ISSN 2764-7412

Ilustração: Magda Cruciol

COMUNICADO TÉCNICO

137

Campinas, SP
Dezembro, 2023

API RESTful

Alerta de doenças em Python com Django Framework

Daniel Rodrigo de Freitas Apolinário
Luiz Antonio Falaguasta Barbosa
Fabio Rossi Cavalcanti

API RESTful - Alerta de doenças em Python com Django Framework[1]

[1] Daniel Rodrigo de Freitas Apolinário, cientista da Computação, mestre em Ciência da Computação, analista da Embrapa Agricultura Digital, Campinas, SP. Luiz Antonio Falaguasta Barbosa, cientista da Computação, mestre em Ciência da Computação, analista da Embrapa Agricultura Digital, Campinas, SP. Fabio Rossi Cavalcanti, agrônomo, doutor em Fitopatologia, pesquisador da Embrapa Uva e Vinho, Bento Gonçalves, RS.

A avaliação de séries temporais climáticas para identificar a ocorrência de ambientes favoráveis à infecção por fitopatógenos é importante para garantir uma resposta rápida do produtor, a fim de evitar perdas de produtividade e de investimento em cultivos. Tal eficiência na resposta à ocorrência de uma determinada doença pode ser alcançada por meio de monitoramento em tempo real de microclima, mapeando pontos de favorabilidade ambiental à infecção (Cavalcanti, 2021). Essa favorabilidade a uma determinada doença está associada a fatores microclimáticos, como temperatura, umidade do ar, etc., ótimos para a interação biológica entre um patógeno e uma célula vegetal hospedeira (Campbell; Madden, 1990; Agrios, 2005). As associações entre ambiente, patógeno e hospedeiro estão resumidas, em Fitopatologia, no chamado “triângulo epidemiológico”. Elas podem definir se uma determinada doença vai ocorrer ou não em uma área de produção vegetal, e com qual severidade. Os três fatores (vértices) do triângulo são passíveis de modelagem Matemática/Estatística e aquisição de dados via sensoriamento por dispositivos eletromecânicos e/ou eletrônicos (Campbell; Madden, 1990; Agrios, 2005).

Um sistema para monitoramento de doenças de planta na agricultura é baseado, tradicionalmente, no monitoramento e coleta de dados climáticos por estações meteorológicas implantadas próximas a um talhão agrícola. Os dados dessas estações são fornecidos a um software capaz de estimar a necessidade do momento (timing) de aplicação de defensivos agrícolas com o objetivo de proteger a lavoura de um patógeno cujo controle esteja fundamentado em pulverização agrícola. Este software pode ser integrado a outros sistemas por meio de sua disponibilização na forma de uma interface de programação de aplicação (application programming interface−API). A Embrapa conta com uma plataforma que gerencia APIs, controlando o acesso às mesmas, onde um estudo de caso, o ”Módulo Computacional para Alerta de Doença por Mapa de Calor (heat map) em área de produção vegetal” (MAHM)(Registro no INPI BR512019002684-5, 2019), será disponibilizado. O MAHM é um algoritmo de alerta georreferenciado de doença baseado em sensoriamento

IoT (internet das coisas) de microclima por triangulação (Agrios, 2005), e pode retornar além do timing de aplicação, a coordenada de GPS onde deve haver um alerta e/ou aplicação de fungicida. O MAHM foi implementado em linguagem Python e, para disponibilizá-lo na forma de API RESTful[2], foi utilizado o Django, um framework web que utiliza o padrão model-template-view.

Esse trabalho apresenta detalhes de como foi projetada, construída e validada a transformação do software MAHM (script em Python) em uma aplicação API REST. Também é apresentada a configuração da API do MAHM na plataforma de gerenciamento de APIs (AgroAPI) para que ela possa ser integrada a outros sistemas.

Framework Django

O Django é um framework de desenvolvimento que dá suporte à transformação do MAHM em uma API REST. Assim, uma vez que o código-fonte do MAHM, desenvolvido em Python, é encapsulado como API usando as bibliotecas disponíveis no framework Django, pode-se disponibilizar o serviço para ser consumido por outras aplicações.

Transformação do software MAHM em uma API REST

Para transformar o MAHM em uma API REST, os seguintes passos devem ser seguidos:

1) Design da API de Favorabilidades: Definição de parâmetros de entrada e de saída e modelagem do formato dos dados.

2) Construção da API de Favorabilidades: A implementação da API de Favorabilidades envolve:

   - Construção das classes de modelo da aplicação, ou seja, modelagem dos objetos que representam as informações que serão enviadas como parâmetros de entrada e as que serão retornadas como parâmetros de saída da API;
   - Construção dos objetos serializadores responsáveis pelo mapeamento e transformação de dados no formato JSON para objetos da linguagem Python;
   - Construção da camada view da aplicação responsável por receber a requisição Web, validar, transformar e invocar o código Python do MAHM para o cálculo de favorabilidades;

3) Ajustes e otimização do núcleo do código MAHM: Explicação e des-

2 RESTful é a designação para uma API que atende as restrições definidas pelo estilo de arquitetura REST (Transferência de Estado Representacional, do inglês representational state transfer) definida por Roy T. Fielding (Fielding, 2000).

crição das principais alterações em relação ao código original do MAHM;

4) Configurações do Django: Principais arquivos e configurações do framework utilizados neste projeto;

5) Testes de funcionamento da API: uso da ferramenta Postman para validar o funcionamento da API;

6) Validação da aplicação: Comparativos de resultados retornados pela API com os resultados retornados diretamente pelo MAHM (script original em Python);

7) Implantação da API na infraestrutura computacional da Embrapa: Uso do *docker* para *deploy* da API em servidor;

8) Disponibilização da API para parceiros na plataforma AgroAPI: Configuração necessária para a AgroAPI devido às características das tecnologias utilizadas

Design da API de Favorabilidades

A API de Favorabilidades foi projetada para receber informações de umidade e temperatura de microrregiões de um vinhedo e calcular a favorabilidade da doença do míldio para cada uma destas microrregiões. Cada microrregião é representada por um quadrante, chamado de pixel. Ou seja, um vinhedo é representado por vários pixels de mesmo tamanho (área). No contexto desta aplicação, definimos o termo *situação ambiental* como sendo o conjunto de dados (umidade e temperatura) de um pixel (microrregião) obtidos em um determinado momento temporal. Por exemplo, se os dados dos sensores forem obtidos de hora em hora, cada hora terá uma situação ambiental para um pixel. Isso é importante, pois cada microrregião será tratada individualmente, podendo ter variações de umidade e temperatura em pixels adjacentes. Por definição, a API de Favorabilidades sempre receberá (para cada situação ambiental) uma matriz de pixels de dimensão 20 x 20, contendo um total de 400 pixels. Para cada um dos pixels, a API receberá as informações de umidade relativa do ar e temperatura correspondente. O retorno da API é a favorabilidade ambiental de cada situação ambiental e para cada pixel. A favorabilidade é representada por um número entre 0 e 1, onde 1 indica o máximo de favorabilidade ambiental para o míldio.

O design desta API, no formato OpenAPI 3.0, está definido em linguagem YAML (acrônimo do inglês YAML Ain't Markup Language) e disponível em https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Modelagem/swagger-API-favorabilidades-v3.yaml.

O parâmetro de entrada da API é um documento no formato JSON que pode conter uma lista de situações ambientais e para cada situação ambiental uma lista de 400 pixels, no qual cada pixel tem a informação de seu id, temperatura e umidade relativa, como no exemplo da Figura 1.

```
[
  {
    “situacaoAmbientalId”: 1,
    “pixels”: [
      {
        “id”: 1,
        “temperatura”: “14.92728659217722000000”,
        “umidadeRelativa”: “92.47594281958516000000”
      },
      {
        “id”: 2,
        “temperatura”: “14.86723579938652000000”,
        “umidadeRelativa”: “92.44566510893438000000”
      },
      {
        “id”: 3,
        “temperatura”: “14.80005140101787100000”,
        “umidadeRelativa”: “92.41179062236196000000”
      },
        … <outros pixels>
    ]
  },
  … <outras situações ambientais>
]
```


Figura 1. Documento no formato JSON com os parâmetros de entrada da API.

A resposta da API de favorabilidade retorna um JSON no formato que pode ser visualizado na Figura 2, no qual há uma lista de situações ambientais e para cada elemento desta lista há uma lista de classificações que contém o id do pixel e o valor correspondente à favorabilidade à doença do míldio.


```
[
  {
    “situacaoAmbientalId”: 1,
    “classificacoes”: [
      {
        “pixelId”: 1,
        “valor”: “0.72138336081446300000”
      },
      {
        “pixelId”: 2,
        “valor”: “0.71633335226026150000”
      },
      {
        “pixelId”: 3,
        “valor”: “0.71061350108468590000”
      },
    ... <outros pixels>
    ]
  },
  ... <outras situações ambientais>
]
```


Figura 2. Documento no formato JSON com a resposta da API de favorabilidades.

Construção da API de Favorabilidades

IDE para Desenvolvimento

Há várias IDEs disponíveis para desenvolvimento em Python. Este projeto pode ser aberto em qualquer uma delas. No projeto da API do MAHM optou-se por utilizar o PyCharm que possui versões disponíveis para Windows, macOS e Linux. Sua instalação é bem simples e é possível iniciar o desenvolvimento de forma rápida com um projeto em computador desktop. O PyCharm já possui integração com o Git de forma nativa. E também possui um bom mecanismo de debug de código. Estas características foram importantes para a equipe escolher esta ferramenta.

Models

A estrutura definida no documento JSON precisa ser mapeada para uma estrutura de objetos na aplicação. No Django, isso é feito declarando classes com seus atributos que sejam equivalentes à estrutura dos documentos JSON. Neste projeto, foi utilizado o mesmo arquivo ***models.py*** no qual são definidos os objetos, seus atributos e suas restrições (tipo, tamanho máximo, obrigatoriedade, etc).

O conteúdo deste arquivo na API de favorabilidades pode ser visualizado em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/mahmAPI/models.py.

Ao analisar o código, pode-se ver que a variável "*managed = False*" é usada, pois este modelo não é persistido em banco de dados. Também é possível notar que todas as classes possuem um método *create* para fazer a instanciação destes objetos no MAHM.

Serializers

Outro conceito importante no Django é o de *serializers*, responsáveis pela transformação dos dados em formato JSON para objetos e estruturas da linguagem Python. Um exemplo de serializer da API MAHM pode ser visualizado em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/mahmAPI/serializer.py.

Portanto, o arquivo *serializer.py* contém os serializadores usados no MAHM API. Estas classes são importantes para fazer o parser do JSON de entrada e depois para criar o JSON de saída que será a resposta da API.

Views

Com o design da API definido, foi utilizada a estrutura do framework Django para implementar o código responsável por receber o JSON de entrada (com dados de temperatura e umidade), tratar, invocar o código do MAHM que calcula as favorabilidades e retornar o resultado no formato JSON. No Django, o arquivo responsável por receber o *request* da API é o arquivo ***views.py***. Note que o encapsulamento do cálculo do MAHM é feito quando se invoca a

classe cuja única responsabilidade é fazer o cálculo das favorabilidades (a invocação à classe é feita por Mahm.calcular_favorabilidades(<parâmetro>)). Ou seja, o cálculo foi separado em uma classe e definido o parâmetro de entrada (objeto que representa os valores de temperatura e umidade para cada pixel em cada situação ambiental). O retorno do cálculo (resultado do cálculo das favorabilidades) é obtido como um novo objeto, agora com os valores das favorabilidades para cada pixel e em cada situação ambiental. O código do arquivo *views.py* está disponível em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/mahmAPI/views.py.

Pode-se notar que o arquivo *views.py* contém as classes que correspondem aos recursos disponibilizados pela API. Seu código também é responsável por executar validações e verificações relacionadas ao formato dos parâmetros de entrada e da integridade das informações.

Ajustes e otimização do núcleo do código MAHM

O código MAHM original recebe um arquivo CSV como parâmetro de entrada, faz os cálculos de favorabilidade e retorna um arquivo resposta no formato CSV, além de gerar alguns gráficos de mapa de calor de acordo com as favorabilidades calculadas.

Seguem abaixo as alterações feitas no código original do MAHM para o código disponível na API do MAHM:

- *Remoção da geração de gráficos:* a API somente iria trabalhar com os resultados numéricos do cálculo de favorabilidades;
- *Adaptação dos parâmetros de entrada*: os parâmetros assim como variáveis associadas aos dados recebidos de temperatura e umidade tiveram que ser tratados como objetos declarados na definição de modelos para ser possível trabalhar com JSON e não mais com CSV;
- *Adaptação dos parâmetros de saída*: os parâmetros assim como variáveis associadas aos resultados dos cálculos de favorabilidades tiveram que ser tratados como objetos declarados na definição de modelos para ser possível enviar uma resposta no formato JSON e não mais um arquivo CSV;
- *Otimização de código*: algumas estruturas de repetição com base em *arrays* foram substituídas por novas funções e os cálculos foram otimizados para usar as operações aritméticas em nível de *array* e não por acúmulo de valores em estruturas de *loop*.

O código do cálculo de favorabilidades (arquivo Mahm.py) não será publicado aqui, nem o original e nem o código otimizado, por questões de propriedade intelectual.

Configurações do Django

Quando um novo projeto Django é criado, automaticamente ele cria uma estrutura mínima de projeto. Este processo não é abordado aqui neste trabalho, pois há vários tutoriais disponíveis para criar um projeto Django. Neste trabalho partiu-se de um projeto criado, implementado e explicado como foi feito. Dentro desta estrutura existe uma pasta *setup* que contém os principais arquivos de configurações globais da aplicação, conforme se vê na Figura 3.

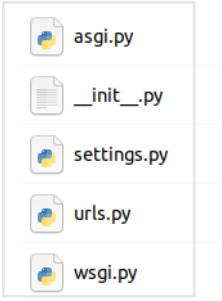

Figura 3. Principais arquivos de configurações globais da aplicação.

O arquivo settings.py é o local no qual pode-se configurar as bibliotecas que a aplicação utilizará, como a rest_framework que consta na lista de bibliotecas que pode ser visualizada na Figura 4.

Neste arquivo também são configurados acessos a bancos de dados (no caso do MAHM não há nenhum acesso a banco de dados). Outro tipo de configuração comum é relacionado ao *timezone* e linguagem da aplicação mostrado na Figura 5.

```
INSTALLED_APPS = [
        'django.contrib.admin',
        'django.contrib.auth',
        'django.contrib.contenttypes',
        'django.contrib.sessions',
        'django.contrib.messages',
        'django.contrib.staticfiles',
        'rest_framework',
        'mahmAPI',
]
```

Figura 4. Trecho de configuração de bibliotecas no arquivo settings.py

```
# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/
LANGUAGE_CODE = 'pt-br'

TIME_ZONE = 'America/Sao_Paulo'
```

Figura 5. Trecho de configuração de timezone e linguagem no arquivo settings.py

Outro arquivo importante é o *urls.py,* no qual são configuradas todas as rotas da aplicação, ou seja, quais ***endpoints*** a aplicação terá e quais classes irão atender às requisições. Isso pode ser visto, como exemplo, no arquivo disponível em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/setup/urls.py

Configurar o ambiente de desenvolvimento da API do MAHM

Os passos para que um desenvolvedor consiga montar o ambiente para desenvolvimento da API do MAHM estão descritos no documento *README.md* disponível em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/README.md

Para invocar a aplicação configurada localmente no computador de um desenvolvedor, pode-se usar o comando curl (Linux) contido no arquivo disponível em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/samples/request-test-curl-v3.txt

Testes de funcionamento da API

Após a execução do comando *python manage.py* runserver, é possível acessar remotamente o Django para a realização de testes dos recursos disponibilizados pela API. Isso pode ser feito tanto por meio do comando *curl*, apresentado na seção anterior, quanto pelo software Postman.

No Postman, basta inserir a URL de acesso ao serviço do Django, conforme Figura 6.

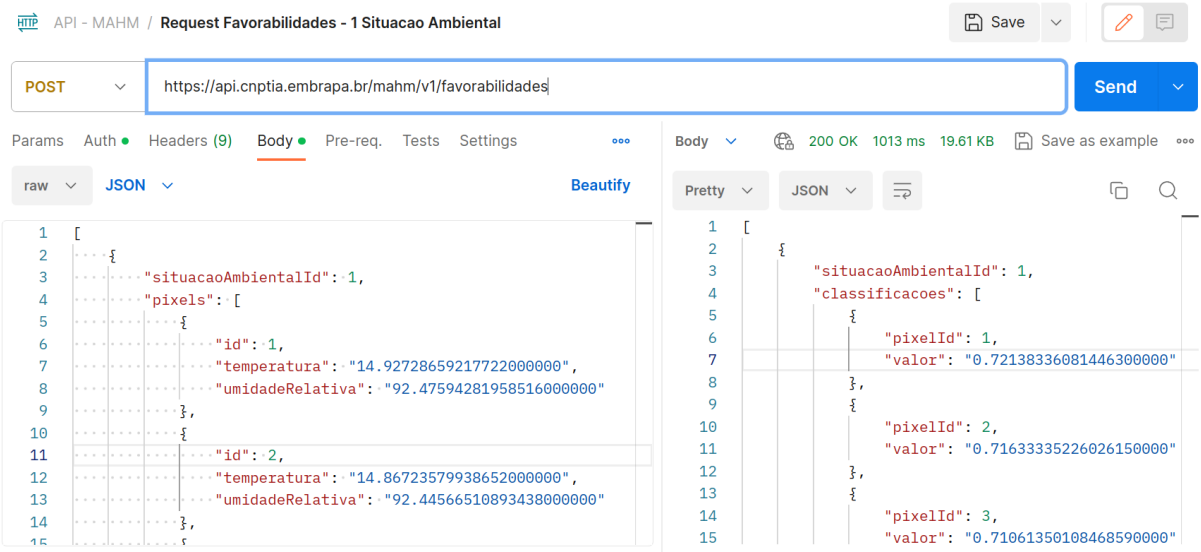

Figura 6. Ferramenta Postman de testes de acesso remoto à API RESTful.

Nela, é possível ver que a requisição foi realizada por método POST à URL http://localhost:8000/mahm/v1/favorabilidades para obter as favorabilidades existentes nos 400 pixels da área analisada.

Validação da aplicação

Para a validação da API, foram executados testes no código originalmente criado para funcionamento manual (fora da API) e através da API na plataforma AgroAPI. Para isso, os mesmos dados de entrada foram utilizados, bem como a comparação dos resultados em cada um dos cenários.

Dessa forma, foi utilizado um script que recebe um arquivo CSV e gera outro em formato JSON para ser usado como entrada na AgroAPI. Assim, o arquivo CSV de entrada pode ser usado para execução manual em Python no script original do MAHM e o arquivo JSON dentro da plataforma AgroAPI, e seus resultados comparados.

Como exemplos de entrada, segue o início do arquivo CSV (Figura 7) e do arquivo (Figura 8) JSON de entrada.

Os arquivos de saída no formato CSV e JSON podem ser vistos respectivamente nas Figuras 9 e 10.

A imagem *docker* é gerada a cada *commit* no Gitlab e então o mecanismo de integração contínua do GitLab cria uma instância de *container* para a imagem dentro de máquina virtual disponível na infraestrutura da Embrapa Agricultura Digital.

```
TEMP1,UR1,TEMP2,UR2,TEMP3,UR3,TEMP4,UR4,TEMP5,UR5,TEMP6,UR6,TEMP7,UR7,TEMP8,UR8,TEMP9,UR9,-
TEMP10,UR10,TEMP11,UR11,TEMP12,UR12,TEMP13,UR13,TEMP14,UR14,TEMP15,UR15,TEMP16,UR16,TEM-
P17,UR17,TEMP18,UR18,TEMP19,UR19,TEMP20,UR20,TEMP21,UR21,TEMP22,UR22,TEMP23,UR23,TEMP24,UR24,-
TEMP25,UR25,TEMP26,UR26,TEMP27,UR27,TEMP28,UR28,TEMP29,UR29,TEMP30,UR30,...
4.190000000000001,89.8,4.1899999999999995,89.79999999999998,4.190000000000001,89.8,4.1899999999999995,89.8,4.19,89.
80000000000001,4.19,89.8,4.19,89.8,4.19,89.8,4.19,89.79999999999998,4.19,89.8,4.19,89.8,4.190000000000001,89.8,4.19,89.800
00000000001,4.19,89.8,4.19,...
```

Figura 7. Trecho de arquivo CSV de testes com parâmetros de entrada.

```
[
    {
        “situacaoAmbientalId”: 1,
        “pixels”: [
            {
                “id”: 1,
                “temperatura”: “14.92728659217722000000”,
                “umidadeRelativa”: “92.47594281958516000000”
            },
            {
                “id”: 2,
                “temperatura”: “14.867235799386520000000”,
                “umidadeRelativa”: “92.44566510893438000000”
            },
            {
                “id”: 3,
                “temperatura”: “14.800051401017871000000”,
                “umidadeRelativa”: “92.41179062236196000000”
            },
            {
                “id”: 4,
                “temperatura”: “14.731126382921056000000”,
                “umidadeRelativa”: “92.37703851239718000000”
            },
            {
                “id”: 5,
                “temperatura”: “14.661514661638346000000”,
                “umidadeRelativa”: “92.34194016553194000000”
            },
…
            }
        ]
    }
]
```


Figura 8. Exemplo de JSON de entrada equivalente ao arquivo CSV da Figura 7.


```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30, ...
0,0.07628267188771747,0.07628267188771737,0.07628267188771747,0.07628267188
8771747,0.07628267188771759,0.07628267188771747,0.07628267188771747,0.0762
8267188771747,0.07628267188771737,0.07628267188771747,0.07628267188771747,
0.07628267188771747,0.07628267188771759,0.07628267188771747,0.076282671887
71737,0.07628267188771759,0.07628267188771747,0.07628267188771747,0.076282
67188771759,0.07628267188771747,0.07628267188771747,0.07628267188771747,0.
07628267188771747,0.07628267188771747,0.07628267188771747,0.0762826718877
1747,0.07628267188771759,0.07628267188771747,0.07628267188771737, ...
```


Figura 9. Exemplo de CSV com o resultado do cálculo das favorabilidades.

```
[
  {
    "situacaoAmbientalId": 1,
    "classificacoes": [
      {
        "pixelId": 1,
        "valor": "0.72138336081446300000"
      },
      {
        "pixelId": 2,
        "valor": "0.71633335226026150000"
      },
      {
        "pixelId": 3,
        "valor": "0.71061350108468590000"
      },
      {
        "pixelId": 4,
        "valor": "0.70467012462265500000"
      },
      {
        "pixelId": 5,
        "valor": "0.69859173517442170000"
      },
…
      ]
    }
]
```


Figura 10. Exemplo do JSON com o resultado do cálculo das favorabilidades.

Implantação da API na infraestrutura computacional da Embrapa

Para gerar uma imagem docker com aplicação e depois executar em um container, basta executar os comandos apresentados na Figura 11 (a instalação do Docker é pré-requisito para a execução destes comandos).

A imagem docker é gerada a cada commit no Gitlab e então o mecanismo de integração contínua do GitLab cria uma instância de container para a imagem dentro de máquina virtual disponível na infraestrutura da Embrapa Agricultura Digital.

```
docker build -t favorabilidade -f ./Dockerfile .
docker run -d --name favorabilidade-container
-p 8000:8000 favorabilidade
```

Figura 11. Comandos docker para geração de imagem e criação de container para execução da API de favorabilidades.

Criação e Publicação da API na AgroAPI

O processo de criação e publicação da API de Favorabilidades segue o processo normal, com exceção de um passo específico no qual precisa-se incluir uma mediação para que a API funcione corretamente. Isto é necessário pois o WSO2 trabalha com *chunked requests* e o Django não pode trabalhar com este tipo de requisição. Por isso, é necessário realizar esta configuração adicional no cadastro da API de Favorabilidades.

A configuração está no arquivo disableChunkingSeq.xml disponível em: https://www.gitlab.cnptia.embrapa.br/publico/mahm/mahm-code/-/blob/main/Sources/agroapi/disableChunkingSeq.xml

Referências

APOLINÁRIO, D. R. de F.; CORREA, J. L.; VACARI, I.; GONZALES, L. E.; LIMA, H. P. de; EVANGELISTA, S. R. M. **Gerenciamento de APIs com o WSO2 API Manager**. Campinas: Embrapa Informática Agropecuária, 2017. 11 p. (Embrapa informática Agropecuária. Comunicado técnico, 128).

AGRIOS, G. N. **Plant Pathology.** 5. ed. [S.l.]: Elsevier, 2005. 948 p.

CAMPBELL, C. L.; MADDEN, L. V. **Introduction to plant disease epidemiology**. New York: Wiley & Sons, 1990. 532 p.

CAVALCANTI, F. R. **Algoritmo (MAHM) para alerta georreferenciado de doença em redes de sensoriamento IoT de microclima: calibração e teste de um método para míldio, em dois vinhedos.** Bento Gonçalves, RS: Embrapa Uva e Vinho, agosto 2021. 25 p. (Embrapa Uva e Vinho. Circular Técnica, 124). Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/handle/doc/1134000. Acesso em: 23 out. 2023

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Dissertation (Doctor of Philosophy) - University of California, Irvine. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 26 jan. 2024.

Embrapa Agricultura Digital

Av. André Tosello, nº 209 Campus da Unicamp, Barão Geraldo
CEP 13083-886, Campinas, SP
Fone: (19) 3211-5700
www.embrapa.br/agricultura-digital
www.embrapa.br/fale-conosco/sac

Publicação digital (2023): PDF

MINISTÉRIO DA AGRICULTURA E PECUÁRIA

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

CGPE 18424