UNIVERSIDADE FEDERAL DE ITAJUBÁ

SISTEMAS DE INFORMAÇÃO

INTEGRAÇÃO DE APLICAÇÕES COM A API PYTHON DO GOOGLE EARTH ENGINE

Tassio Silva

Trabalho de Conclusão de Curso
Bacharelado em Sistemas de Informação

INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
UNIFEI - Itajubá

2022

Silva, Tassio
Integração de aplicações com a API Python do Google Earth Engine / Tassio Silva. - Itajubá: UNIFEI, 2022.
10 p.
Trabalho de Conclusão de Curso (Sistemas de Informação) - Universidade Federal de Itajubá, Itajubá, 2022
Orientação: Vanessa Cristina Oliveira de Souza
1. Geotecnologias. 2. Agricultura digital. 3. *Google Earth Engine*. I. Souza, Vanessa Cristina Oliveira de , orient. II. Título.

UNIVERSIDADE FEDERAL DE ITAJUBÁ
INSTITUTO DE MATEMÁTICA E COMPUTAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Integração de aplicações com a API Python do Google Earth Engine

Tassio Silva

tassio.s13@gmail.com

Orientador: Vanessa Cristina Oliveira de Souza

RESUMO
Este trabalho descreve a evolução de um aplicativo desenvolvido para auxiliar cafeicultores do Sul de Minas Gerais em decisões de manejo hídrico. O App *Regador* está incluído no contexto da agricultura digital e sua arquitetura original não contempla a carga automática dos mapas de potencial hídrico para o servidor de arquivos do APP, compromentendo a temporalidade do serviço. Nesse trabalho a carga dos dados para o aplicativo foi automatizada utilizando a API Python do *Google Earth Engine* e soluções de agendamento de tarefas. Para o desenvolvimento do trabalho, foi utilizada a metodologia de pesquisa *Design Science Research methodology for Information Systems* (DSRM), consistindo no segundo ciclo de iteração, que visa melhorias no artefato já existente. Para avaliar a nova arquitetura foram feitos testes de *stress* para garantir que o desempenho do aplicativo não foi alterado. Os resultados apontam que a integração foi realizada com sucesso e essa melhoria traz uma grande vantagem para a manutenção da carga de dados do aplicativo.

Palavras-chave: Geotecnologias; Agricultura digital; *Google Earth Engine*

1 Introdução

A água é um recurso essencial para qualquer tipo de produção agrícola. Estima-se que até 2050, o mundo tenha aproximadamente 318 milhões de hectares irrigados (Dubois et al., 2011). É essencial que o manejo da água em uma lavoura seja o mais correto possível para não prejudicar o meio ambiente e também não prejudicar o desenvolvimento da planta (Rodrigues, 2017).

Sendo assim, é de suma importância o desenvolvimento de novas tecnologias para auxiliar as tomadas de decisões em relação ao manejo hídrico no campo, de maneira que garanta o desenvolvimento sustentável da agricultura sem afetar a produtividade das lavouras. As plantas podem sofrer vários tipos de *stress* ambientais. Um deles é o *stress* hídrico, que é uma pressão excessiva relacionada a hidratação da planta (Araújo Júnior et al., 2019).

O potencial hídrico foliar (representado pela letra grega Ψ) representa a energia livre associada às moléculas de água em uma planta e está relacionado com fluxos da água no sistema solo-planta-atmosfera (Correia, 2014). Desse modo, variações no potencial hídrico da folha comprometem a produtividade da maioria das plantas cultivadas. Em especial no café, onde pequenas modificações nas condições hídricas podem reduzir intensamente o crescimento, mesmo não ocorrendo as respostas típicas das plantas nessas condições, como a murcha das folhas (Batista et al., 2010).

Considerando a importância de se conhecer o potencial hídrico foliar nas plantas do cafeeiro e a dificuldade de obter seu valor em campo Silveira et al. (2015), (Maciel et al., 2020) desenvolveram um modelo matemático para estimar o Ψ a partir de imagens de sensoriamento remoto provenientes do satélite orbital Landsat 8. O modelo é válido para a região Sul de Minas Gerais.

Com o intuito de democratizar e popularizar o modelo desenvolvido por (Maciel et al., 2020), (Silva et al., 2022) desenvolveram um aplicativo mobile nomeado App *Regador*, que disponibiliza o potencial hídrico estimado por meio de mapas. Para tanto, o modelo matemático foi implementado no *Google Earth Engine* (GEE), uma plataforma em nuvem para processamento de dados geoespaciais (Gorelick et al., 2017). No GEE, operações matemáticas são realizadas sobre imagens do LandSat 8 e resulta em uma imagem de saída no qual cada *pixel* armazena um número real referente à estimativa do potencial hídrico. Portanto, nesse trabalho, o termo "mapa"refere-se à essa imagem resultante.

Durante o desenvolvimento de um *software*, as necessidades dos usuários devem ser traduzidas em requisitos e, posteriormente, em artefatos de *software* (Turner et al., 1998). O App *Regador* foi baseado em nove requisito funcionais descritos em Silva et al. (2022), que cobriram todas as necessidades dos usuários. Porém o requisito não-funcional referente à atualização automática dos mapas no aplicativo não foi implementado. Atualmente, a carga de novos dados no APP é um processo manual. Através da API Javascript do GEE, o mapa é gerado e armazenado no Google Drive para depois ser carregado manualmente no servidor do *Regador*.

No entanto, a temporalidade dos dados no aplicativo é uma questão de suma importância para que os agricultores sempre recebam informações atualizadas sobre suas lavouras e possam tomar decisões de manejo mais assertivas. Um novo mapa deve ser gerado a cada nova passagem do satélite e por isso a automatização do processo de atualização dos mapas do *Regador* é importante para que o aplicativo não fique com

dados desatualizados.

A API Javascript quando utilizada através da interface web do GEE (*Code Editor*) dificulta a automação do processo de atualização dos mapas no servidor do APP. Como alternativa, o GEE fornece uma API em Python, que é mais flexível pois permite agregar qualquer função ou *lib* da linguagem durante o desenvolvimento (Bade, 2020).

Nesse contexto, o presente trabalho parte da hipótese de que com o uso da API Python do GEE será possível integrar o APP *Regador* com o GEE, automatizando a carga dos mapas de estimativa de potencial hídrico e garantindo a temporalidade dos dados no aplicativo. Para tanto, o trabalho teve como principais objetivos:

- Analisar a viabilidade de uso da API Python do GEE no processo de atualização automática dos mapas do APP *Regador*;
- Desenvolver em Python o algoritmo que implementa o modelo matemático de estimativa de potencial hídrico de Maciel et al. (2020), tendo o como referência a implementação em Javascript.
- Propor uma estratégia de automação periódica da geração do mapa de potencial hídrico via GEE e sua carga no servidor de arquivos do App *Regador*;

Este trabalho é um estudo de caso, no qual será proposto estratégias de automação para o problema descrito anteriormente. O trabalho pretende contribuir com a evolução do APP *Regador* e, consequentemente, em subsidiar a tomada de decisão no manejo hídrico de culturas cafeeiras, atendendo principalmente os pequenos cafeicultores de Minas Gerais.

O restante do trabalho está estruturado da seguinte forma: A Seção 2 aborda o referencial teórico do tema; a Seção 3 apresenta as etapas de desenvolvimento do trabalho, juntamente com a solução desenvolvida. A Seção 4 apresenta a avaliação da solução desenvolvida e a ultima Seção apresenta as considerações finais.

2 Referencial Teórico

Nos últimos anos as novas tecnologias proporcionaram um aumento do poder computacional, o armazenamento e disponibilidade de dados. Essas tecnologias impulsionaram a industria tradicional a se elevar para um novo patamar conhecido como indústria 4.0 (Santos et al., 2018). Com essa evolução, novas soluções surgiram para aumentar a produtividade e auxiliar a tomada de decisão em indústrias, com o objetivo de criar produtos mais baratos, em menor tempo e com maior qualidade (Coito et al., 2020).

Para o agronegócio esse panorama não foi diferente. O uso de tecnologias para auxiliar a tomada de decisões de manejo, reduzir custos, otimizar processos de produção, melhorar eficiência do uso de insumos, entre outros, são os desafios que levaram a agricultura tradicional para a agro 4.0. O Agro 4.0, também conhecida como agricultura digital, é uma referência a Indústria 4.0, que emprega métodos computacionais, redes de sensores, comunicação entre máquinas, conectividade entre dispositivos móveis, computação em nuvem e soluções analíticas para processar grandes volumes de dados e construir sistemas de suporte a tomada de decisões de manejo em campo (Massruhá & Leite, 2017).

Neste contexto, aplicações móveis para auxiliar o manejo de lavouras são exemplos de aplicações que vão de encontro com a premissa do Agro 4.0. O aplicativo *Regador* é um exemplo deste tipo de aplicação.

2.1 APP *Regador*

O App *Regador* (Silva et al., 2022) surgiu da necessidade de democratizar o acesso dos cafeicultores mineiros aos resultados de pesquisas científicas. Para tanto, o aplicativo teve como principal objetivo traduzir o que estava publicado em periódico especializado, em um produto de *software* que propicia o acesso dos cafeicultores a informações práticas e descomplicadas, subsidiando o manejo em campo.

O modelo matemático desenvolvido por Maciel et al. (2020) é apresentado nas Equações 1 e 2. Onde a sigla NDVI representa o índice de vegetação da diferença normalizada, que é calculado através das bandas espectrais captadas pelos satélites, a sigla NIR representa o espectro infravermelho próximo e RED representa o espectro vermelho (Grohs et al., 2009). Na Figura 1, o modelo traduzido em mapa e disponibilizado no App *Regador*.

$$\Psi = [-8,712 + (17,325 * NDVI) - (8,739 * NDVI^2)] \quad (1)$$

Sendo:

$$NDVI = \frac{NIR - RED}{NIR + RED} \quad (2)$$

Pela interface do App *Regador*, o usuário pode escolher uma região que deseja ver o potencial hídrico como demonstrado na Figura 1A, ou criar uma nova região com os limites desejados através do botão sinalizado na Figura 1B. Os potenciais hídricos são retornados para o usuário conforme a data mais recente em que o satélite obteve imagens da região.

O aplicativo foi desenvolvido utilizando um conjunto de tecnologias demonstradas na Figura 2. A arquitetura é dividida em 3 partes.

Parte 1 representa a geração do mapa de estimativa do potencial hídrico, que é o produto disponibilizado pelo App. Para tanto, a Equação 1 foi implementada utilizando a API Javascript da plataforma em nuvem *Google Earth Engine*, detalhada na Seção 2.2. Nessa API, o mapa é gerado e enviado para o *Google Drive*. Uma vez neste repositório, o mapa é carregado manualmente para o servidor de arquivos do *Regador*, representado pelo número 2 na Figura 2.

A parte 2 representa o servidor *back-end* do aplicativo. O servidor é baseado em uma *API rest* desenvolvida na linguagem Python. O *back-end* está hospedado na plataforma em nuvem chamada *Heroku*[1], que é uma plataforma para hospedagem de sistemas e sites. Os dados de potencial hídrico que alimentam o aplicativo são gerados na etapa 1 e carregados manualmente no código fonte do *back-end* do aplicativo. E devido a isso, sempre que uma nova imagem é gerada, é necessário que essa imagem seja versionada no código, e que o *back-end* seja recompilado e enviado ao servidor como se uma nova versão tivesse sido disponibilizada (Silva et al., 2022).

A parte 3 demonstra o aplicativo que é utilizado pelos usuários. O qual foi desenvolvido utilizando o *framework Ionic* (Chaudhary, 2018). O aplicativo consome os *endpoints* da

[1] https://www.heroku.com/

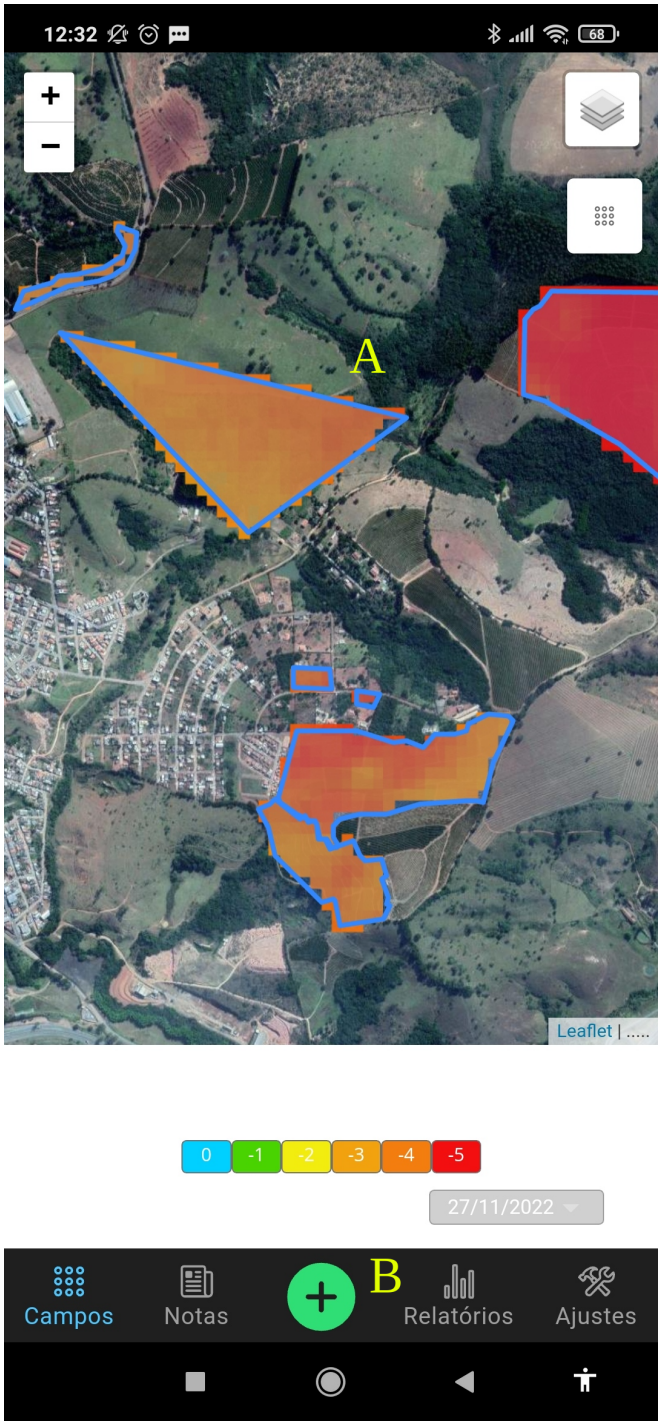

Figura 1: Tela principal do App *Regador*

API REST, fazendo todas as requisições necessárias para o funcionamento da solução.

Em relação ao fluxo de dados, o usuário deve primeiro escolher na interface do aplicativo uma área (denominada campo) e uma data para visualizar o potencial hídrico estimado. Nesse momento, o aplicativo fará executará as seguintes ações: (I) fará uma requisição, ao *back-end* ; (II) buscará a imagem correspondente à data selecionada; (III) fará as computações necessárias e retornará para o *front-end* os dados de potencial hídrico correspondente àquela região naquela data específica. No aplicativo o mapa é apresentado em escala de cor e também permite que o usuário clique sobre a área e obtenha os valores numéricos referentes ao potencial hídrico.

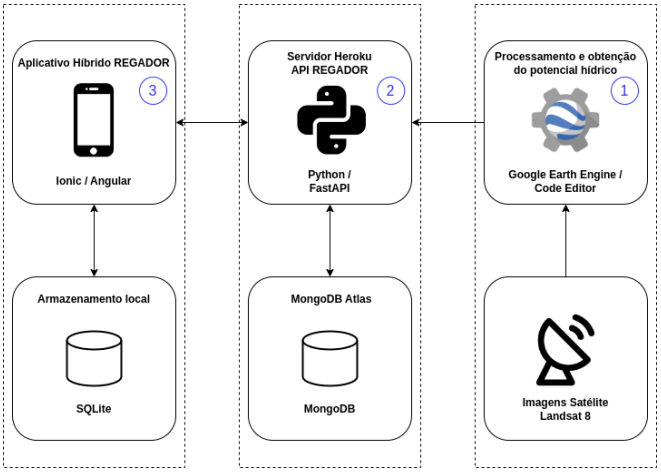

Figura 2: Arquitetura original do aplicativo - Adaptado de: Silva et al. (2022)

2.2 Plataformas em nuvem de processamento de dados geoespaciais

Computação de alto desempenho tem se tornando cada vez mais acessível com os avanços das tecnologias em nuvem. Paralelamente, dados geoespaciais de diversas agências do mundo se tornaram gratuitos. Unindo esses fatos à grande variedade de ferramentas que facilitam o processamento de dados geoespaciais como o Hadoop (Whitman et al., 2014), GeoSpark (Yu et al., 2015), e GeoMesa (Hughes et al., 2015), foi possível criar plataformas de geoprocessamento em nuvem para geoprocessamento que não exigem alto conhecimento técnico nem um grande esforço para serem utilizadas, deste modo, ocorre uma popularização da área, levando informação a vários pesquisadores e usuários comuns (Gorelick et al., 2017).

O *Google Earth Engine* é uma plataforma de computação em nuvem para processamento de imagens de satélite e outros dados geoespaciais e de observação. Ela fornece acesso a um grande banco de dados geoespacial e o poder computacional necessário para analisar esses dados. A popularização do GEE diminuiu a complexidade da observação de mudanças dinâmicas na agricultura, recursos naturais e clima por meio do geoprocessamento. A plataforma fornece interfaces de programação de aplicativos nas linguagens Python e JavaScript para fazer solicitações aos servidores (Gorelick et al., 2017).

O GEE é mais utilizado através de uma plataforma conhecida como *Code Editor* (Gorelick et al., 2017), por meio da qual é possível escrever, executar e compartilhar *scripts* com o objetivo de analisar e processar dados geoespaciais (Bade, 2020). Devido ao fato dessa plataforma ser utilizada dentro do ambiente do GEE, não é necessário nenhum tipo de configuração de ambiente, só é necessário uma conta de desenvolvedor Google para acessar. A outra API é utilizada através da linguagem Python[2] e diferente da API javascript que é majoritariamente utilizada dentro do ambiente WEB do GEE, ela pode ser executada localmente, ou em qualquer máquina que possua os pré-requisitos instalados e ambientes configurados. A principal configuração necessária é a geração de credenciais atreladas a conta de desenvolvedor Google para que as requisições ao GEE sejam autorizadas.

[2] https://developers.google.com/earth-engine/guides/Python_install

Existe também uma terceira distribuição em Javascript que pode ser executada com o motor de execução Node.js[3]. Esta opção carrega a mesma flexibilidade da API Python, e também as mesmas necessidades de configuração de ambiente e instalação de pré requisitos.

Ambas APIs funcionam no nível *Client Libraries*, como demonstrado na Figura 3. Qualquer aplicação pode ser integrada com o GEE através de suas APIs, como o próprio *Code Editor*. O GEE possui *APIs REST* privadas que são acessadas através das APIs públicas (Python ou Javascript) disponibilizadas para quem tenha interesse em utilizar o GEE. Apesar de escritas em linguagens diferentes, ambas APIs consomem os mesmos métodos encapsulados do GEE. Portanto, independente de qual API for utilizada, o resultado será o mesmo.

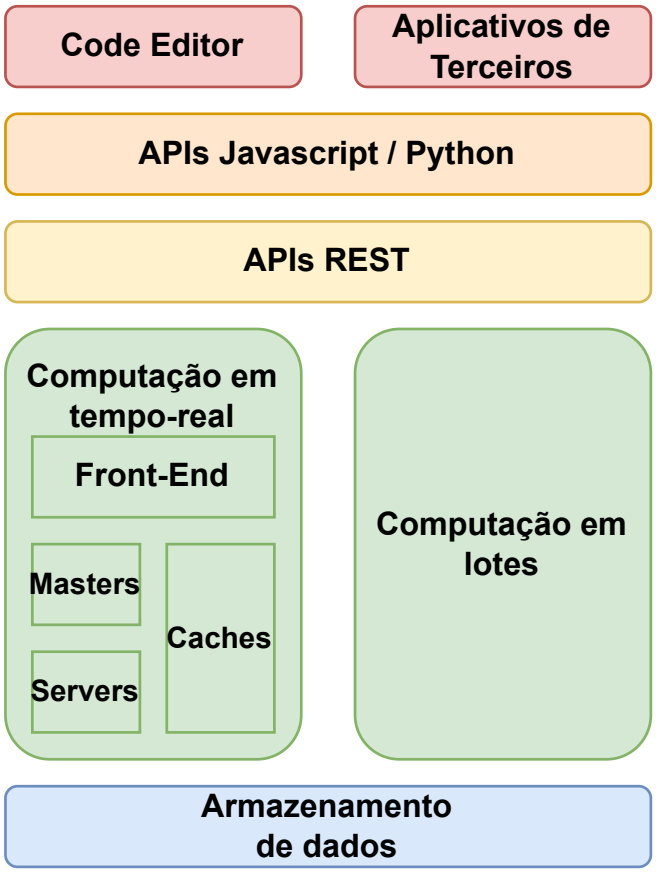

Figura 3: Arquitetura GEE - Adaptado de: (Gorelick et al., 2017)

As principais diferenças entre as APIS são citadas na Tabela 1. É importante ressaltar que com as diferenças citadas é possível concluir que a principal vantagem da API Python é a flexibilidade. Uma vez que o ambiente do *Code Editor* é limitado ao que foi implementado e está disponível em sua plataforma web. Ao utilizar a API Python ou a API Javascript em conjunto com o motor de execução Node.js, é possível expandir a quantidade de ferramentas que podem ser utilizadas em uma solução que usa o GEE. Como exemplo, pode-se citar a solução criada neste trabalho que utiliza o mesmo ambiente *back-end* para execução do *script* gerador do modelo de potencial hídrico.

Um bom exemplo para a flexibilidade da API Python é o pyveg (Barlow et al., 2020) que é uma biblioteca que utiliza o GEE através da API Python, para obter e processar imagens de satélite e avaliar a evolução da vegetação. O resultado do processamento pode ser usado para avaliar indícios de colapsos ambientais. Outra biblioteca Python interessante é a *eemont*(Montero, 2021) que é um conjunto de métodos implementados a partir da extensão de métodos nativos do GEE para auxiliar no pré e pós processamento de imagens. Essas ferramentas são exemplo de como a API Python expande o uso do GEE através de ferramentas criadas pelos próprios usuários.

2.3 Integração entre aplicações e o GEE

De acordo com Barkmeyer et al. (2008) uma integração é um estado de um conjunto de agentes que os habilita a atuar conjuntamente em uma ou mais subfunções de uma função de um sistema maior. Sendo assim, o aplicativo *Regador* e o GEE são agentes de uma integração que formam um sistema de monitoramento de potencial hídrico de lavouras de café. Como descrito anteriormente, o App *Regador* utiliza dados gerados na API Javascript do GEE, o que torna essa integração manual. Ainda segundo Barkmeyer et al. (2008), a automação de uma integração acontece quando todo o processo da integração ou algumas etapas da mesma são automatizadas. Ou seja, automação completa de alguns processos no projeto e desenvolvimento de um sistema maior.

A maior parte da comunidade GEE utiliza o *Code Editor* para desenvolver os algoritmos. Mas essa plataforma web de acesso ao GEE tem algumas limitações, como por exemplo, a dificuldade ao debuggar, a dificuldade de integrar o GEE a outras aplicações e também a dificuldade de usar outras ferramentas que não estão disponíveis no ambiente do GEE. Outra dificuldade é a comunidade de dados ser em sua maioria composta por desenvolvedores Python e R, o que torna difícil a utilização da plataforma *WEB* baseada em javascript (Krishnan, 2022).

Devido a essas dificuldades, e outras diferenças como as citadas na Tabela 1, integrações de aplicações com o GEE são melhores desenvolvidas utilizando as APIs fora do ambiente WEB do GEE com o *Code Editor*. Isso porque não é possível fazer uma comunicação entre a aplicação e o *Code Editor*, tornando necessário o uso de etapas manuais como a descrita na Seção 2.1.

2.4 Trabalhos Correlatos

Essa seção apresenta alguns trabalhos que realizaram a integração de aplicações com o GEE. O trabalho desenvolvido por Panidi et al. (2020) mostra o desenvolvimento de uma aplicação que integra o QGIS à API Python para o gerenciamneto de dados de sensoriamento remoto em nuvem através de uma plataforma desktop. Outra utilização interessante da API Python do GEE é a desenvolvida por (Krishnan, 2022), onde foi desenvolvido uma pequena aplicação utilizando o GEE e o greppo[4] para exemplificar como a integração de uma aplicação com API Python é simples.

Em Schultz et al. (2019) houve a integração entre o GEE e o sistema DataSafra para monitoramento da área plantada de talhões do chamado milho safrinha. Para realizar o mapeamento dos talhões foram utilizados mosaicos/composições

[3] https://developers.google.com/earth-engine/guides/npm_install

[4] https://greppo.io/

Tabela 1: Principais Diferenças entre as APIs JS e Python Adaptado de: (Bade, 2020)

Categoria	API JS	API Python
IDE	*Code Editor* IDE integrada que fornece acesso a todo o GEE de rapidamente	É possível usar qualquer IDE que desejar como Pycharm, jupyter notebook ou VScode
Bibliotecas	Possui várias bibliotecas nativas para criar tabelas, mapas e layouts	Possibilidade de utilizar qualquer biblioteca disponível para Python como TensorFlow, Pandas e MatPlotLib
Compartilhamento	Compartilhamento simples através de links do google drive.	Possibilidade de armazenar e versionar o código através de qualquer serviço como GitHub ou GitLab
Curva de aprendizado	Curva de aprendizado rápida devido a ambiente pré-configurado e boa documentação	Curva de aprendizado lenta devido a falta de boa documentação e bibliotecas nativas. Também existe a necessidade de configuração de ambiente.
Gerenciamento de arquivos	Todos os arquivos utilizados nos scripts são armazenados na infraestrutura do Google Drive	Flexibilidade para decidir se os arquivos serão armazenados localmente ou no Google Drive ou em qualquer outro serviço de armazenamento
Comunidade	Por ser mais utilizado tem comunidade maior, mais colaborativa e engajada	Comunidade menor e pode ser um pouco mais difícil de conseguir colaboração em problemas no código
Credenciais	A IDE *Code Editor* roda dentro do GEE e sendo assim não é necessário nenhuma autorização ou preparação do ambiente para utilizar o GEE.	Como esta API pode ser utilizada em qualquer ambiente é necessário configurar o ambiente e preparar as credenciais para fazer requisições ao GEE.

multitemporais de imagens de satélites Landsat durante a temporada de inverno. Para a extração da data de plantio foi realizada a análise de séries temporais e o calculo do NDVI de cada talhão através do GEE. O sistema fornece mensalmente para os agricultores as melhores datas de plantio, colheita, tamanhos de ciclo, entre outras informações. Para essa cultura, essa temporalidade significa que o monitoramento é praticamente em tempo real.

Existe na literatura exemplos de sistemas que utilizaram a API Python do GEE em seu desenvolvimento. O *CoastSat*, apresentado por Vos et al. (2019), utiliza a API para obter imagens, processar e obter a posição de linhas costeiras de qualquer região do litoral do mundo dos últimos 30 anos. Já a *AgriSuit* (Yalew et al., 2016) utiliza todo o poder computacional do *back-end* do GEE para preparar diferentes visualizações baseadas em várias características para obter informações sobre avaliação de terras agrícolas.

Outra aplicação interessante é o *rgee* (Aybar et al., 2020) que é um pacote de funções implementados para a linguagem R utilizando a API Python do GEE para tornar possível a utilização do GEE através da linguagem R.

3 Solução Desenvolvida

Para solucionar o problema de carga automática dos mapas no servidor do aplicativo, foi utilizada metodologia DSRM, que prioriza a geração de conhecimento durante execução de projetos de pesquisa que resolvam problemas reais (Oyelere et al., 2018). A definição de todas as etapas de trabalho definidas pela DSRM podem ser vistas na Figura 4. Cada etapa será abordada nas próximas seções, que apresentarão o desenvolvimento da pesquisa de acordo com a DSRM.

Vale ressaltar que o APP *Regador* foi desenvolvido utilizando a metodologia DRSM (Silva et al., 2022). Como o DSRM segue uma abordagem de solução de problemas, é importante avaliar os artefatos para uma compreensão mais clara das melhorias que podem ser realizadas (Markus et al., 2002). Por isso a DSRM sugere um processo onde a construção e avaliação do artefato geralmente é iterado várias vezes antes de alcançar a versão final. Como visto na Figura 4, após a Avaliação é sugerido voltar à etapa de Modelagem do Artefato, caso seja necessário. Portanto, este trabalho refere-se à segunda iteração no processo de implementação do *Regador*, conforme representado na Figura 4.

3.1 Objetivo da Solução

Esta seção aborda as etapas de "Identificação do Problema"e "Definição dos Objetivos da Solução". A partir da definição do problema a ser resolvido e do conhecimento das possíveis ferramentas a serem utilizadas, foram definidos os objetivos da solução. A partir dessa análise, deve ser definida qual a inovação do artefato e qual a diferença dele para a atual solução existente.

Esse trabalho relata a segunda iteração do DSRM na implementação do APP *Regador*, que foi focada exclusivamente em automatizar a carga dos mapas de potencial hídrico no servidor de arquivos do aplicativo. Sendo assim, as seguintes ações foram executadas:

- **Avaliação do artefato atual:** objetiva entender o artefato atual, sua implementação, seus problemas, suas necessidades e possíveis melhorias.
- **Avaliação das ferramentas disponíveis:** objetiva avaliar as ferramentas disponíveis que poderão ser utilizadas para a solução dos problemas avaliados no passo anterior.
- **Análise de trabalhos correlatos:** objetiva analisar trabalhos correlatos que utilizaram as ferramentas avaliadas no passo anterior.

Para avaliar o artefato atual, foi realizado um estudo da sua arquitetura, de sua implementação, ferramentas utilizadas e serviços na nuvem utilizados no desenvolvimento. Dessa maneira foi possível entender o que era possível, o que era necessário, o que valeria a pena testar para solucionar o problema proposto e possíveis pontos de melhoria para o aplicativo que não são abordados neste trabalho.

Durante a avaliação das ferramentas foram avaliadas as APIS disponibilizadas pelo GEE, e que poderiam ser usadas na integração com o *back-end* do aplicativo.

Para a realização do trabalho, foram consultadas referências científicas que utilizaram as mesmas ferramentas ou que

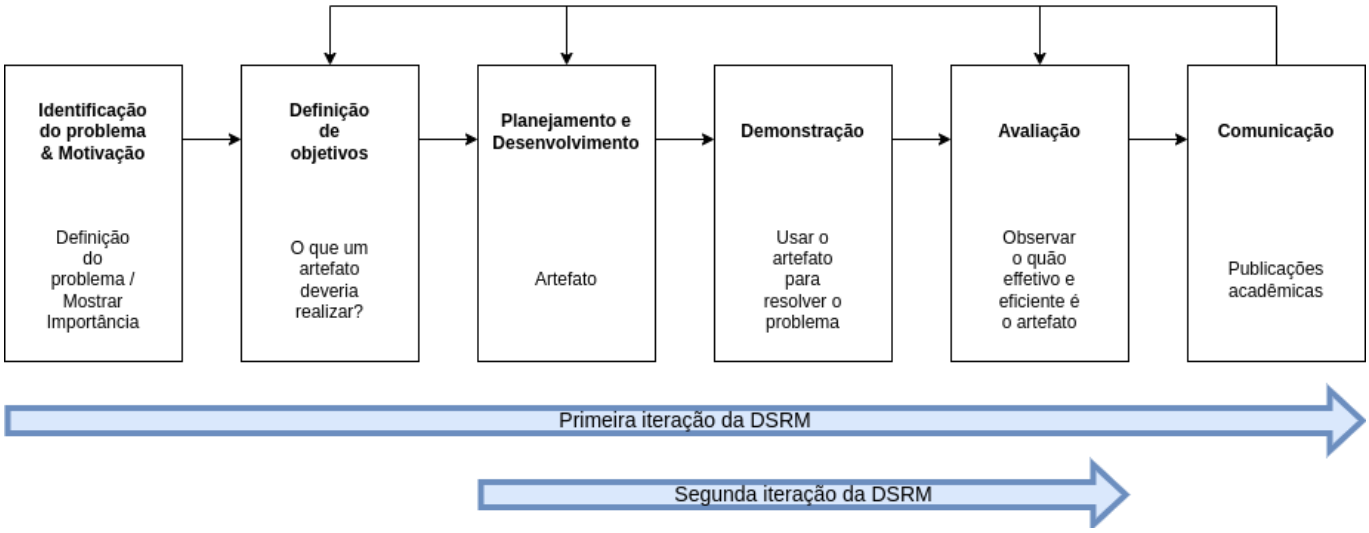

Figura 4: Modelo DSRM - Adaptado de: (Boné et al., 2020)

resolveram problemas semelhantes. Os trabalhos foram apresentados na Seção 2.4.

3.2 Modelagem e Desenvolvimento do Artefato

Esta seção se refere a etapa de "Modelagem e Desenvolvimento"descrita na abordagem DSRM (Figura 4). Após a identificação do problema e análise da arquitetura do aplicativo demonstrada na Figura 2, foi possível mapear os requisitos de softwares que a solução deveria cumprir, assim como a nova arquitetura do aplicativo e ferramentas que seriam utilizadas para o desenvolvimento da solução.

Os requisitos mapeados para a resolução do problema foram:

- **[REQ01] Desenvolvimento do modelo:** O script para gerar o modelo de estimativa do potencial hídrico deve ser traduzido para python e ser armazenado no código fonte do *back-end* do aplicativo.
- **[REQ02] Execução automatizada do script:** O script deve ser executado a cada nova passagem do satélite Landsat 8 para garantir a temporalidade do dado no aplicativo.
- **[REQ03] Manter imagens:** As imagens devem ser mantidas em um servidor de arquivos para que estejam disponíveis para serem consumidas pelo aplicativo.
- **[REQ04] Consumir imagens do servidor de arquivos:** O servidor *back-end* do aplicativo deve consumir as imagens salvas no servidor de arquivos.
- **[REQ05] Desempenho do aplicativo:** A nova arquitetura deve ser otimizada para manter o aplicativo funcionando de maneira semelhante a antiga arquitetura.

Após a definição dos requisitos, a nova arquitetura foi definida e está descrita na Figura 5. O desenvolvimento da solução iniciou-se a partir da conversão do script original de obtenção dos mapas. O novo script utiliza a API python do GEE e uma biblioteca chamada *geemap*[5], que é uma biblioteca que facilita o uso da API python do GEE tornando seus métodos parecidos com a API javascript. Dessa forma ele foi versionado juntamente com código fonte do *back-end* do aplicativo.

Utilizando essa API é possível salvar o mapa gerado localmente. Dessa maneira, a automatização da geração do mapa foi realizada. O próximo passo foi automatizar a carga do mapa para o servidor de arquivos do *Regador*. Como explicado na Seção 2.1, a plataforma Heroku na qual o APP está hospedado não possui um serviço para armazenamento de arquivos e recomenda o uso do *Amazon Simple Storage Service* (Amazon S3) que é o servidor de arquivos em nuvem da Amazon. Sendo assim, foi implementada uma função no script que utiliza a biblioteca boto3[6], para fazer o *upload* do mapa gerado pelo modelo diretamente para o servidor de arquivos S3. A biblioteca boto3 é uma biblioteca python para manipulação de serviços da *Amazon Web Services* (AWS).

Após esse desenvolvimento, todas as imagens geradas pelo *script* são armazenadas no servidor de arquivos da Amazon e recuperadas quando necessário.

O próximo passo foi criar uma maneira automatizada de rodar o script utilizado para rodar o modelo de estimativa de potencial hídrico a cada 16 dias, que corresponde à resolução temporal do satélite [7]. Servidores nuvem em sua maioria são sistemas UNIX, sendo assim a estratégia utilizada foi de usar o *cron* (Peters, 2009) que são agendadores de tarefas para sistemas UNIX. Porém o Heroku não permite manipulação da máquina virtual a nível de utilizar recursos nativos.

Nesse contexto, duas soluções foram testadas, sendo uma paga e uma gratuita. Para a solução paga, foi utilizado uma versão *trial* do *Add-on "Advanced Scheduler"*[8] que é uma ferramenta da plataforma heroku que agenda tarefa da mesma maneira que o cron. Sendo possível agendar tarefas utilizando o mesmo tipo de expressão do cron (minuto | hora | dia do mês | mês | dia da semana) que seria: 0 0 */16 * *, para rodar o script no minuto 0, na hora 0, a cada 16 dias. Onde os caracteres *

[5] https://geemap.org/
[6] https://github.com/boto/boto3
[7] A resolução temporal é definida em função do tempo de revisita do sensor para um mesmo ponto da superfície terrestre.
[8] https://elements.heroku.com/addons/advanced-scheduler

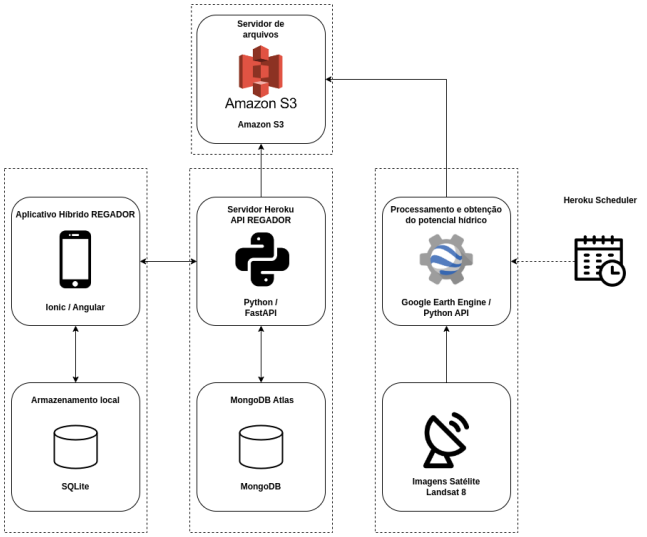

Figura 5: Nova arquitetura do aplicativo.

significam qualquer valor.

Para a solução gratuita foi utilizado o *Heroku scheduler*[9], que é menos flexível que o pago no que diz respeito às opções de agendamento. Ele fornece apenas opções de agendamento diário, por hora ou a cada 10 minutos. Nesse caso, foi agendado uma tarefa diária para rodar às 0h e o *script* que gera o modelo de potencial hídrico foi alterado para verificar a data da imagem mais recente presente no servidor de arquivos. Caso a imagem mais recente tenha menos de 16 dias ele não gera uma nova imagem. Sendo assim foi possível agendar a execução do *script* sem a necessidade de pagar por um serviço extra.

Desta maneira os requisitos **[REQ01]**, **[REQ02]** e **[REQ03]** da solução foram satisfeitos. Para finalizar o desenvolvimento da solução, foram feitas adaptações ao *back-end* do aplicativo para que consumisse as imagens com os valores de potencial hídrico do servidor de arquivos S3.

Para satisfazer o **[REQ04]** foram implementadas duas novas funcionalidades. A primeira funcionalidade é uma verificação: Sempre que for necessário acessar o servidor de arquivos para recuperar uma imagem, antes de fazer essa requisição ele verifica se a imagem já existe localmente. Caso exista ele apenas pega ela localmente e economiza uma requisição a um serviço externo.

A segunda é sobre a reinicialização dos dynos (container)[10]. Qualquer aplicação hospedada no Heroku funciona dentro de um dyno e esses dynos são reiniciados diariamente. Por padrão nenhum dyno existe por mais de 24h. Sendo assim, de acordo com a arquitetura atual do aplicativo, foi implementada uma função para automaticamente baixar as imagens presentes no S3 para a pasta local do aplicativo. Dessa maneira, na maioria das vezes que uma imagem for requisitada pelo aplicativo, ela já existirá localmente e raramente será necessário acessar o S3 e baixar uma imagem durante o uso do aplicativo.

4 Avaliação

A etapa de avaliação é o momento de medir o quanto o artefato está alinhado com a solução do problema. Para testar o artefato, foi proposto duas avaliações em relação a custo e desempenho. Isso porque o objetivo desse ciclo iterativo no desenvolvimento do aplicativo não interferiu na interface do mesmo. Portanto, não houve necessidade de novos testes com usuários.

4.1 Custo

Na arquitetura original do aplicativo, a plataforma em nuvem escolhida para servir o *back-end* foi o plano gratuito do Heroku. O MongoDB foi hospedado em uma plataforma gratuita chamada Atlas, e o script do modelo gerador do potencial hídrico não possui custo pois é executado manualmente. Sendo assim, a ferramenta utilizando a arquitetura original não gerava custos.

Na nova arquitetura duas novas ferramentas foram introduzidas, o *Heroku Scheduler* que é uma ferramenta limitada porém gratuita; e o servidor de arquivos S3 da Amazon. Com isso é possível concluir que a nova arquitetura gerará custos a serem pagos para manter o aplicativo funcionando, devido a necessidade do uso do S3. O sistema de cobrança da Amazon tem um valor mínimo que deve ser usado para iniciar as cobranças. No caso do S3, as cobranças começam a partir de 5GB de dados armazenados após 12 meses do início de uso do produto. Cada imagem gerada pelo script precisa de aproximadamente 5.5MB de espaço para ser armazenado, e uma imagem será gerada a cada 16 dias. Sendo assim, é possível concluir que o aplicativo levará aproximadamente 8 anos para atingir seu primeiro GB de dados armazenados no S3 como demonstrado no gráfico da Figura 6.

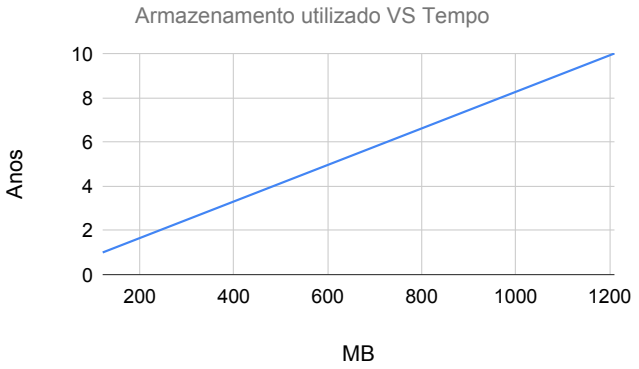

Figura 6: Armazenamento utiliza VS tempo

Quando o aplicativo atingir o consumo mínimo e começar a gerar cobranças, os valores não tendem a ser muito altos. A Tabela 2 traz os valores atuais de cobrança da Amazon.

Tabela 2: Tabela de preços *S3 Standard*[11]

Espaço utilizado	Preço do armazenamento
Primeiros 50 TB/mês	0,023 USD por GB
Próximos 450 TB/mês	0,022 USD por GB
Mais de 500 TB/mês	0,021 USD por GB

[9] https://devcenter.heroku.com/articles/scheduler
[10] https://www.heroku.com/dynos
[11] https://aws.amazon.com/pt/s3/pricing

Sendo assim é possível concluir que apesar de o aplicativo utilizar recursos que geram cobranças, essa cobrança será muito baixa, ou seja, a nova arquitetura se mantém viável.

4.2 Desempenho

O requisito **[REQ04]** exige que a nova arquitetura mantenha o desempenho do aplicativo semelhante ao desempenho da antiga arquitetura. Para garantir que esse requisito foi cumprido foram realizados testes de *stress* no *endpoint* alterado na implementação dessa solução. A máquina utilizada para realizar os testes foi um laptop Lenovo Ideapad, com 16GB de memória RAM e processador I5 7200U de 2.5GHz.

O endpoint original é o endpoint que funciona na arquitetura original do aplicativo demonstrado na Figura 2. Já o endpoint novo é o implementado neste trabalho para solucionar o problema de carga de dados do aplicativo, e esta descrito na Figura 5.

Foram realizados testes de stress com a ferramenta Apache JMeter. Os testes foram realizados realizando uma requisição por usuário e repetido 100 vezes para cada etapa. A primeira etapa se iniciou com 1 usuário e esse valor foi dobrado a cada etapa. Para exemplificar, primeira etapa foi realizada com um usuário e repetida 100 vezes, logo, a primeira etapa realizou 100 requisições. A segunda etapa foi realizada com 2 usuários, logo, foram realizadas 200 requisições.

Para a criação do Figura 7 foi utilizada a latência média da quantidade de requisições em cada etapa pela quantidade de usuários de cada etapa. Antes de cada etapa os ambientes eram reiniciados para garantir que não houvesse ruídos entre os testes. Os resultados podem ser vistos no gráfico da Figura 7.

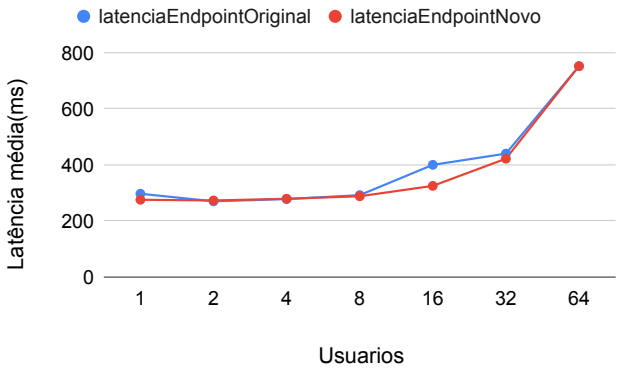

Figura 7: Teste com 100 repetições

Com os resultados dos testes, é possível afirmar que o desempenho do aplicativo não sofreu grandes alterações com a nova arquitetura. Ambos os testes foram realizados até a primeira falha acontecer.

Com os testes foi possível identificar uma falha na arquitetura do aplicativo, relacionada aos recortes das regiões demonstradas na Figura 1. Sempre que uma chamada é feita para retornar o potencial hídrico, um novo recorte é feito e armazenado no servidor. Sendo assim, a cada requisição uma nova imagem é armazenada levando o servidor a sobrecarga. Sendo assim a quantidade de usuários simultâneos está diretamente ligada a quantidade de requisições já feitas. Para exemplificar, os mesmos testes foram feitos com apenas 20 repetições. Dessa maneira menos requisições são feitas por etapa permitindo elevar o numero de usuários simultâneos. Como demonstrado no Figura 8.

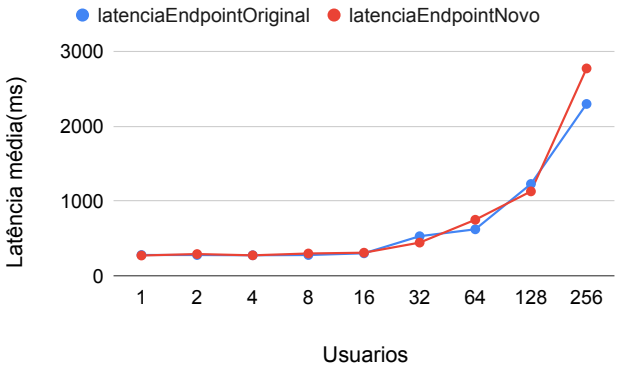

Figura 8: Teste com 20 repetições

Com os resultados demonstrados nessa seção, foi confirmado que a solução desenvolvida nesse trabalhos foi eficiente para resolver os problemas propostos.

5 Considerações finais

Este trabalho desenvolveu uma solução para automatizar a geração e carga dos mapas de potencial hídrico para o aplicativo *Regador* (Silva et al., 2022). A hipótese do trabalho foi corroborada, uma vez que a solução foi proposta, implementada e, de acordo com as avaliações realizadas, cumpriu seu propósito satisfatoriamente.

A API Python fornecida pelo GEE mostrou-se adequada para integrar aplicações com dados fornecidos pelo GEE. O desenvolvimento do código python a partir do código em Javascript pode trazer algumas dificuldades, uma vez que, o ambiente Javascript do *Code editor* é muito mais utilizado na comunidade, e ele possui um mapa integrado. Ele possui muitos métodos de manipulação e exibição de mapas que não existem nativamente na API python. Sendo assim os métodos são menos intuitivos e necessitam de mais linhas de código para realizarem a mesma tarefa.

Porém existem ferramentas como a biblioteca *geemap* que faz os métodos do GEE na API python apresentarem os mesmos nomes e conteúdos, e também o *Jupyter notebook*, que cria um ambiente de desenvolvimento *WEB* que, se usado junto com *geemap* os mapas serão exibidos e manipulados de maneira semelhante a do ambiente WEB do *Code editor*.

A solução utilizada para disparar a execução do *script* foi uma tarefa agendada utilizando o *Heroku Scheduler* que nada mais é que uma versão do *cron*, agendador de tarefas de sistemas Unix. O Cron não é indicado para ser usado em sistemas reais uma vez que ele pode falhar facilmente por vários motivos como horários de verão ou máquina sobrecarregada. Além disso, as falhas de uma tarefa agendada no cron é muito difícil de ser depurada, uma vez que ele não gera logs e também não salva o responsável por alterá-lo. Porém, neste contexto foi utilizado pois é uma solução pequena, que não sofre alterações com frequência e a tarefa agendada não rodará muitas

[11] https://jupyter.org/

vezes. Logo não sofrerá muitos danos com uma possível falha do Cron.

De acordo com os testes de stress realizados, um importante ponto de melhoria foi identificado, que é, a forma como o aplicativo salva os recortes dos campos de cada usuário. A cada chamada do *endpoint* os recortes são gerados em tempo de execução e ficam armazenadas no mesmo diretório dos mapas gerando desperdício de espaço de armazenamento do servidor, e também, aumentando o tempo de busca dos mapas neste diretório tornando o *endpoint* cada vez mais latente. Sendo assim, como principal ponto de melhoria para um trabalho futuro é importante repensar na estrutura de recuperação de recortes dos campos usuário para não causar problemas de performance no servidor. Um segundo ponto de melhoria para um trabalho futuro é estudar a migração de toda a infraestrutura do aplicativo do heroku para uma plataforma que oferece todos os serviços que ele necessita, uma vez que em breve o nível básico de uso gratuito do heroku deixará de existir, e uma vez que não será possível manter o aplicativo de maneira gratuita, deve-se estudar a melhor maneira de mantê-lo funcionando.

O aplicativo *Regador* é uma importante ferramenta para auxiliar os agentes da cadeia de produção de café a tomarem decisões de manejo em suas lavouras. O objetivo do aplicativo é dar informações atualizadas sobre a lavoura, e por isso, a automatização dos mapas consumidos pelo aplicativo é primordial. Sendo assim, esse trabalho contribuiu para a evolução do *Regador*, beneficiando a cadeia produtiva do café em Minas Gerais. O aplicativo pode ser acessado em https://drive.google.com/drive/folders/1C8DgEhAXDoQb1IkSr2IICyh1m7oPfkK7?usp=sharing

Referências

Araújo Júnior, G., Gomes, F., Silva, M., Maniçoba da Rosa Ferraz Jardim, A., Simões, V. J. L., Izidro, J., Leite, M., Teixeira, V., & Silva, T. (2019). Estresse hídrico em plantas forrageiras: Uma revisão. *Pubvet*, 13, 1–10.

Aybar, C., Wu, Q., Bautista, L., Yali, R., & Barja, A. (2020). rgee: An r package for interacting with google earth engine. *Journal of Open Source Software*, 5(51), 2272.

Bade, B. (2020). Google earth engine comparison between python and javascript. Disponível em: https://bikeshbade.com.np/tutorials/Detail/?title=Google+earth+engine+Detail+comparison+between+Python+and+JavaScript&code=9; Acessado em: 07/05/2022.

Barkmeyer, E., Barnard Feeney, A., Denno, P., Flater, D., Libes, D., Steves, M., & Wallace, E. (2008). Concepts for automating systems integration. *NIST Interagency/Internal Report (NISTIR) 6928*.

Barlow, N., Smith, C. R., Stroud, S. V., Abrams, J. F., Boulton, C. A., & Buxton, J. (2020). pyveg: A python package for analysing the time evolution of patterned vegetation using google earth engine. *Journal of Open Source Software*, 5(55), 2483.

Batista, L. A., Guimarães, R. J., Pereira, F. J., Carvalho, G. R., & Castro, E. M. d. (2010). Anatomia foliar e potencial hídrico na tolerância de cultivares de café ao estresse hídrico. *Revista Ciência Agronômica*, 41, 475–481.

Boné, J., Ferreira, J. C., Ribeiro, R., & Cadete, G. (2020). Disbot: a portuguese disaster support dynamic knowledge chatbot. *Applied Sciences*, 10(24), 9082.

Chaudhary, P. (2018). Ionic framework. *Int. Res. J. Eng. Technol*, 5(05), 3181–3185.

Coito, T., Martins, M. S., Viegas, J. L., Firme, B., Figueiredo, J., Vieira, S. M., & Sousa, J. M. (2020). A middleware platform for intelligent automation: An industrial prototype implementation. *Computers in Industry*, 123, 103329.

Correia, S. (2014). Potencial hídrico. *Revista de Ciência Elementar*, 2(1).

Dubois, O. et al. (2011). *The state of the world's land and water resources for food and agriculture: managing systems at risk*. Earthscan.

Gorelick, N., Hancher, M., Dixon, M., Ilyushchenko, S., Thau, D., & Moore, R. (2017). Google earth engine: Planetary-scale geospatial analysis for everyone. *Remote Sensing of Environment*, 202, 18–27. Big Remotely Sensed Data: tools, applications and experiences.

Grohs, D. S., Bredemeier, C., Mundstock, C. M., & Poletto, N. (2009). Modelo para estimativa do potencial produtivo em trigo e cevada por meio do sensor greenseeker. *Engenharia Agrícola*, 29, 101–112.

Hughes, J. N., Annex, A., Eichelberger, C. N., Fox, A., Hulbert, A., & Ronquest, M. (2015). Geomesa: a distributed architecture for spatio-temporal fusion. Em *Geospatial informatics, fusion, and motion video analytics V*, volume 9473 (pp. 128–140).: SPIE.

Krishnan, A. (2022). Geospatial app with google earth engine and greppo. Disponível em: https://towardsdatascience.com/geospatial-app-with-google-earth-engine-and-greppo-2c166b373382; Acessado em: 02/07/2022.

Maciel, D. A., Silva, V. A., Alves, H. M. R., Volpato, M. M. L., Barbosa, J. P. R. A. d., Souza, V. C. O. d., Santos, M. O., Silveira, H. R. d. O., Dantas, M. F., Freitas, A. F. d., et al. (2020). Leaf water potential of coffee estimated by landsat-8 images. *Plos one*, 15(3), e0230013.

Markus, M. L., Majchrzak, A., & Gasser, L. (2002). A design theory for systems that support emergent knowledge processes. *MIS quarterly*, (pp. 179–212).

Massruhá, S. M. F. S. & Leite, M. d. A. (2017). Agro 4.0-rumo à agricultura digital. Em *Embrapa Informática Agropecuária-Artigo em anais de congresso (ALICE)*: In: MAGNONI JÚNIOR, L.; STEVENS, D.; SILVA, WTL da; VALE, JMF do; PURINI, SR ....

Montero, D. (2021). eemont: A python package that extends google earth engine. *Journal of Open Source Software*, 6(62), 3168.

Oyelere, S. S., Suhonen, J., Wajiga, G. M., & Sutinen, E. (2018). Design, development, and evaluation of a mobile learning application for computing education. *Education and Information Technologies*, 23(1), 467–495.

Panidi, E., Rykin, I., Kikin, P., & Kolesnikov, A. (2020). Clouddesktop remote sensing data management to ensure time series analysis, integration of qgis and google earth engine. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B4-2020, 553–557.

Peters, R. (2009). cron. *Expert Shell Scripting*, (pp. 81–85).

Rodrigues, L. (2017). Quantidade de água utilizada na agricultura irrigada: certezas e incertezas nas estimativas.

Santos, B., Alberto, A., Lima, T., & Santos, B. (2018). IndÚstria 4.0: Desafios e oportunidades. *Revista Produção e Desenvolvimento*, 4.

Schultz, B., Ferreira, R. M. P., Juinior[1], E. M., Cecarelli[1], I. F., & Guerra, J. B. (2019). Datasafra–monitoramento de milho safrinha no mato grosso por sensoriamento remoto e google earth engine.

Silva, P., Souza, V., Volpato, M., & Silva, V. (2022). Regador: App for coffee water potential estimation. Em *Anais do XVIII Simpósio Brasileiro de Sistemas de Informação* Porto Alegre, RS, Brasil: SBC.

Silveira, H. R. D. O., Santos, M. D. O., Silva, V. A., Volpato, M. M., Alves, H. M. R., Dantas, M. F., & Carvalho, G. R. (2015). Relações entre índices de reflectância foliares e potencial hídrico de cafeeiro irrigado e de sequeiro. Em *Simpósio de Pesquisa dos Cafés do Brasil* Curitiba/PR: Embrapa Café.

Turner, C. R., Fuggetta, A., Lavazza, L., & Wolf, A. L. (1998). Feature engineering [software development]. Em *Proceedings Ninth International Workshop on Software Specification and Design* (pp. 162–164).: IEEE.

Vos, K., Splinter, K. D., Harley, M. D., Simmons, J. A., & Turner, I. L. (2019). Coastsat: A google earth engine-enabled python toolkit to extract shorelines from publicly available satellite imagery. *Environmental Modelling & Software*, 122, 104528.

Whitman, R. T., Park, M. B., Ambrose, S. M., & Hoel, E. G. (2014). Spatial indexing and analytics on hadoop. Em *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems* (pp. 73–82).

Yalew, S., van Griensven, A., & van der Zaag, P. (2016). Agrisuit: A web-based gis-mcda framework for agricultural land suitability assessment. *Computers and Electronics in Agriculture*, 128, 1–8.

Yu, J., Wu, J., & Sarwat, M. (2015). Geospark: A cluster computing framework for processing large-scale spatial data. Em *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems* (pp. 1–4).