Relatório de atividades - Bolsa TT-4A - FAPESP

Investigar e Analisar a Cidade
INACITY

Artur André Almeida de Macedo Oliveira

Supervisor: R. Hirata Jr.

Processo Número: 2018/10767-0

1 Introdução

O projeto INACITY compreende uma plataforma para coleta e processamento de imagens urbanas através do uso de plataformas como o "Google Street View"e sistemas de informações geográficas como o "OpenStreetMap". Através do INACITY, usuários podem verificar, em uma grande área metropolitana, a concentração de características de relevância urbana (p. ex. árvores, rachaduras em edifícios históricos, etc.). Esse tipo de visualização possibilita aos usuários conscientizarem-se das condições de uma região e aos administradores da cidade tomarem decisões (p. ex. replantar árvores ou investir na manutenção de um edifício em risco) mais rapidamente.

Além disso a plataforma foi desenvolvida tendo-se em vista a possibilidade de se incluir facilmente novas fontes de imagens geo-localizadas, bases de dados geográficos e módulos para processamento de imagens, sendo assim a plataforma também contribui como ferramenta cientifica no contexto de cidades inteligentes.

O repositório oficial da plataforma INACITY [Oli] fica hospedado juntamente ao repositório do projeto InterSCity [Int]. Uma instância pública para uso da plataforma INACITY foi criada e pode ser acessada no endereço `https://inacity.org`.

2 Objetivos

1. Definir e configurar o ambiente, o processo e a arquitetura de software do aplicativo; ***(arquitetura)***

2. Projetar e implementar módulos de software para anotar (ex. desenhos livres e texto) um mapa, da mesma forma como é feito em um sistema de informações geográficas; ***(interação)***

3. Modelar e implementar um banco de dados para o gerenciamento de usuários e de suas sessões. Uma sessão compreende todos os passos realizados pelo usuário quando este interage com o sistema, de forma a permitir que uma sessão de trabalho possa ser consultada e apresentada posteriormente sem que o usuário precise selecionar regiões, imagens e/ou resultados novamente; ***(armazenamento)***

4. Projetar e implementar um painel do usuário para permitir que este veja uma lista de sessões previamente salvas e também retome uma destas sessões; ***(painel do usuário)***

5. Projetar e implementar mecanismos para segmentar requisições dos usuários e permitir a entrega de resultados parciais já processados; ***(segmentação)***

6. Otimização dos componentes de integração de diferentes plataformas (ex. OpenStreetMap e GSV) para diminuir o tempo de espera do usuário; ***(otimização)***

7. Projetar e implementar mecanismos para recuperação automática de falhas para que o usuário não precise refazer uma requisição longa no caso em que a conexão é perdida antes que a requisição tenha sido totalmente processada; ***(resiliência)***

3 Desenvolvimento

3.1 Definições gerais

Objetivos: ***(arquitetura)***, ***(interação)***

A plataforma deverá ser acessível por meio de um navegador Web, ter um banco de dados geo-localizado, processar e extrair informações de imagens urbanas e ser facilmente integrada com bases de informações geográficas assim como bases de imagens. Para tanto foi adotada a linguagem de programação Python 3 [Foub] por ser de uso livre, isso é sem a necessidade de uma licença paga, pelo grande número de módulos que este possui tanto para o processamento de imagens (ex. Numpy [Dev], Pandas [McK], scikit-image [vdWSN$^+$14] etc) e também pela possibilidade do uso do framework Django [Foua] e, usado para o desenvolvimento de aplicações web que façam uso de bancos de dados. Este último framework possui uma integração com o sistema de banco de dados PostgreSQL [Grob] o qual por sua vez possui um suporte nativo à dados geo-localizados, sendo portanto o Sistema Gerenciador de Banco de Dados Relacional (SGBDR) adotado para o projeto.

A fim de se obter uma noção geral de como modelar os diversos componentes de software da plataforma foram estudadas as plataformas Google Street View [LLC] com a finalidade de se projetar o componente para integração com bases de imageamento geo-localizado e a plataforma OpenStreetMap [Ope17] com a finalidade de se projetar o componente para integração com Sistemas de Informações Geográficas. O componente para processamento de imagens é inspirado no trabalho de mesmo nome INvestigate and Analyse a City-INACITY [OJ18], no qual foi desenvolvido um filtro para segmentação e quantização de vegetação em imagens.

3.2 Implementação do back-end

Objetivos: ***(arquitetura)***

Tendo-se em vista as definições e decisões tomadas acima, a primeira versão do servidor (também chamado de 'back-end') foi implementada usando-se o framework Django.

O back-end da plataforma INACITY consiste de componentes voltados à integração com outros sistemas externos, à comunicação com o uma aplicação de front-end (como um navegador Web por exemplo).

3.2.1 Sistemas de Informações geográficas

Objetivos: ***(arquitetura)***

Para realizar a integração com Sistemas de Informações Geográficas (SIGs) foi criado o componente chamado de 'MapMiner'. O componente específico responsável por intermediar a comunicação com a plataforma OpenStreetMap deriva do componente 'MapMiner' e se chama 'OSMMapMiner'. Componentes derivados do componente 'MapMiner' são responsáveis por responder à requisições que solicitem por informações geográficas dentro de uma região delimitada pelo usuário, esta região será referida como sendo uma região de interesse.

A fim de gerenciar os diversos possíveis componentes derivados de 'MapMiner' foi criado o componente 'MapMinerManager' que atua de forma a permitir que um usuário escolha em qual SIG a busca por informações geográficas deve ser feita.

Na figura 1 temos a classe 'MapMiner' agindo (segundo a nomenclatura de engenharia de software) como uma classe abstrata com duas subclasses. A motivação para se adotar esta arquitetura se deve ao fato de que facilmente pode-se manter um registro de quais são as subclasses de uma dada superclasse e com isso o processo de descoberta de novos componentes de integração com SIGs pode ser automatizado, bastando-se para isso que estes novos componentes sejam subclasses de 'MapMiner'.

3.2.2 Provedores de imagens urbanas

Objetivos: ***(arquitetura)***

Um provedor de imagens urbanas essencialmente corresponde à um repositório de imagens, ou mais especificamente, ao mecanismo que permite o acesso às imagens.

O componente criado para a integração da plataforma INACITY com um provedor arbitrário de imagens urbanas é o 'ImageProvider'. Este componente assim como o 'MapMiner' também é abstrato e suas subclasses são os componentes responsáveis pela integração com um provedor

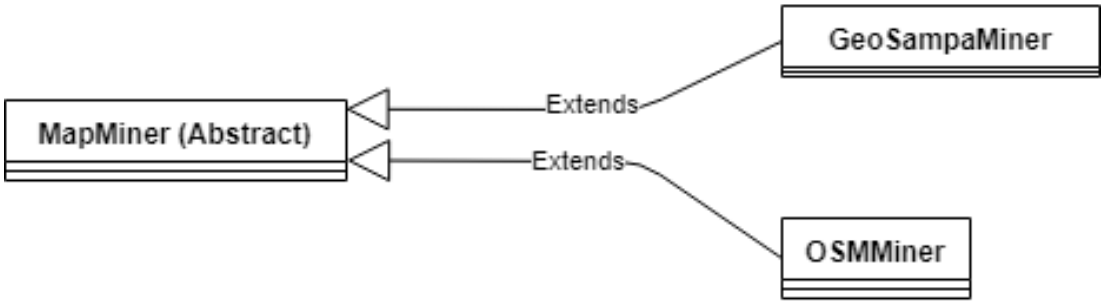

Figura 1: 'MapMiner' e subclasses

específico como por exemplo o Google Street View [LLC].

O componente 'ImageProvider' é responsável por manter um registro de todas as suas subclasses e cada uma de suas subclasses deve implementar além de propriedades para identificação (como nome do provedor de imagens) também uma função para coleta de imagens de uma dada coleção de pontos geográficos, por exemplo, uma vez que o usuário requisita pelas imagens de uma rua, ou de uma coleção de pontos de ônibus esta requisição será delegada para uma subclasse de 'ImageProvider' e esta deve então retornar uma coleção de imagens da rua requisitada ou da coleção de pontos de ônibus. A subclasse implementada de 'ImageProvider' integrada com o Google Street View na plataforma INACITY se chama 'GoogleStreetViewProvider'.

3.2.3 Processadores e filtros de imagens

Objetivos: ***(arquitetura)***

Os componentes para processamento/filtragem de imagens são implementados criando-se subclasses de 'ImageFilter'. A classe abstrata 'ImageFilter', assim como as demais classes abstratas desta subseção, também é responsável por manter um registro de quais são suas subclasses e estas por sua vez devem possuir um nome (como por exemplo 'Vegetação') indicando a finalidade desta subclasse e devem implementar dois tipos de requisição, um voltado ao processamento de imagens de coleções de objetos geográficos (como uma coleção de ruas) e outro voltado para o processamento de um único objeto geográfico (como uma única avenida).

Na versão atual da plataforma foi implementado o componente 'GreeneryFilter' que é uma subclasse de 'ImageFilter'. Este componente segue a especificação descrita em [OJ18], isso é, um filtro para imagens que marca regiões de vegetação e extrai um índice (conhecido como Green View Index na literatura [OJ18]) baseado na porcentagem da imagem que corresponde à áreas marcadas como vegetação.

3.2.4 Gerenciadores

Objetivos: ***(arquitetura)***

Componentes chamados de gerenciadores são responsáveis por direcionar requisições que chegam a partir de uma aplicação cliente (como o front-end) para o servidor e tem como destino um SIG, Provedor de imagens ou um Filtro de imagens. Há três componentes gerenciadores na plataforma INACITY, estes são: 'MapMinerManager', 'ImageProviderManager' e 'ImageFilterManger', responsáveis por delegar requisições para subclasses de 'MapMiner', 'ImageProvider' e 'ImageFilter' respectivamente.

Além de delegar requisições os componentes gerenciadores também são responsáveis por informar a um cliente quais são os componentes disponíveis no sistema, como por exemplo quais subclasses para integração com SIGs estão disponíveis.

Excepcionalmente o componente para gerenciamento de usuários, chamado de 'UserManager', não segue o padrão dos demais gerenciadores. Este último é responsável apenas pela criação de usuários e se integra com os demais componentes para gerenciamento de usuários do framework Django usado para criar a plataforma INACITY.

3.3 Implementação do front-end

Objetivos: ***(arquitetura)***, ***(armazenamento)***, ***(painel do usuário)***

Seguindo um modelo cliente-servidor o projeto possui uma parte responsável pela integração com outros sistemas e acesso ao banco de dados chamada de servidor ou também de back-end (definido na subseção 3.2). O back-end de um projeto normalmente não é diretamente usado pelo usuário final, o acesso ao back-end é feito através de uma aplicação cliente que também é denominada front-end no modelo cliente-servidor.

O front-end implementado na plataforma INACITY é responsável por:

- Exibir mapas digitais, marcações e características geográficas (como pontos de ônibus por exemplo);
- Exibir imagens processadas ou não de características geográficas;
- Realizar requisições ao servidor para a coleta de características geográficas e de imagens destas características;
- Permitir que o usuário selecione regiões de interesse sobre o mapa digital;

- Exibir quais são os SIGs e Provedores de Imagens disponíveis no back-end para que as requisições sejam feitas;
- Permitir que o usuário visualize e gerencie sessões de trabalho, estas são coleções de regiões de interesse e os dados coletados através de requisições ao back-end.

Tanto o front-end quanto o back-end são baseados na arquitetura conhecida como Model-View-Controller (MVC), ou seja, enquanto a nível de aplicação a plataforma INACITY é baseada na arquitetura cliente-servidor, as partes cliente e servidor são baseadas na arquitetura MVC.

3.3.1 Mapas digitais

Objetivos: ***(arquitetura)***, ***(interação)***

Através de um mapa digital o usuário pode selecionar uma região de interesse que contenha posições geográficas que lhe interessem. E também visualizar informações e características geo-localizadas como mapas de calor indicando a densidade de alguma quantidade (como a quantidade de vegetação em um dado local) ou a presença/ausência de algum elemento pontual como por exemplo uma boca de bueiro sem tampa.

Na plataforma INACITY é usada a Interface para Programação de Aplicativos (de sigla API em inglês) chamada OpenLayers.

A API OpenLayers permite a exibição de mapas digitais oriundos de diferentes provedores de mapas, a criação de marcações e anotações sobre os mapas, facilitando assim as tarefas de exibição tanto dos mapas quanto das características geográficas. Em sua implementação atual na plataforma INACITY são exibidos mapas de ruas dos provedores OpenLayers e Google Maps, e um mapa com imagens de satélite oriundo do Google Maps.

Além disso a API OpenLayers também da suporte à criação de mapas de calor, anotações e marcações sobre os mapas digitais, desta forma facilitando que parte dos objetivos do front-end sejam alcançados. Para se realizar o controle de marcações sobre os mapas fornecidos através do OpenLayers foi elaborado o componente OpenLayersHandler, isso é, o componente OpenLayersHandler é um componente que atua como um intermediador entre os demais componentes front-end da plataforma INACITY e os mapas do OpenLayers.

3.3.2 Consultas à Sistemas de Informações Geográficas

Objetivos: ***(interação)***

A seleção de uma região de interesse sobre um mapa digital é implementada através da API OpenLayers, uma vez selecionada uma região de interesse o usuário tem a opção (através do

modo de mapa) de coletar características geográficas oriundas de um ou mais Sistemas de Informações Geográficas(SIGs).

Na versão atual a plataforma INACITY conta com um componente para acesso ao SIG OpenStreetMap [Ope17] o qual permite a coleta de endereços (por exemplo ruas e avenidas) presentes dentro de uma região de interesse, e também conta com um componente para acesso ao conjunto de pontos de ônibus da cidade de São Paulo oriundo do SIG GeoSampa [dSP]. Instruções para a inclusão de outros SIGs e características geográficas podem ser encontradas no manual técnico da plataforma INACITY.

Toda consulta originada no front-end que seja direcionada ao back-end é realizada pelo componente chamado UIModel. Este componente se insere na camada de Modelo na arquitetura MVC. O componente UIModel e alguns outros também contidos na camada de Modelo são responsáveis por manter o estado do front-end, e também responsáveis por realizar consultas ao back-end ou eventualmente à outras plataformas externas.

3.3.3 Consultas à Provedores de Imagens

Objetivos: ***(interação)***

Assim como explicado na sessão 3.3.2 toda consulta ao back-end é realizada através do componente UIModel. Uma vez cadastrado no back-end um componente para integração com algum Provedor de Imagens, este deve ser disponibilizado ao front-end e a partir de então consultas por imagens de características urbanas podem ser realizas.

O primeiro Provedor de Imagens integrado à plataforma é o do Google Street View (GSV). Apesar de arquiteturalmente a plataforma ter sido planejada de forma que a integração com Provedores de Imagens devesse ocorrer somente através do back-end existem casos excepcionais em que a integração à um dado Provedor de Imagens não é possível através do back-end, como é o caso da plataforma Google Street View (GSV) devido a restrições em sua licença de uso que forçam o acesso através do front-end. Para se obter imagens oriundas do (GSV) criou-se o componente de front-end 'GSVService'. Este componente segue todas as diretrizes e regulamentos impostos pela licença de uso do GSV.

Visando a simples instalação da plataforma, de maneira que ela por padrão já dê suporte ao uso do GSV, informações a respeito de como modificar chaves de acesso e informações necessárias para uso de uma conta de terceiros do GSV numa nova instância da plataforma INACITY são fornecidos na documentação oficial técnica presente no repositório da plataforma INACITY [Oli] dentro do repositório InterSCity [Int].

3.3.4 Requisições de processamento de imagens

Objetivos: ***(arquitetura)***, ***(interação)***

Da mesma forma como são feitas as requisições por características geográficas e imagens também são feitas requisições para que imagens previamente coletadas sejam processadas a fim de se extrair alguma característica ou se obter algum tipo de transformação da imagem. O componente responsável por realizar a requisição, como nos demais casos, é o 'UIModel'.

Na versão atual o projeto conta com o filtro para vegetação. Este filtro seleciona pontos da imagem que correspondem a vegetação e os destaca usando uma cor esverdeada ao mesmo tempo que marca regiões classificadas como não sendo vegetação com uma cor azulada. Este esquema de cores foi adotado tendo-se em vista facilitar a visualização das imagens por pessoas com dificuldades visuais. O filtro de vegetação também calcula qual porcentagem da imagem foi classificada como vegetação e esta porcentagem, chamada de Índice de Visualização Verde [YZMG09], é usada para se criar o mapa de calor sobre o mapa digital, renderizado pelo componente 'OpenLayersHandler'.

3.3.5 Sessões de usuário

Objetivos: ***(arquitetura)***, ***(interação)***, ***(painel do usuário)***

Uma vez que o usuário selecione uma região de interesse, faça alguma requisição para o back-end ou altere o estado da aplicação um tipo especial de requisição é enviado para o back-end. Esta requisição envia para o servidor todas as variáveis de estado da aplicação incluindo:

- Características geográficas coletadas;
- imagens coletadas;
- imagens filtradas e métricas extraídas das imagens;
- Estado das regiões selecionadas (quais estão ativas e quais não);

O componente responsável por interagir com o back-end neste caso não é o 'UIModel' mas sim o componente 'SessionManager'. Uma vez que a sessão está salva ela é automaticamente carregada quando o usuário volta à página, ou caso o tempo da sessão tenha expirado, o usuário pode recuperar a sessão no painel do usuário, desde que tenha sido feito o login e a sessão salva esteja vinculada a um usuário cadastrado no sistema.

3.4 Banco de dados da plataforma INACITY

Objetivos: ***(arquitetura)***, ***(armazenamento)***, ***(otimização)***

O banco de dados a priori foi projetado para armazenar as sessões de usuário, contudo para agilizar as consultas de usuário, principalmente aquelas dependentes do processamento de imagens, alguns dados além daqueles dos usuários são armazenados no banco de dados também.

Além disso uma forma de se integrar outras bases de dados (SIGs ou Provedores de Imagens) é importando-se estas bases para dentro do banco de dados do INACITY como é o caso dos pontos de ônibus oriundos do GeoSampa. Na figura 2 é ilustrado o diagrama de entidade e relacionamentos da plataforma INACITY. Note que este diagrama não contém todas as entidades pertencentes ao framework Django, nele damos foco às entidades responsáveis pela persistência dos dados de usuário (tabelas User, Session, AbstractBaseSession e Session <AbstractBaseSession>), a entidade que representa os pontos de ônibus do GeoSampa (tabela GeoSampa_BusStops) e às entidades que detém informações a respeito dos dados extraídos das imagens (tabelas GeoImage e FilterResult) assim como referências às imagens em si (tabelas ImageBaseRef e ImageInstance).

A tabela 'Session' ligada diretamente à tabela 'User' armazena os dados relativos à sessão do usuário no campo 'uimodelJSON'. Neste campo são armazenadas de forma textuais todas as características usadas para reconstruir uma sessão de usuário, incluindo imagens e características geográficas carregadas na sessão. A vantagem principal desta abordagem é a facilidade na alteração do formato da sessão de usuário e também a possibilidade de se fazer consultas em objetos no PostgreSQL. Este tipo de consulta permite o agrupamento de sessões similares, reduzindo assim o espaço consumido pelo banco de dados. Além disso novas requisições feitas ao servidor, que solicitem dados já previamente consultados anteriormente e armazenados em sessões de usuários, podem fazer uso destas sessões armazenadas de forma a reduzir o número de chamadas a servidores e/ou serviços externos ao back-end.

4 Otimizações e estratégias

Objetivos: ***(arquitetura)***, ***(armazenamento)***, ***(otimização)***

Um ponto crucial para o bom funcionamento do sistema refere-se a como a integração de outros sistemas é feita.

Neste sentido é preciso se conciliar dois objetivos que são potencialmente contraditórios. Por um lado é preciso manter a plataforma INACITY o mais agnóstica o possível com relação à

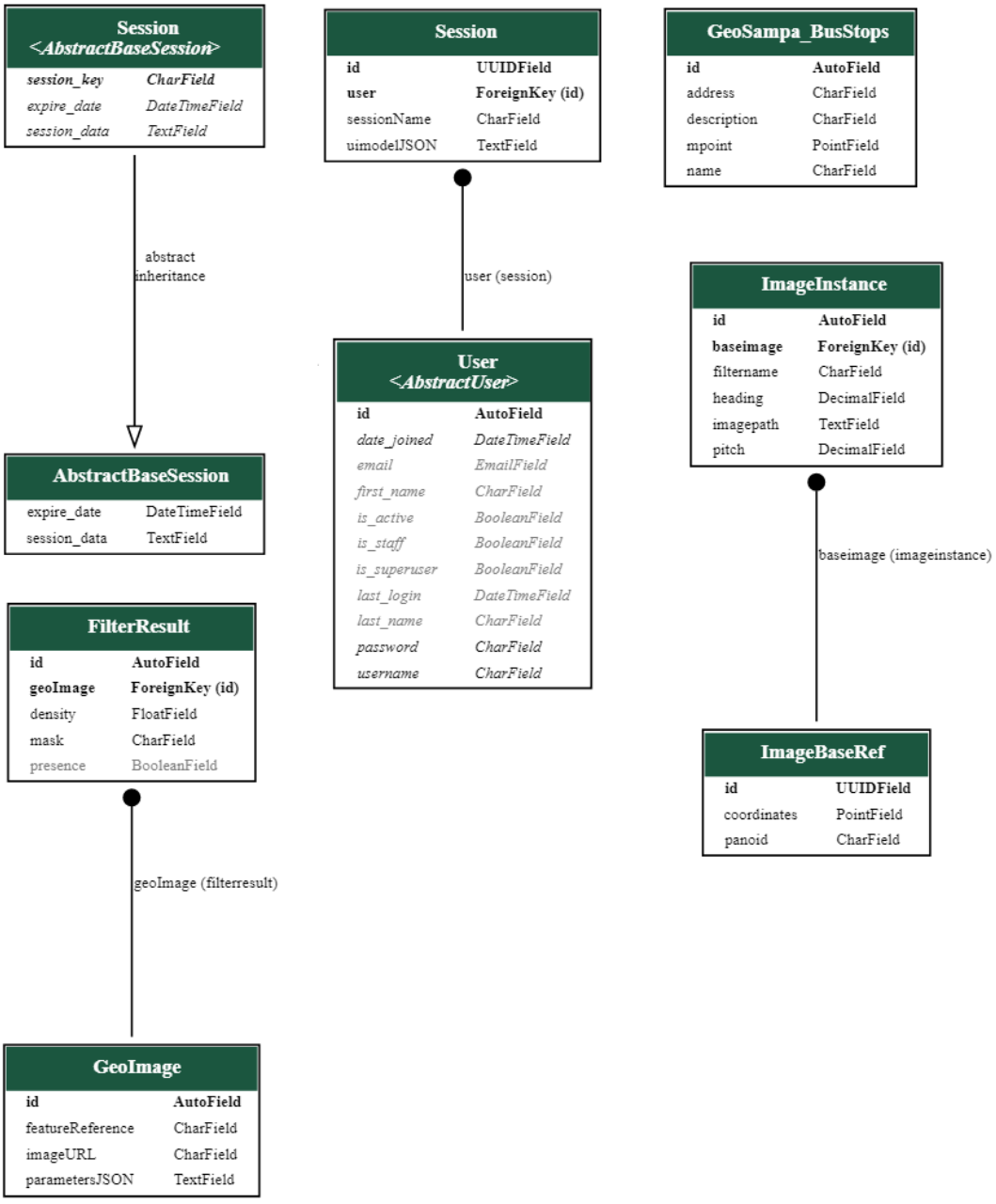

Figura 2: Diagrama de Entidades e Relacionamentos do banco de dados do back-end.

quais sistemas ela está integrada, isso é, a integração de um novo SIG, Provedor de Imagens ou Filtro não deve interferir com o funcionamento dos demais sistemas já integrados. Por outro lado é desejável que consultas à sistemas externos sejam otimizados a fim de se reduzir o tempo de cada consulta ou até mesmo o custo no caso do sistema externo ter associado a cada consulta um custo.

Para atingir o primeiro objetivo, um sistema agnóstico, cada novo sistema a ser integrado deve possuir um componente correspondente que serve como uma interface entre o INACITY e esta plataforma externa. Este componente será, como explicado antes na subsessões 3.2.1, 3.2.2 e 3.2.3, uma subclasse de algum componente abstrato da plataforma INACITY e portanto deverá implementar algumas propriedades e comportamentos que servem para manter uma homogeneidade no que tange a interação entre os diversos componentes da plataforma INACITY. Com relação ao segundo objetivo cada componente derivado de uma classe abstrata do INACITY deve ser modelado seguindo as diretrizes de sua superclasse, porém sua implementação deve seguir as melhores práticas definidas pelo sistema externo que este componente é responsável por integrar.

4.1 OpenStreetMap

Objetivos: ***(arquitetura)*****,** ***(otimização)***

O SIG OpenStreetMap (OSM) possui sua própria representação do que constitui sua base de dados. Um dos objetivos da plataforma INACITY é permitir de forma transparente a integração de um novo SIG, isso é, a integração de um novo SIG deve ser feita de forma que informações oriundas dele possam ser combinadas com os demais componentes da plataforma sem que estes precisem ser modificados.

Tendo este objetivo em vista e o fato de que o OSM não segue um modelo padronizado para que consultas ao OSM sejam otimizadas elas são feitas usando-se a linguagem 'Overpass Query Language' (OQL)[Wik19]. As consultas criadas com a linguagem OQL são feitas a API chamada Overpass Turbo [Wik18], esta API permite acesso ao banco de dados público que dá acesso ao GIS mantido pelo OSM. Os resultados da consulta ao Overpass Turbo seguem a especificação definida pelo OpenStreetMap, e como tal especificação não é padronizada é necessário que este resultado seja convertido pelo back-end antes de ser devolvido ao front-end.

4.1.1 Simplificação de características geográficas

Objetivos: ***(arquitetura)*, *(otimização)***

Uma particularidade da representação de objetos geográficos no OSM diz respeito a fragmentação de uma mesma entidade geográfica (por exemplo uma avenida). Uma das razões desta fragmentação é por causa da construção da base de dados se dar a partir de contribuições de usuários, que faz com que uma mesma entidade seja representada por um ou mais conjuntos distintos de características geográficas. Cada conjunto distinto de características geográficas tipicamente pertence à uma entidade diferente, porém em alguns casos uma mesma entidade pode acabar sendo representada por conjuntos distintos de características geográficas. Foi observado no caso de alguns logadouros (por exemplo ruas e avenidas) que quando estes eram compostos por vias de mão única e sentidos opostos (dois trechos separados que convergem ou divergem de um mesmo ponto central) ou alguns logadouros que eram formados por dois trechos intercalados por um segundo logadouro (trechos separados) cada trecho do logadouro pode ser representado por um conjunto de características geográficas distintas, sendo a única forma de identificar que ambos os conjuntos compunham o mesmo logadouro é através do seu nome.] Para sanar este problema na plataforma INACITY tratamos entidades com o mesmo nome como sendo a mesma entidade, independentemente dos conjuntos de características geográficas que os compõem. Uma limitação clara desta abordagem é que uma vez que duas entidades de mesma natureza (por exemplo edifícios, logadouros ou pontos de ônibus) oriundas do mesmo SIG possuam a mesma identificação (por exemplo nome) ambas serão tratadas como sendo a mesma entidade. Notamos que este fenômeno ocorre em casos em que regiões selecionadas cobrem uma área muito grande, sejam estas regiões relativamente pequenas e muito afastadas ou regiões de interesse muito grandes. Uma possível abordagem seria usar um outro identificador de entidade, como um identificador único provido pelo SIG ou algo estritamente relacionado ao tipo de entidade como CEP para logadouros. Contudo notamos que este tipo de decisão não pode ser aplicado em todos os casos, tendo-se em vista que identificadores próprios de SIGs podem acarretar no problema de entidades fragmentadas e portanto com identificadores distintos para uma mesma entidade, ou no caso de identificadores relacionados à um tipo de entidade não há garantias de que tal identificador seja único ou sequer exista para qualquer entidade em qualquer região no mundo.

Além da fragmentação de entidades em conjuntos distintos de características geográficas também notou-se uma outra particularidade do SIG OSM no que tange a representação de um logadouro qualquer, incluindo os citados no parágrafo anterior. No OSM os logarouros são representados

como sendo conjuntos de segmentos de retas e estes são representados como uma lista ordenada de nós. Sendo assim, um logadouro é essencialmente uma lista de listas de nós, onde a lista do primeiro conjunto de listas representa um trecho de um logadouro e um trecho é uma lista de nós. Essa representação tende, na maior parte dos casos, a repetir nós que existem na interseção de trechos. Na plataforma INACITY usamos os nós de um trecho como referências para a coleta de imagens, ou seja, é solicitado à um provedor de imagens uma imagem cuja localização é indicada por um nó de um determinado trecho. Uma vez que cada nó em uma mesma entidade é tido como sendo único sem um tratamento adequado os nós que existem nas interseções de trechos acabam sendo consultados múltiplas vezes, oque acarreta na mesma imagem sendo coletada e apresentada múltiplas vezes. Para contornar este problema, a abordagem adotada consiste na fusão de trechos de uma mesma entidade de forma que cada par de trechos que possuam uma interseção serão tratados como sendo um único trecho. Desta forma, nenhum par de nós distintos de uma mesma entidade possuirão as mesmas coordenadas. Note que esta abordagem não exclui a possibilidade dois nós distintos de duas entidades distintas possuam as mesmas coordenadas, contudo, notou-se que isto tipicamente ocorre em cruzamentos de logadouros e portanto, apesar da localização ser a mesma, as direções frontais em cada ponto serão diferentes e portanto as imagens coletadas também.

4.2 Filtros de imagens

Objetivos: ***(arquitetura)***, ***(armazenamento)***, ***(otimização)***

Notou-se que a operação mais custosa (como já era esperado) para o back-end é o processamento das imagens. Foram consideradas as diversas consequências das diferentes formas de se armazenar os resultados de uma imagem filtrada.

Ao armazenarmos uma imagem filtrada, e juntamente todos os resultados extraídos desta imagem, incorremos num custo de armazenamento potencialmente muito alto, tendo-se em vista que cada localização pode conter múltiplas imagens em momentos, ângulos e provedores de imagens diferentes. Um segundo ponto que deve ser considerado ao armazenar as imagens filtradas é o da licença de uso de um provedor de imagens que em alguns casos não permite que a imagem filtrada seja armazenada ou, caso permita, as vezes o faz somente por um limitado período de tempo. Tendo-se as duas considerações em vista atualmente cada imagem filtrada é armazenada tanto na sessão do usuário quanto num espaço reservado somente para imagens filtradas e indexado por localização. Desta forma podemos otimizar a recuperação de uma sessão de usuário, sem necessariamente estarmos mantendo as imagens filtradas acessíveis a todo usuário do sis-

tema, e similarmente para os casos possíveis, as imagens filtradas armazenadas num espaço indexado por localização podem ser reaproveitadas em outras consultas de demais usuários. Nesta configuração foi usado espaço de armazenamento em troca de performance.

4.3 Fragmentação das consultas

Objetivos: ***(segmentação)***, ***(otimização)***, ***(resiliência)***

Dependendo da quantidade de características geográficas dentro das regiões de interesse do usuário, a consulta ao servidor pode ser ultrapassar os limites permitidos pelos SIGs e/ou provedores de imagens. Além disso uma requisição muito grande pode tomar mais recursos do que o servidor dispõem para poder processá-la inteiramente e devolvê-la inteiramente ao front-end. Para contornar este tipo de problema, cada requisição feita ao back-end leva em conta somente uma característica geográfica de cada vez, ou seja, dado um conjunto de regiões de interesse, cada um com um conjunto de características geográficas, primeiro o front-end irá listar quais são as características geográficas presentes em cada região de interesse e então formular uma requisição para cada característica geográfica. Note que esta abordagem elimina requisições duplicadas para uma mesma característica geográfica que exista em duas regiões sobrepostas. Além disso uma vez que uma requisição é completada para uma determinada característica geográfica a informação vinculada a requisição fica vinculada à característica geográfica de forma que caso o mesmo tipo de requisição seja feita novamente, não será gerada uma requisição para uma característica geográfica que já contenha a informação desta requisição vinculada a si.

Como cada requisição é formulada individualmente, o retorno de cada requisição pode ser vinculado à característica urbana usada para formular a requisição e desta forma, ao se realizar a mesma requisição novamente, somente as características geográficas cujas requisições não foram completadas serão feitas novamente.

5 Adaptações

Ao longo do desenvolvimento do projeto alguns aspectos fundamentais da implementação e da implantação do projeto mudaram. Estas mudanças acarretaram em adaptações necessárias para que o projeto pudesse ser concluído. Para que estas adaptações não interferissem com o funcionamento dos demais componentes já criados, criamos novos componentes para atender as mudanças que ocorreram ao longo do projeto, de forma que estas mudanças fossem transparentes para os componentes que não estivessem diretamente ligados a elas.

As duas principais mudanças que ocorreram ao longo do projeto foram mudanças relacionadas a implantação do projeto, isto é, onde o projeto deveria ser hospedado e uma segunda mudança foi ocasionada por uma mudança repentina na API do Google Street View, principal Provedor de Imagens na versão atual da plataforma INACITY.

5.1 Migração do Azure

O projeto INACITY foi inicialmente patrocinado pela Microsoft através do programa "Microsoft Azure Sponsorship"onde 20 mil dólares em créditos foram disponibilizados para serem usados em serviços da plataforma Azure [Mic]. Após a expiração dos créditos, a plataforma INACITY foi migrada para um novo servidor.

A instalação e configuração de um novo servidor para hospedar a plataforma INACITY motivou a decisão da forma primária de implantação da plataforma INACITY ser baseada na ferramenta Docker[Inc]. Essa ferramenta permite a criação de contêineres pré-configurados através de scripts. Com isto, todos os requisitos de bibliotecas, configurações de sistema e inclusive a escolha de um Sistema Operacional podem ser embutidos num script que irá criar um contêiner e este será gerenciado pelo Docker.

Além da implantação da plataforma INACITY ser feita através da ferramenta Docker, tendo-se em vista a necessidade de um banco de dados e a existência de um contêiner para Docker do Sistema Gerenciador de Banco de Dados PostgreSQL, foi adotada a imagem oficial para Docker de PostgreSQL[Groa].

5.2 Mudança da licença e da arquitetura do Google Street View

A API Google Street View (GSV) foi projetada para ser usada como um componente em uma página Web, isso é, um componente de front-end. Contudo, a plataforma INACITY foi criada tendo-se em vista a premissa de que a integração de fontes externas de dados (como Provedores de Imagem) deveria ser realizada no back-end. Para se poder utilizar o GSV no back-end foi configurada uma instância de Node.js[Fouc]. Node.js é um ambiente de execução de código javascript criado para que este tipo de código (tipicamente usado em front-ends) possa ser executado independentemente de um navegador Web. Através deste ambiente foi possível se executar as bibliotecas de javascript do GSV no back-end, criando-se um canal de comunicação entre o ambiente Node.js e o servidor Django da plataforma INACITY.

Em outubro de 2018 a biblioteca do GSV foi atualizada de forma que sua execução no ambiente Node.js se tornou inviável. Sendo assim foi necessário que a integração com o Provedor de

Imagens Google Street View fosse feita no front-end. Esta integração é realizada através do componente 'GSVService' do front-end. Este componente recebe as requisições de imagens da mesma forma como o back-end recebe as demais requisições. Uma vez que as imagens do Google Street View são coletadas estas são então devolvidas ao componente 'UIModel' do front-end, o qual se encarrega de repassar as imagens ao componente que as solicitou. Note que, para que seja mantida a consistência da sessão do usuário, os demais comportamentos relativos ao armazenamento de sessão à cada requisição executada continuam sendo executados, portanto, mesmo esta integração no front-end com fontes externas tem um comportamento consistente com os demais componentes para integração com fontes externas localizados no back-end.

6 Conclusão

Concluimos este relatório agradecendo à FAPESP o fomento desta bolsa TT.

Referências

[Dev] NumPy Developers. Numpy home page.

[dSP] Prefeitura de São Paulo. Sistema de consulta do mapa digital da cidade de são paulo — geosampa.

[Foua] Django Software Foundation. About the django software foundation.

[Foub] Python Software Foundation. Python 3 documentation.

[Fouc] © Node.js Foundation. Node.js.

[Groa] The PostgreSQL Global Development Group. postgres - docker hub.

[Grob] The PostgreSQL Global Development Group. Postrgresql: About.

[Inc] © 2019 Docker Inc. Get started with docker — docker.

[Int] InterSCity. Grupo interscity no gitlab.

[LLC] Google LLC. Discover street view and contribute your own imagery to google maps.

[McK] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[Mic] © 2016 Microsoft. Microsoft azure: Cloud computing platform & services.

[OJ18] Artur André Almeida de Macedo Oliveira e Roberto Hirata Jr. Investigate and analyse a city-inacity. Dissertação de Mestrado, Universidade de São Paulo, 2018.

[Oli] Artur André Almeida de Macedo Oliveira. Repositório inacity no gitlab.

[Ope17] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[vdWSN+14] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu e the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[Wik18] OpenStreetMap Wiki. Overpass api/installation — openstreetmap wiki,, 2018. [Online; accessed 29-August-2019].

[Wik19] OpenStreetMap Wiki. Overpass api/overpass ql — openstreetmap wiki,, 2019. [Online; accessed 29-August-2019].

[YZMG09] Jun Yang, Linsen Zhao, Joe Mcbride e Peng Gong. Can you see green? assessing the visibility of urban forests in cities. *Landscape and Urban Planning*, 91(2):97–104, 2009.