

FATEC PROFESSOR JESSEN VIDAL

EQUIPE POLARIS

Visualizador de imagens de satélite

SÃO JOSÉ DOS CAMPOS - SP

2021

Lista de abreviaturas e siglas

Dev. Team - Development Team

P.O. - Product Owner

S.M. - Scrum Master

A.P.I. - Application Programming Interface

D.R.Y. - Don't Repeat Yourself

K.I.S.S. - Keep It Simple, Stupid

Y.A.G.N.I. - You Ain't Gonna Need It

P.O.O. - Programação Orientada a Objetos

C.S.S - Cascade Style Sheets

J.S. - JavaScript

Sumário

1 INTRODUÇÃO	4
1.1 Objetivos	4
1.1.1 Geral	4
1.1.2 Específicos	4
1.2 Metodologias aplicadas	5
1.2.1 SCRUM	5
1.2.2 Kanban	5
1.2.3 D.R.Y., K.I.S.S. e Y.A.G.N.I.	5
2 DESENVOLVIMENTO	7
2.1 Análise de projeto	7
2.1.1 Levantamento de requisitos	7
2.1.1.1 Product backlog	7
2.1.1.2 Sprint backlog	7
2.2 Projeto	8
2.2.1 Lógico	8
2.2.1.1 Elaboração da identidade visual e design do sistema	8
2.2.1.2 Desenvolvimento do wireframe e mockup	9
2.2.2 Físico	9
2.2.2.1 Desenvolvimento do protótipo	9
2.2.3 Codificação	10
2.2.3.1 Desenvolvimento das funcionalidades	10
2.2.3.1.1 RF 04: Desenvolvimento de interface com basemap de mapa e imagens de satélite	10
2.2.3.1.2 RF 05: Controle de interface pan, zoom in e zoom out	10
2.2.3.1.3 RF 01: Consulta às imagens definindo um ou mais satélites, área de interesse, período e cobertura de nuvens	11
4 CONCLUSÃO	14
5 ANEXOS	15
5.1 Product Backlog	15
5.2 Sprint Backlog	16
5.3 Mockup	16
5.3.1 Paleta de cores, tipografia e componentes	16
5.3.2 Tela inicial (home)	17
5.3.3 Tela de busca	18
5.3.4 Tela de download	18
5.4 Classe “SatSearchController”	19

1 INTRODUÇÃO

O tema abordado por este projeto se dá com a visualização, manipulação e download de imagens geradas por satélites específicos a pedido da empresa parceira, já que a mesma não possui um sistema próprio para o consumo dessas imagens a partir de seus repositórios em nuvem.

Como solução, o grupo constituído pelo Development Team Caio, Gabriel, Otávio, João, Thiago e Giovana, Product Owner Monique e Scrum Master Maria Gabriela, desenvolverão, no prazo de 4 sprints, cada qual com 21 dias, uma documentação completa (a qual refere-se este documento), uma interface gráfica e uma API para o consumo e visualização das imagens, visando ser uma solução acessível e proporcionando uma experiência tão boa quanto as que outros serviços (atualmente feitos e mantidos por terceiros) oferecem.

1.1 Objetivos

1.1.1 Geral

Produzir um site capaz de consumir imagens de satélite armazenadas em repositórios públicos, permitindo sua visualização e manipulação, bem como o download com aplicação de determinados filtros disponíveis.

1.1.2 Específicos

O produto deste projeto deve resultar em um sistema que permita:

- Consulta aos catálogos públicos de imagens dos satélites Landsat 8, Sentinel 1, Sentinel 2, CBERS4, CBERS4A e Amazônia 1;
- Geração de serviços web de tiles de imagens em nuvem;
- Exibição, manipulação e download das imagens em um site através de uma interface de mapa.

1.2 Metodologias aplicadas

Como metodologias ágeis para desenvolvimento e organização do projeto foram utilizadas o SCRUM e Kanban, já para desenvolvimento do produto foram aplicados conceitos como D.R.Y., K.I.S.S. e Y.A.G.N.I.

1.2.1 SCRUM

Metodologia ágil que divide as responsabilidades entre papéis em um mesmo grupo, onde os integrantes podem ser do Dev Team, que irão desenvolver o projeto propriamente dito; P.O., que estará sempre em contato direto com o cliente para alinhamento das necessidades e S.M., que tem o dever de coordenar e extinguir impedimentos de trabalhos do Dev Team. Os períodos de desenvolvimento são chamados de “sprints”, onde no projeto atual se configuram como 4 no total, com 21 dias cada, e a cada fim de sprint é gerado uma entrega para validação com o cliente, mantendo sempre a transparência e rapidez perante mudanças de planejamento.

1.2.2 Kanban

Método de organização de tarefas que consiste na classificação de cada atividade em “*To do*”, que representam as tarefas pendentes, “*In progress*”, onde estão alocadas tarefas que estão atualmente em desenvolvimento e “*Done*”, fase final, onde a tarefa se encontra concluída de acordo com critérios definidos anteriormente pelo próprio time.

1.2.3 D.R.Y., K.I.S.S. e Y.A.G.N.I.

Conceitos considerados boas práticas na programação, onde D.R.Y. significa “*Don’t Repeat Yourself*”, um princípio que incentiva a reutilização de código. Já K.I.S.S. significa “*Keep It Simple, Stupid*”, que prega o uso de lógicas com o mínimo de complexidade possível, enquanto que Y.A.G.N.I. significa “*You Ain’t Gonna Need It*”, que provê a ideia de não desenvolver coisas pensando que no futuro serão usadas, mas na verdade não são

necessárias, onde todos estes conceitos visam facilitar o entendimento do código e aumentar a produtividade dos desenvolvedores.

1.3 Tecnologias escolhidas

O produto foi desenvolvido utilizando-se as seguintes tecnologias:

- HTML e CSS, linguagens de marcação e estilização, respectivamente, de páginas web;
- React.js, biblioteca do JavaScript que permite a criação de interfaces gráficas com a sintaxe JSX;
- Node.js, uma tecnologia que permite o uso do JavaScript no “lado do servidor” (pensando na arquitetura cliente-servidor);
- TypeScript, um superset de JavaScript que permite o uso de tipagem e aplicação de conceitos do paradigma da Programação Orientada a Objetos;
- Leaflet, uma biblioteca do JavaScript usada para construir aplicativos que envolvam mapas;
- MongoDB, banco de dados utilizado para testes com crawlers que consumiam imagens dos repositórios e armazenavam informações sobre tais imagens;
- Axios, uma biblioteca do JavaScript que permite integração de um projeto em React com um serviço de API;
- Git, um sistema de versionamento para as entregas a cada sprint, onde os códigos e versões estão disponíveis também através do GitHub da equipe (acesso por: <https://github.com/Equipe-Polaris-DSM-2021>).

2 DESENVOLVIMENTO

2.1 Análise de projeto

2.1.1 Levantamento de requisitos

Assim que passada a cerimônia de “kick off” com a empresa, foi dada a largada para a aplicação do levantamento de requisitos, onde é tarefa do P.O. manter a comunicação com o cliente e extrair seus desejos e necessidades para o produto. Com isso em mente, foi iniciada a documentação de todo e qualquer requisito levantado, construindo em cima deles User Stories e classificando-os em requisitos funcionais e não funcionais.

2.1.1.1 Product backlog

Após concentrar certa quantidade de requisitos, a realização da organização deles em um backlog de produto ordenado e priorizado se fez necessária, acordando sempre junto do cliente o que é imprescindível e o que é desejável, resultando no Product Backlog descrito pela figura **5.1** na seção de Anexos.

2.1.1.2 Sprint backlog

Com o Product Backlog em mãos, foi realizada a cerimônia de Sprint Planning, onde todo o time se encontra para estimar as tarefas e quebrá-las em tarefas menores para melhor execução de cada uma, separando-as entre as 4 sprints que iriam decorrer até o fim do projeto. A separação de cada requisito pelas sprints se encontra ilustrada na imagem **5.2** na seção de Anexos.

Bem como o Product Backlog, o Sprint Backlog não é um documento estático, já que novos requisitos podem surgir, fazendo o planejamento se adequar às novas necessidades. Por conta dessa facilidade de mudança perante novas informações, a cada final de sprint, visando a transparência e qualidade, ocorrem algumas cerimônias que podem alterar tais documentos. São elas: Sprint Review e Sprint Retrospective. A Review se concentra na análise do atendimento aos requisitos do produto, além da validação do cliente,

impactando no Product Backlog, enquanto que a Retrospective analisa o rendimento do time, podendo impactar no Backlog da Sprint a partir da avaliação da rapidez, qualidade e comunicação da equipe.

2.2 Projeto

2.2.1 Lógico

2.2.1.1 Elaboração da identidade visual e design do sistema

Como sendo uma solução proposta para o desafio oferecido pelo cliente, foram adotadas como cores principais as cores predominantes da empresa, o laranja e o branco, além de uma cor análoga para contraste, o azul, onde a paleta de cores, contendo todos os tons acordados, se encontra ilustrada a seguir:

Primary Dark #DDAA44	Secondary Dark #2288CC	Black #112233	Gray Light #EEF3F3
Primary #FFBB55	Secondary #44AAFF	Gray Dark #72787A	White #FFFFFF
Primary Light #FFDDAA	Secondary Light #AADDFF	Gray #B3C3CB	

Para a tipografia foi escolhida a família de fontes “Poppins”, que passa a sensação do moderno e simples, já que se constitui de uma fonte sem serifa e com diversas variações. Para visualizar as variações selecionadas, bem como o tamanho e cores aplicadas, observe a imagem abaixo:

Título principal (medium, 36)

Título secundário (medium, 24)

Texto (regular, 18)

Texto secundário (regular, 14)

Existem alguns elementos que, para transparecer consistência à interface, foram padronizados, tais como botões, ícones listas e campos de formulário, os quais podem ser checados nesta figura:



A criação destes elementos deram base para a organização dos mesmos em telas, as quais, juntas, formam o mockup do produto, que pode ser entendido melhor a partir do próximo tópico.

2.2.1.2 Desenvolvimento do wireframe e mockup

Com a pesquisa por referências, produtos semelhantes, estudo das funcionalidades e preferências do cliente, além da escolha da paleta de cores, tipografia e criação dos componentes, foi idealizado um layout para as telas iniciais, que, mais a frente, se transformaria em um protótipo codificado em React, visando dar ao usuário um feedback visual antes mesmo das funcionalidades reais serem desenvolvidas, onde o mockup, feito utilizando a ferramenta gratuita Figma, pode ser encontrado na no tópico **5.3** da seção de Anexos.

2.2.2 Físico

2.2.2.1 Desenvolvimento do protótipo

Validado o mockup com o cliente, iniciaram os trabalhos para transformá-lo em um protótipo, dando a experiência e interação ao usuário como se o mesmo tivesse em mãos o produto final. Para isso, foi utilizado o

React, confeccionando os componentes e criando as telas e servidor que iriam provê-las e consumir a A.P.I que ainda seria construída a partir da próxima sprint. Tal protótipo constituiu a entrega de valor final da primeira sprint, juntamente com toda a pesquisa sobre o contexto do desafio e seus conceitos.

2.2.3 Codificação

2.2.3.1 Desenvolvimento das funcionalidades

Dado o levantamento dos requisitos e elencados os mais valiosos, começaram os preparativos para o desenvolvimento de cada um deles, configurando o projeto e separando-o em duas frentes: frontend e backend, cada qual com seu repositório de códigos, onde o mesmo foi feito com o time, dividindo o Dev Team dentre as duas frentes a partir da habilidade e afeição que cada integrante mostrou por cada área do projeto. As funcionalidades foram desenvolvidas seguindo o planejamento descrito no Sprint Backlog (vide imagem na seção 5.2 de Anexos).

2.2.3.1.1 RF 04: Desenvolvimento de interface com basemap de mapa e imagens de satélite

A User Story deste requisito é “O usuário quer visualizar o terreno em forma de mapas ou imagens de satélite para analisar e navegar pelas áreas“, onde tal funcionalidade foi concluída com a criação do protótipo do produto, que provê a possibilidade de alternar entre visualizar o mapa a partir de um basemap ou utilizando as imagens de satélite, onde para observar as imagens de satélite é preciso fazer a consulta passando os filtros de satélite, área de interesse, período e cobertura de nuvens, assim selecionando do resultado da busca as imagens que deseja observar na interface.

2.2.3.1.2 RF 05: Controle de interface pan, zoom in e zoom out

A User Story deste requisito é “O usuário quer controlar a visualização do mapa através de pan, zoom in e zoom out para ver mais detalhes do terreno e/ou localizar a área“, onde o objetivo era disponibilizar uma forma de o usuário realizar estas ações na interface de mapa. Tal funcionalidade foi implementada

através do Leaflet, que integra tais ações nativamente ao mapa gerado, mostrando-se de fácil manutenção e manipulação.

2.2.3.1.3 RF 01: Consulta às imagens definindo um ou mais satélites, área de interesse, período e cobertura de nuvens

A User Story deste requisito é “O usuário quer fazer a busca pelas imagens selecionando um ou mais satélites, além de ter opção de filtrar por área, período e cobertura máxima de nuvens para filtrar de acordo com os parâmetros que necessita em cada consulta”, onde tal funcionalidade foi implementada utilizando Node.js e uma API externa chamada SAT-API, Development Seed, que facilita o consumo das imagens dos satélites Landsat 8 e Sentinel 2, também sendo criado um crawler de testes para consumo de outros satélites como o CBERS, já que este, por exemplo, não se encontra dentro das opções de consulta disponibilizadas pela API externa utilizada.

```
class MapFilterProvider extends Component {
  state = {
    tilesDynamicList: [],
    showTileList: false,
    boundingBox: [],
    imageUrl: "",
    imageOpacity: 0,
    imageBounds: [],
  };
  ...
}
```

Dentre os códigos do portal web, tem-se a criação da classe “MapFilterProvider”, onde se é tida uma variável de estado que armazena dados sobre as imagens e suas particularidades (como localização, cobertura de nuvens, opacidade e etc), havendo alguns métodos de atualização destes.

Nesta classe também é possível observar o método de “performFilteredSearch”, que recebe como parâmetro uma resposta de formulário (os filtros da busca escolhidos pelo usuário na interface do site), e estrutura a requisição para o backend com as constantes “bbox” (que armazena a localização desejada) e “inputBody” (que contém as informações do corpo da requisição, ou seja, os filtros de satélite, cobertura de nuvens e período desejados).

```
performFilteredSearch = async (form) => {
  this.setState({
    ...this.state,
    showTileList: false,
  });

  const bbox = [
    this.state.boundingBox[0].lng,
    this.state.boundingBox[0].lat,
    this.state.boundingBox[1].lng,
    this.state.boundingBox[1].lat,
  ];

  const inputBody = {
    satelliteOptions: form.satelliteOptions,
    bbox,
    date_initial: form.periodFilter["date-initial"],
    date_final: form.periodFilter["date-final"],
    cloudCover: form.cloudFilter["cloud-range"],
  };
};
```

Num bloco “try catch”, é realizada a chamada da requisição para a API criada, utilizando o método “POST” com a rota dos dados e a constante de “inputBody”, armazenando o resultado da busca numa constante, que por sua vez é passada como valor para o estado da classe no campo “tilesDynamicList”. Caso a ação não seja bem sucedida, o bloco “catch” captura o erro e o exibe no terminal.

```
try {
  const { data } = await api.post("/satSearch", inputBody);
  this.setState({
    ...this.state,
    showTileList: true,
    tilesDynamicList: data,
  });
} catch (error) {
  console.log(error);
}
```

Enquanto isso o método “render()” fica responsável por retornar o componente de “Context.Provider” com os valores atuais do estado da classe.

```
render() {
  return (
    <Context.Provider
      value={{
        ...this.state,
        performFilteredSearch: this.performFilteredSearch,
        setBoundingBox: this.setBoundingBox,
        setShowTileList: this.setShowTileList,
        showTileList: this.state.showTileList,
        tilesDynamicList: this.state.tilesDynamicList,
        imageUrl: this.state.imageUrl,
        imageBounds: this.state.imageBounds,
        imageOpacity: this.state.imageOpacity,
        handleImageOverlay: this.handleImageOverlay,
      }}
    >
      {this.props.children}
    </Context.Provider>
  );
}
```

Já no backend, com o Axios, foi possível fazer uso da API externa através da criação de um controller no backend, que contém uma classe “SatSearchController”, responsável pela consulta a partir do recebimento dos filtros passados pelo frontend. Nesta classe, pode-se observar um método “index” que recebe dois parâmetros: *req*, do tipo Request, e *res*, do tipo Response, onde em seu corpo há um bloco de “try catch” para obtenção das imagens ou o tratamento de alguma exceção ou erro.

No bloco “try”, temos a criação de uma constante que recebe o corpo da requisição feita pelo frontend, além de criar um vetor para armazenamento das imagens que seriam obtidas como resposta. Um laço de repetição é feito para cada satélite escolhido dentre as opções, criando uma variável “inputBody” que armazena as informações para a consulta na API externa da forma correta.

Assim é criada a constante “satCollection”, que guarda o resultado do método “POST” na API, passando como argumentos o caminho da busca e as variáveis ‘inputBody’ e ‘headers’ (cabeçalho da requisição), onde a resposta é armazenada no vetor “imagesResponse” e enviado como resposta ao frontend como JSON. Já se a operação não for bem sucedida, o bloco de “catch” captura o erro e o exibe no console (observe a classe “SatSearchController” na íntegra pela imagem na seção **5.4** de Anexos).


Com a resposta da requisição recebida pelo frontend, é utilizado o componente de “ResultsMenu” para a listagem das imagens recebidas, dando a opção para visualização na interface de mapa através do Leaflet e também a opção para realizar o download, porém essa funcionalidade ainda está para ser feita, com expectativa de conclusão na sprint 3.

4 CONCLUSÃO

Em breve.

5 ANEXOS

5.1 Product Backlog



Backlog do Produto

Requisitos Funcionais

Nº da sprint em que será realizado


→ Nº do requisito	User Story que o requisito faz referência ←		↑
RF 01	Consulta às imagens definindo um ou mais satélites, área de interesse, período e cobertura de nuvens máxima	#09	02
RF 02	Serviço de geração de tiles dinâmicos	#05	03
RF 03	Exibição das imagens em cores naturais e realçadas	#04	03
RF 04	Desenvolvimento de interface com basemap de mapa e imagens de satélite	#01	01
RF 05	Controle de interface pan, zoom in e zoom out	#03	01
RF 06	Ferramenta de download de imagens	#02	03
RF 07	Exibição das imagens em composição colorida falsa-cor e Índice de Vegetação de Diferença Normalizada (NVDI)	#08	04
RF 08	Ajuste de brilho, realce de contraste e transparência	#07	04
RF 09	Linha do tempo para imagens multi-temporais	#06	04

Requisitos Não Funcionais

RNF 01	Uso da linguagem de programação TypeScript orientada à objetos
RNF 02	Servidor em Node.js e uso de frameworks duas camadas (React)
RNF 03	Documentação
RNF 04	Uso de serviço em nuvem para geração de tiles dinâmicos de imagens
RNF 05	Uso de serviço em nuvem para consulta aos catálogos de imagens disponíveis em repositórios públicos em nuvem
RNF 06	Uso de tecnologias como PostgreSQL, Leaflet, AWS Lambda, STAC, Python, Node.js

Equipe Polaris – 2º DSM/ 2021
FATEC Profº Jessen Vidal – São José dos Campos, SP

5.2 Sprint Backlog



Backlog das Sprints

Planejamento das funcionalidades que serão entregues a cada sprint, baseado nos requisitos levantados e seu valor e importância para o cliente

SPRINT 1

- Desenvolvimento de interface com basemap de mapa e imagens de satélite
- Controle de interface pan, zoom in e zoom out

SPRINT 2

- Consulta às imagens definindo um ou mais satélites, área de interesse, período e cobertura de nuvens máxima

SPRINT 3

- Exibição das imagens em cores naturais e realçadas
- Serviço de geração de tiles dinâmicos
- Ferramenta de download de imagens

SPRINT 4

- Exibição das imagens em composição colorida falsa-cor e Índice de Vegetação de Diferença Normalizada (NDVI)
- Ajuste de brilho, realce de contraste e transparência
- Linha do tempo para imagens multi-temporais

Equipe Polaris – 2º DSM/ 2021
FATEC Profº Jessen Vidal – São José dos Campos, SP

5.3 Mockup

5.3.1 Paleta de cores, tipografia e componentes

Primary Dark #DDAA44	Secondary Dark #2288CC	Black #112233	Gray Light #EEF3F3
Primary #FFBB55	Secondary #44AAFF	Gray Dark #72787A	White #FFFFFF
Primary Light #FFDDAA	Secondary Light #AADDFF	Gray #B3C3CB	

Título principal (medium, 36)

Título secundário (medium, 24)

Texto (regular, 18) Texto secundário (regular, 14)

Filtro

Escolha um exemplo ▾

Exemplo

Exemplo

Exemplo

Exemplo

Exemplo

Exemplo

Lista de checkbox

☐ Exemplo

☐ Exemplo

☐ Exemplo


☐ Exemplo

☐ Exemplo

☐ Exemplo

+ − × ≡ → + − × ≡

Botão Botão



5.3.2 Tela inicial (home)



Polaris

Acessar o mapa

Um jeito simples e rápido de consumir imagens de satélites

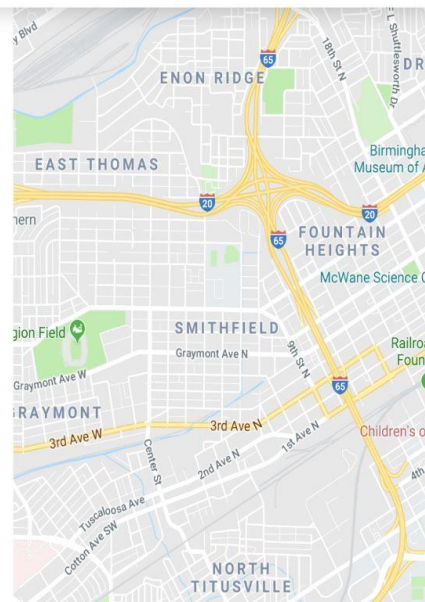
Com poucos cliques, acesse imagens de diversos satélites disponíveis, definindo as opções de visualização de acordo com sua preferência.

+

-

Mapa

Satélite



Satélites disponíveis

Disponibilizamos uma série de satélites para que você selecione de acordo com sua necessidade no momento. Veja abaixo as opções:

Landsat 8

Sentinel 1 e 2

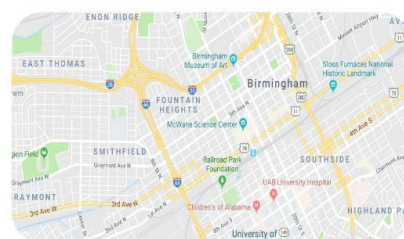
CBERS 4

CBERS 4A

Amazônia 1

Download de imagens selecionadas

Selecione a área de interesse, ainda é possível fazer o download da imagem da região de duas formas: como **imagem bruta** ou como **imagem processada**. Basta clicar no ícone da área de interesse, escolher uma das opções e fazer o download.



Polaris


Sobre o projeto

Acessar o mapa

Confira o GitHub do projeto

2021 © TODOS OS DIREITOS RESERVADOS

5.3.3 Tela de busca

**Polaris**

Satélite

☐

Landsat 8

☐

Sentinel 1

☐

Sentinel 2

☐

CBERS 4☐☐

Área de Interesse

Selecione a área de interesse no mapa

Período

Início

-

Fim

Cobertura de nuvens

Selecione a porcentagem de nuvens

Buscar

Mapa

Satélite



5.3.4 Tela de download

Resultados

Nome da imagem

Vizualizar

Baixar

Nome da imagem

Vizualizar

Baixar

Nome da imagem

Vizualizar

Baixar

Nome da imagem

Vizualizar

Baixar

Nome da imagem

Vizualizar

Baixar

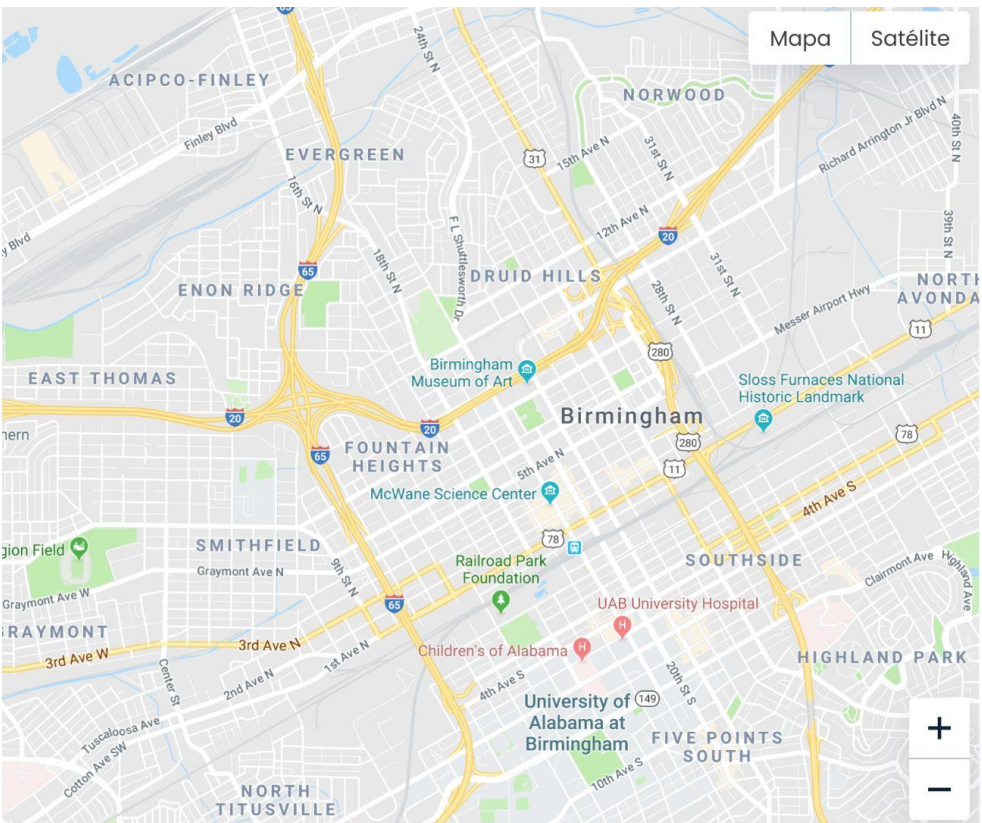
Nome da imagem

Vizualizar

Baixar

Mapa

Satélite



5.4 Classe “SatSearchController”

```
class SatSearchController {
  async index(req: Request, res: Response) {
    //using DevSeed's API
    try {
      const { satelliteOptions, bbox, cloudCover, date_initial,
date_final } = req.body

      const period = `${date_initial}/${date_final}`

      let imagesResponse:any = []

      for(let satellite of satelliteOptions) {
        console.log(satellite)
        let inputBody = {
          "bbox": bbox,
          "time": period,
          "intersects": null,
          "limit": 50,
          "query": {
            "eo:cloud_cover": {
              "lt": cloudCover
            },
            "collection": {
              "eq": satellite
            }
          },
          "sort": [
            {
              "field": "eo:cloud_cover",
              "direction": "desc"
            }
          ]
        };

        let headers = {
          'Content-Type': 'application/json',
          'Accept': 'application/geo+json'
        };

        const satCollection = await DEV_SEED.post(
          '/stac/search',
          inputBody,
          { headers }
        )

        imagesResponse.push({[satellite]:satCollection.data})
      }

      return res.json(imagesResponse)
    } catch (err) {
      console.log(err)
    }
  }
}
```