seguintes pacotes (com uso do instalador de pacotes *pip*): *django*, *djangorestframework*, *django_neomodel* e *shapely*. Outra opção é montar os *containers* disponíveis em `https://github.com/Andersonsr/agrograph-api/tree/main/dockerfiles`, já prontos para uso.

A API disponibiliza *endpoints* para cadastro e *login* de usuários. As formas de autenticação adotadas são as seguintes: (i) autenticação por sessão para usuários acessando diretamente a API e (ii) autenticação por *token* para possibilitar a comunicação entre servidores.

Cada usuário criado pelo sistema de autenticação do Django é relacionado com um perfil de usuário no banco de dados Neo4j. Desta forma é possível guardar informações relativas ao usuário além das informações normalmente armazenadas pelo sistema de autenticação. As relações dos nodos de usuários com os demais nodos no banco são criadas a partir das operações que o usuário realiza.

O *endpoint* sign-in/ é usado para criar novos usuários. A requisição para este *endpoint* deve conter os seguintes campos: *name*, *email*, *password*, *institution*. Quando a requisição é realizada corretamente são criados um usuário Django e um perfil no banco de dados Neo4j. Ambos usuários possuem o mesmo email, possibilitando que o perfil no Neo4j seja encontrado uma vez que o usuário Django seja autenticado com sucesso.

O *endpoint* login/ é usado para autenticar usuários. A requisição deve conter os campos email e *password*. Caso as credenciais sejam válidas, será iniciada uma sessão para o usuário ou um *token* de acesso será retornado caso a autenticação usando *tokens* seja necessária, como no caso de comunicação entre servidores. O *token* retornado é gerado pelo próprio sistema de autenticação do Django e não possui uma data de expiração. O *endpoint* logout/ é usado para encerrar a sessão do usuário.

O *endpoint* edit-profile/ é usado para alterar os dados do usuário já autenticado. A requisição deve conter os mesmos campos que o *endpoint* login/. O *endpoint* altera as informações do usuário Django e no perfil criado no Neo4j com os dados passados na requisição.

A inserção e a busca dos dados apropecuários inseridos usam dois *endpoints* distintos: insert/ e measurements/. Para usar qualquer um dos dois o usuário deve estar autenticado, podendo encontrar apenas as medições que o mesmo usuário inseriu previamente no sistema. Esse controle é herdado do sistema AgroGraph, que possibilita o inserção de dados de diferentes usuários, mas controla a visibilidade dos dados inseridos.

Para inserir novos valores de dados/medições o usuário deve realizar uma requisição com o campo *data*. O valor desse campo deve ser um *array* no formato JSON em que cada elemento do *array* é um objeto com os atributos da medição: *longitude* (tipo *float*), *latitude* (também do tipo *float*), *date* (tipo *string*, com formato dd/mm/yyyy), o campo opcional *time* (tipo *string* com formato hh:mm:ss), *variable* (tipo *string*), *value* (tipo *float*), *unit* (tipo *string*) e *category* (tipo *string*). Cada variável deve ser de uma das seguintes categorias predefinidas pelo sistema: solo, produção vegetal, meteorologia ou produção animal.

O *endpoint* measurements/ é usado para realizar buscas no banco de dados. Os resultados das buscas podem ser filtrados de acordo com o aspecto espacial, temporal e do valor do dado. A requisição para busca de medições pode ser uma requisição sem nenhum campo e, neste caso, todas as medições do usuário serão retornadas. Podem ser usados os campos *date-min*, *date-max*, *time-min*, *time-max* para filtrar as medições de acordo com o aspecto temporal. Para filtrar de acordo com o aspecto espacial pode ser usado o campo *polygon*, composto de uma lista com pelo menos três coordenadas (um objeto JSON com os atributos longitude e latitude). Para filtrar de acordo com o dado observado, podem ser usados os campos *name*, formado por uma *string* contendo o nome das variáveis desejadas (ex. 'potássio fósforo'), *value-min*, do tipo *float*, *value-max* também do tipo *float* ou *category*, campo do tipo *string* contendo uma das categorias

predefinidas e citadas anteriormente.

4. Resultados e Discussão

A AgroGraphAPI correntemente disponibiliza as funções para inserção e recuperação de dados agropecuários. Os *endpoints* da API possibilitam diversas formas de filtragem. Consultas direcionadas ao banco, que teriam alguma complexidade de elaboração, podem ser facilmente executadas ao usar o *endpoint* de busca e informar os limitadores da busca. O sistema de autenticação baseado em *tokens* permite que a API seja integrada com facilidade a outros sistemas. Como o modelo de dados é genérico o suficiente para receber qualquer dado georreferenciado, a API pode ser usada como base de dados por diversas aplicações diferentes.

Alguns pontos referentes à escalabilidade do projeto ainda devem ser analisados mais profundamente, especialmente quanto ao desempenho da aplicação quando há um volume de dados maior. Especificamente, o uso da *hash* para identificar medições comparado ao uso das próprias relações entre os nodos para identificar as medições; e também uso do pacote *shapely* para filtrar nodos de acordo com o aspecto espacial do dado na aplicação, comparado a usar o *plugin* de georreferenciamento do Neo4j realizando o filtro ainda no banco de dados.

Ainda existem algumas *features* que necessitam ser implementadas, como possibilitar o compartilhamento autorizado de dados entre diferentes usuários. A funcionalidade de Krigagem [Krige 1951] dos dados inseridos na base, implementada no sistema AgroGraph mas ainda não na API é um trabalho em andamento. A possibilidade de guiar a inserção de dados no sistema por meio de ontologias e tesauros é um trabalho futuro já planejado, visto que permitiria que o próprio sistema seja capaz de identificar se as variáveis que estão sendo inseridas já fazem parte do sistema, com um outro nome equivalente. Em sistemas em que os termos usados possuem forte influência regional, essa característica é importante.

A AgroGraphAPI está disponível no repositório do GitHub `https://github.com/Andersonsr/agrograph-api`, juntamente com o manual de uso.

5. Conclusão

A agricultura de precisão depende de dados provenientes dos sistemas produtivos agropecuários para a análise de diferentes ações de manejo que levem em conta a variabilidade do sistema. Diferentes técnicas podem ser usadas para análise dos dados, que dependem de um sistema de armazenamento confiável. Este trabalho apresentou a AgroGraphAPI, *application programming interface* que permite que operações de autenticação, armazenamento e recuperação de dados possam ser usados em quaisquer sistemas, favorecendo a construção modular de sistemas de software integrados, que atendam a necessidades específicas do produtores.

A evolução da API, com operações de geoestatística, definição pelo usuário de formatos de saídas específicos e outras operações, faz parte da continuação deste trabalho.

Referências

Arcuri, A. (2019). RESTful API automated test case generation with EvoMaster. *ACM Trans. Softw. Eng. Methodol.*, 28(1).

Bernardi, A. C. C., Naime, J. d. M., Resende, A. V., Inamasu, R. Y., and Bassoi, L. H., editors (2014). *Agricultura de precisão: resultados de um novo olhar*. Embrapa Instrumentação, São Carlos.

Bolfe, E. L., Junior, A. L., de Castro Victoria, D., and Grego, C. R. (2020). Agricultura digital no Brasil - tendências, desafios e oportunidades (resultado de pesquisa online). Technical report, Campinas.

Django Software Foundation (2023). Django documentation. [Online; accessed 25-Jul-2023].

Edwards, R. (2019). Neomodel documentation. [Online; accessed 25-Jul-2023].

Ehrig, H., Kreowski, H.-J., Montanari, U., and Rozemberg, G., editors (1996). *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, Singapore.

Fiss, R. E., Ferreira, A. P. L., and Perez, N. B. (2020). Análise de consultas SQL e Cypher em dados de produção agrícola. In *Anais da 7ª Conferência Ibero-americana de Computação Aplicaca (CIACA 2020)*, pages 212–216, Lisboa.

Gillies, S. (2023). The shapely user manual. [Online; accessed 26-Jul-2023].

Habel, A. (1992). *Hyperedge Replacement: Grammars and Languages*, volume 643 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin. 214p.

Ian Robinson, J. W. and Eifrem, E. (2015). *Graph Databases*, volume 224. O'Reilly Media, 1005 Gravenstein Highway North Sebastopol, CA 95472. 214p.

Krige, D. G. (1951). A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Southern African Institute of Mining and Metallurgy*, 52(6):119–139.

Massruhá, S. M. F. S. and Leite, M. A. A. (2017). *AGRO 4.0 – Rumo à Agricultura Digital*. Centro Paula Souza, São Paulo, 2 edition.

Neo4J (2020). The Neo4J Cypher manual v4.1. [Online; accessed 20-July-2020].

Neo4J, Inc. (2020). Neo4J documentation. [Online; accessed 25-Jul-2023].

Robinson, I., Webber, J., and Eifrem, E. (2015). *Graph Databases*. O'Reilly, 2 edition.

Rosa, A. S., Fiss, R. E., Ferreira, A. P. L., and Perez, N. B. (2022). AgroGraph: um sistema baseado em grafos para a agricultura de precisão. In *Congreso Argentino de AgroInformática*, volume 8, pages 67–80, Buenos Aires. Sociedad Argentina de Informática (SADIO).

Shepherd, M., Turner, J. A., Small, B., and Wheeler, D. (2018). Priorities for science to overcome hurdles thwarting the full promise of the 'digital agriculture' revolution. *Science of Food and Agriculture*, 100:5083–5092.

Silberschatz, A., Korth, H. F., and Sudarshan, S. (2006). *Sistema de Banco de Dados.* Elsevier, Rio de Janeiro.

Subramanian, H. and Raj, P., editors (2019). *Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*, volume 1. Packt Publishing Ltd, Birmingham.