LUCAS BOTELHO COELHO
MÁRCIO CARLOS PERIN TEDESCO
TIAGO LEE

ATAIRU – REDE SOCIAL PARA VIAJANTES

São Paulo
2015

LUCAS BOTELHO COELHO
MÁRCIO CARLOS PERIN TEDESCO
TIAGO LEE

ATAIRU – REDE SOCIAL PARA VIAJANTES

Trabalho de conclusão de curso de Engenharia de Computação apresentado à Escola Politécnica da Universidade de São Paulo – Departamento de Engenharia de Computação e Sistemas Digitais

Orientador: Prof. Dr. Selma Shin Shimizu Melnikoff

São Paulo
2015

DEDICATÓRIA

Agradeço à minha família, pelo apoio incondicional nessa trajetória. Mãe, seu cuidado e dedicação foram o que me deram, em alguns momentos, a esperança para seguir. Pai, seu incentivo ao estudo me fez chegar até aqui. Irmão, obrigado pela parceria. Agradeço também aos meus amigos e professores.

Lucas Botelho Coelho

Dedico este trabalho à minha companheira Sophie Fluzin, com muito carinho, pelo suporte, companheirismo, motivação e apoio indiscutível durante todas as etapas desse projeto e momentos difíceis da minha graduação. Ao meu irmão pela parceria e colaboração no *design* deste projeto. Aos meus pais, pelo apoio incondicional durante todas as etapas da minha vida, por sempre escolherem acreditar em mim e investirem cegamente nos meus sonhos e ambições.

Márcio Carlos Perin Tedesco

Agradeço a Deus por permitir todas as experiências que tive até o presente momento e que ainda terei na minha vida. Dedico este trabalho aos meus pais e ao meu irmão, que me apoiaram nesta jornada de estudos e de trabalho, além do amor que recebi deles. Aos amigos que acreditaram em mim e que permitiram a minha participação neste projeto.

Tiago Lee

AGRADECIMENTOS

Agradecemos à nossa professora orientadora, Profa. Dra. Selma Shin Shimizu Melnikoff, por aceitar orientar o nosso projeto e por nos orientar durante todos os momentos em que buscamos seu auxilio. Obrigado, pelos conselhos e pela orientação.

Agradecemos ao Marcos Heitor Perin Tedesco, pela grande ajuda no *design* do nosso sistema. Obrigado pela dedicação e perfeccionismo de todos os seus trabalhos.

Agradecemos também aos nossos amigos do coop14 que nos acolheram como se sempre fossemos da classe e nos ajudaram durante todo este ano a tornarem as aulas e estudos mais agradáveis. Sem eles não seria tão motivador estudar em grupo para as provas. Com eles aprendemos que nem sempre é possível fazer tudo com perfeição mas sempre é possível atingir o alvo com força e tranquilidade.

RESUMO

A possibilidade e a facilidade de viajar por meio dos transportes aéreos permitem que um número maior de turistas percorram o mundo. Porém, na hora de procurar atrações locais, restaurantes típicos, museus e outras informações, é necessário gastar um certo tempo com pesquisas na Internet, Blogs, aplicativos de viagens como o TripAdvisor, agência de viagens, para receber recomendações e saber quais os melhores atrativos da região. Em todos os casos, se a pessoa vai a algum lugar pela primeira vez não há como evitar este gasto de tempo em pesquisa. Denominado de ATAIRU, palavra de derivação indígena, tupi-guarani, que significa companheiro de viagem, este projeto de formatura tem como foco principal desenvolver um sistema que facilite a busca de informações de atrações da cidade de destino e possibilitar a interação entre os usuários por meio de uma rede social.

Palavras chaves: Sugestão de roteiros. Filtragem colaborativa. Sistema de recomendação.

ABSTRACT

The possibility and the easiness to travel through the air transport system allow to a higher number of tourists to discover the world. However, to look for attractions, venues, typical restaurants, museums and other information, it is necessary to spend some time to search in the Internet, Blogs, travelling apps like TripAdvisor, travel agencies, to receive recommendations and to get to know the best attractions of the area. Denominated as ATAIRU, word with origins in the indian language tupi-guarani which means travel companion, this graduation project has the main focus in developing a system that makes it easy to find information about attractions of the destined city and make it possible to interact between the users of the system through its social network.

Keywords: Itinerary Suggestions. Collaborative Filter. Recommendation System.

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

Este capítulo apresenta a motivação, o objetivo, a justificativa e a estrutura da monografia.

1.1. Motivação

Este trabalho insere-se no contexto de viagens, no qual pessoas desejam viajar e conhecer lugares mas, muito além disso, querem conhecer pessoas que tenham o mesmo objetivo que elas ou compartilhem de interesses semelhantes. Quando as viagens não são feitas através de agências de turismo, é difícil saber onde se hospedar, visitar, ter uma boa refeição e conhecer a cultura local. Todas essas questões são consideradas quando o assunto tratado é viagem, no entanto, nem sempre são feitas de maneira eficiente por falta de conhecimento e informação. Dentro desse contexto, a motivação para o projeto de formatura é oferecer um sistema, cujo principal serviço é oferecer, às pessoas, melhores experiências e facilidades ao visitarem novos lugares.

1.2. Objetivo

O objetivo do trabalho é desenvolver um sistema, denominado de ATAIRU (palavra de derivação indígena, tupi-guarani, que significa companheiro de viagem), com ênfase em uma rede social voltada para viajantes e mochileiros. A plataforma de viagens integra os seguintes recursos:

- Informações atualizadas de restaurantes, museus, hotéis, *hostels* e atrações turísticas com seus respectivos horários de funcionamento;
- Capacidade de traçar rotas de viagem e verificar se há alguém que irá percorrer uma parte da mesma rota ou ainda um mesmo roteiro dentro da cidade;
- Plataforma social com a capacidade de interação entre usuários do aplicativo;
- Possibilidade de elaboração de um roteiro e compartilhá-lo com outros usuários;
- *Gamification*, para permitir que a visita de novos lugares e a busca de novas pessoas através do aplicativo seja um jogo, motivando e incentivando a prática do turismo e desenvolvimento cultural da pessoa.

1.3. Justificativa

A equipe de projeto é formada por três alunos que realizaram intercâmbio em três continentes diferentes: Lucas no Canadá, continente americano, Márcio na Espanha, continente europeu e Tiago na Coréia, continente asiático. Isto os levou a conhecer muitas pessoas, muitos lugares e, nas discussões para a definição do trabalho, puderam identificar problemas que ocorreram durante as viagens e maneiras de melhorar essa experiência que tem, como intuito, alimentar as pessoas culturalmente, promovendo o seu crescimento pessoal.

Durante as viagens, foram realizados estudos das ferramentas de viagens disponíveis e chegou-se à conclusão de que não há nenhuma, entre as analisadas, que atendesse às expectativas dos participantes, no mercado. O sistema de referência da TripAdvisor é o mais próximo no que se refere à disponibilização de informações de lugares, porém os usuários precisam requisitar a busca de lugares apenas, sem ter uma ferramenta para montar um roteiro de viagens (TRIPADVISOR, 2014). Outro sistema de referência é o Gogobot, que apresenta uma ferramenta para auxiliar na construção de um roteiro, sendo que o próprio usuário precisa fazer a pesquisa de lugares pelo sistema para adicionar à sua lista de lugares a visitar, além de não oferecer uma rede social a seus usuários (GOGOBOT, 2014).A falta de um sistema, que tenha todas as informações relevantes integradas, ofereça sugestões de roteiros e que apresente uma rede social, motivou a propor este projeto de formatura.

1.4. Estrutura da Monografia

O documento é organizado em 5 capítulos: Introdução, Especificação do Sistema, Conceitos e Tecnologias, Desenvolvimento e Considerações Finais.

No capítulo 1 - Introdução – são apresentadas as seções de Motivação, Objetivo, Justificativa e Organização do trabalho.

No capítulo 2 - Especificação do Sistema – são apresentados os resultados da análise de requisitos e do sistema Atairu: Funcionalidades, Modelo de Casos de Uso, Diagrama de Navegação, Interface Homem-Computador, Diagrama de Classes Simplificado e Considerações Finais do Capítulo.

No capítulo 3 - Conceitos e Tecnologias - são apresentadas as explicações e as justificativas sobre as decisões referentes às tecnologias adotadas, abrangendo: SOA, *Restful* APIs, iOS, *Ruby on Rails*, Plataforma Heroku, SGBD PostgreSQL, Provas de Conceito e Considerações Finais do Capítulo.

No capitulo 4 - Desenvolvimento – são apresentados: Divisão de Atividades, Arquitetura de Sistema, Modelagem de Dados, Algoritmo de Geração de Roteiros, *Backlog* de Implementação do *Backend*, Resultados de Implantação do *Backend* e Considerações Finais do Capítulo.

No capítulo 5 – Considerações Finais – são apresentadas as seções Conclusão, Contribuições e Trabalhos Futuros.

2. ESPECIFICAÇÃO DO SISTEMA

Este capítulo apresenta os requisitos do sistema, através de Funcionalidades, Modelo de Casos de Uso, Interface Homem Computador, Diagrama de Navegação e Diagrama de Classes Simplificado.

2.1. Funcionalidade

O sistema Atairu possui sete funcionalidades, que são: criação de rotas e roteiros, sugestão de roteiro, composição de roteiros, *matching* de roteiros, *chat*, *gamification* e aviso sobre eventos e pontos importantes. As cinco iniciais possuem maior prioridade, pois constituem um conjunto consistente para mostrar os recursos do aplicativo e, portanto, foram consideradas para o projeto de formatura. As duas últimas (6 e 7) não foram consideradas para essa etapa, porém são interessantes para complementar o sistema.

2.1.1. Funcionalidade 1 - Criação de Rotas e Roteiros

Descrição: Tem o objetivo de fornecer recursos para criar uma rota dentro de uma viagem (cidades a serem visitadas), incluindo datas, tempo de duração, interesses, e para permitir adicionar um usuário ou grupo de usuários no roteiro (eventos a serem percorridos em uma cidade) ou à rota.

A rota é constituída das seguintes informações:

- Cidades a serem visitadas em ordem cronológica
- Datas de início e fim de viagem
- Usuários que participarão da viagem
- Um ou mais roteiros para cada cidade

O roteiro é constituído das seguintes informações:

- Agenda dos eventos a serem realizados
- Datas de estadia
- Horário dos eventos
- Descrição dos eventos

Cada evento pode listar os usuários (de maneira discreta, por exemplo, uma

pequena foto do usuário) que possuem este evento registrado no seu roteiro. O perfil do usuário deve conter as seguintes informações:

- Foto
- Nome do usuário
- Cidades já visitadas
- Interesses da pessoa: utilizados para obter um roteiro orientado aos gostos do usuário

Motivação: Tem a finalidade de facilitar a organização da viagem do usuário.

Recursos:

- O usuário pode criar o seu próprio roteiro para uma determinada cidade.
- O usuário pode editar um roteiro sugerido pela aplicação.
- O usuário pode compartilhar diferentes roteiros com diferentes usuários.
- As informações do usuário podem ser obtidas através de uma interface com o Facebook.

2.1.2. Funcionalidade 2 - Sugestão de Roteiro

Descrição: Tem o objetivo de fornecer um mecanismo para apresentar sugestões de roteiro para uma determinada cidade ou lugar. O sistema fornece um roteiro, de acordo com a duração da viagem, o período do ano e o perfil do usuário. A partir dos roteiros de vários usuários e de roteiros previamente elaborados, o algoritmo faz sugestões de acordo com certos filtros, levando em conta os interesses do usuário.

Motivação: Quando se tem poucos dias para visitar uma cidade, por exemplo, é difícil otimizar o roteiro com lugares que são mais significativos.

Filtros:

- Tempo da viagem
- Período do ano
- Interesses/Perfil do usuário
- Outros a definir

2.1.3. Funcionalidade 3 - Composição de Roteiros

Descrição: Tem o objetivo de fornecer um mecanismo para compor (ou misturar) os roteiros personalizados de diversos usuários, para criar um roteiro único para os usuários em questão.

A partir de vários roteiros de vários usuários que compartilham o mesmo roteiro, esta funcionalidade gera um roteiro único de forma a se adaptar aos interesses de cada usuário.

Motivação: pessoas que viajam juntas e que pretendem desfrutar o tempo juntas na cidade podem, através dessa funcionalidade, elaborar um roteiro unificado.

2.1.4. Funcionalidade 4 - *Matching* de roteiros:

Descrição: Tem o objetivo de fornecer um mecanismo para realizar *matching,* em que um usuário pode encontrar e entrar em contato com um ou mais usuários que realizam a mesma rota que ele.

Motivação: pessoas que viajam sozinhas e que pretendem desfrutar o tempo na cidade com alguém podem, através dessa funcionalidade, encontrar outras pessoas com o mesmo perfil e, dessa maneira, aumentar o intercâmbio cultural e tornar a viagem mais dinâmica.

2.1.5. Funcionalidade 5 - *Chat*

Descrição: Tem o objetivo de fornecer um mecanismo para um *chat,* para facilitar a comunicação entre usuários que compartilham de uma mesma rota ou com amigos da sua lista de *chat.*

Motivação: Tem a finalidade de promover um primeiro contato entre os usuários que são pareados via *Matching* ou para facilitar a comunicação entre os usuários que já pertencem à lista de amigos do usuário.

2.1.6. Funcionalidade 6 (Extensão) - *Gamification*

Descrição: Tem o objetivo de fornecer uma plataforma mais interativa entre os usuários por meio de *gamification.*

Motivação: Tem a finalidade de atrair e incentivar os usuários a utilizarem o

aplicativo, o que pode levá-los a fazer comentários, sugestões e *feedbacks* de viagens.

2.1.7. Funcionalidade 7 (Extensão) - Aviso sobre Eventos e Pontos Importantes

Descrição: Tem o objetivo de fornecer um mecanismo que notifica, ao usuário, qual é o evento histórico associado ao lugar por onde está passando; além disso, pela posição adquirida pelo GPS, o aplicativo notifica pontos importantes a uma determinada distância.

Motivação: Tem a finalidade de mostrar possibilidades de se visitar outros lugares além dos já programados pelo roteiro.

2.2. Modelo de Casos de Uso

Os atores possuem diferentes permissões de acesso às funcionalidades do sistema. São eles: usuário deslogado, usuário logado, sistema externo e administrador de sistema.

2.2.1. Usuário Deslogado

Corresponde ao ator que não está autenticado no sistema. Usufrui da funcionalidade 2, Sugestão de Roteiro, porém sem a possibilidade de salvar o roteiro em um perfil até que esteja autenticado no sistema.

2.2.2. Usuário Logado

Corresponde ao ator que está autenticado no sistema e, por isso, possui acesso a todas as funcionalidades do sistema.

2.2.3. Sistemas Externos

Estes atores são APIs do Facebook, Whatsapp, Google Maps e *sites* sobre informações turísticas. O sistema utiliza, a partir destes sistemas externos, as informações de perfis, como o do Facebook, de mapas, como o Google Maps, e de interfaces com *messengers,* como o Whatsapp (considerado um dos mais usados no

mundo), para possibilitar a troca de mensagens.

2.2.4. Administrador do Sistema

O administrador do sistema corresponde ao ator que interage com o sistema de maneira administrativa, tendo como principal função adicionar informações ou fontes de informações sobre eventos e pontos turísticos.

2.2.5. Casos de Uso por Ator

O usuário deslogado possui os seguintes casos de uso:

- UC1. Fazer Login
- UC2. Cadastrar usuário
- UC3. Gerar sugestão de roteiros
- UC4. Visualizar tela de Login
- UC6. Adicionar evento
- UC7. Remover evento
- UC10. Visualizar roteiro do dia no mapa

O usuário logado tem os seguintes casos de uso:

- UC1. Fazer Login
- UC3. Gerar sugestão de roteiro
- UC4. Visualizar tela de Login
- UC5. Visualizar tela de Perfil
- UC6. Adicionar evento
- UC7. Remover evento
- UC8. Visualizar informações do evento
- UC9. Visualizar roteiro do dia
- UC10. Visualizar roteiro do dia no mapa
- UC11. Visualizar tela de gerenciamento de rota
- UC12. Visualizar tela de contatos
- UC13. Visualizar tela de configurações do sistema

- UC14. Criar rota
- UC15. Remover rota
- UC16. Adicionar roteiro a partir de uma rota
- UC17. Adicionar roteiro a uma rota
- UC18. Remover roteiro
- UC19. Editar informações da rota
- UC20. Visualizar informações do roteiro (incluindo data e filtros)
- UC21. Adicionar contato à rota
- UC22. Executar *Matching*
- UC23. Adicionar um usuário à lista de amigos
- UC24. Enviar mensagem
- UC25. Adicionar amigos à rota
- UC26. Selecionar rota
- UC34. Remover contato da rota

O sistema externo tem os seguintes casos de uso:

- UC27. Obter perfil do usuário no Facebook
- UC28. Obter mapa do Google Maps
- UC29. Mostrar rota no mapa
- UC30. Executar *Web Crawler*

O administrador de sistema tem os seguintes casos de uso:

- UC31. Adicionar site à lista do *Web Crawler*
- UC32. Adicionar eventos no Banco de Dados
- UC33. Atualizar informações de eventos

Os casos de uso mais importantes, relacionados com as principais funcionalidades do sistema estão descritos com detalhes no APÊNDICE A – DESCRIÇÃO DOS CASOS DE USO.

2.3. Diagrama de navegação

O diagrama de navegação mostra a sequência das telas para a realização dos caos de uso do Aitaru. Foi dividido em partes para a melhor visualização, a partir das telas: Tela 0, Tela 2, Tela 6, Tela 7 e Tela 10. As Figuras 1 a 5 apresentam os componentes divididos do diagrama.

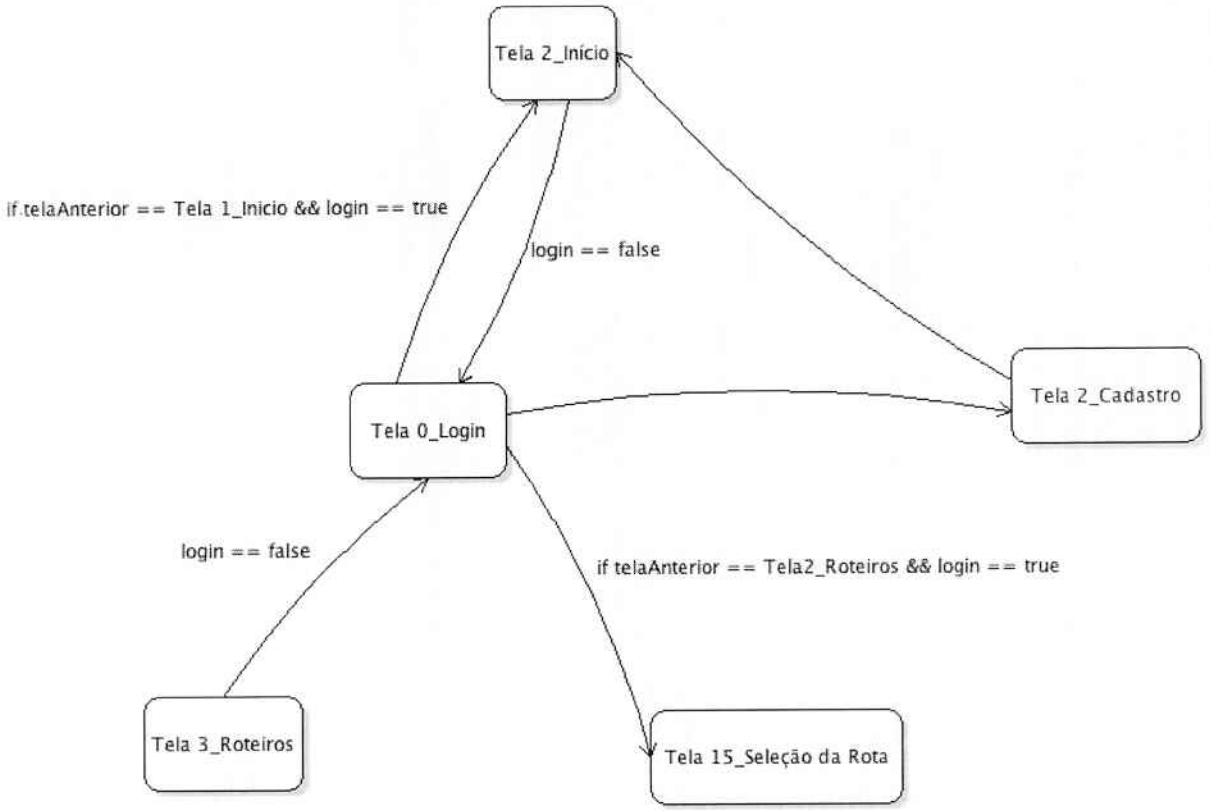

Figura 1 - Diagrama de Navegação - Parte 1

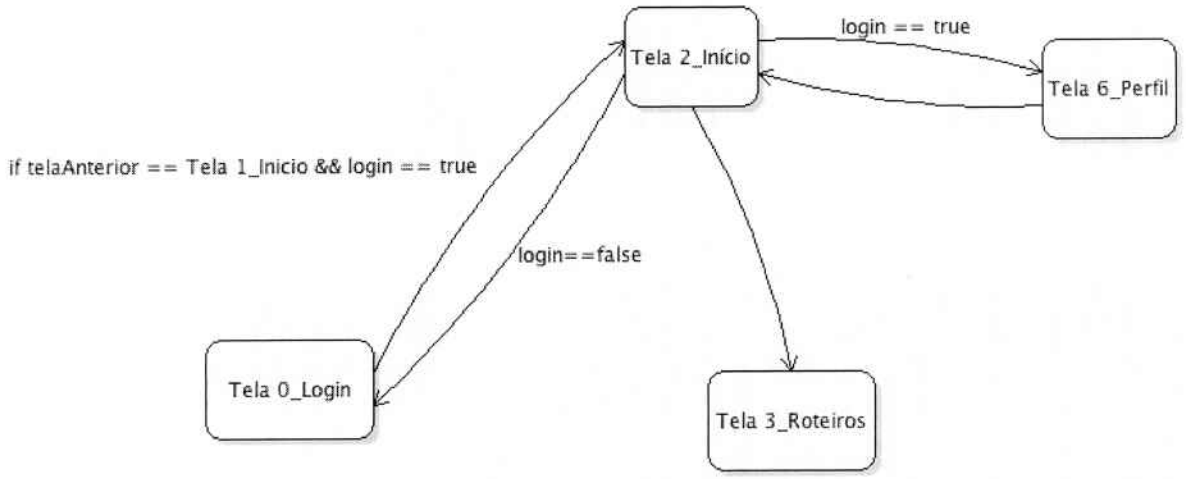

Figura 2– Diagrama de Navegação - Parte 2

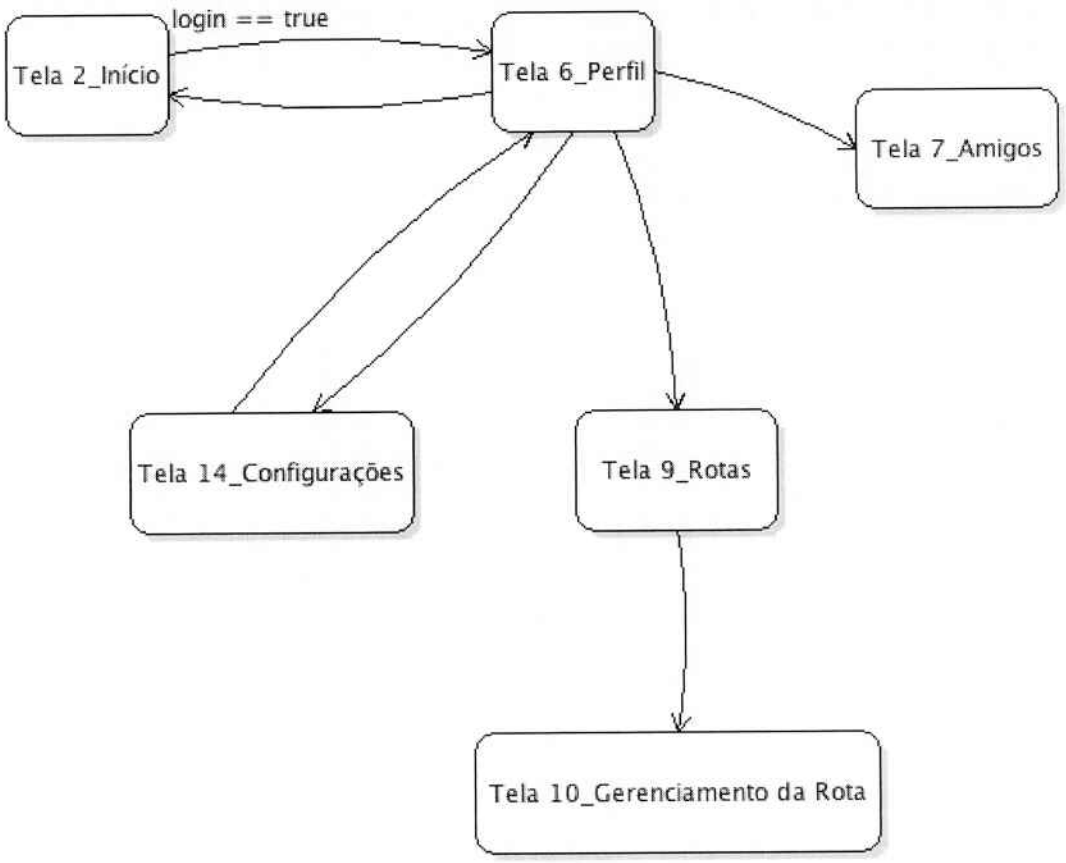

Figura 3 – Diagrama de Navegação - Parte 3

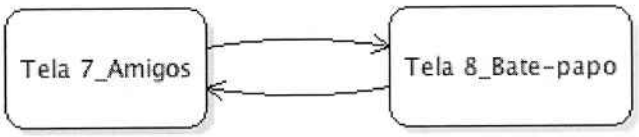

Figura 4 – Diagrama de Navegação - Parte 4

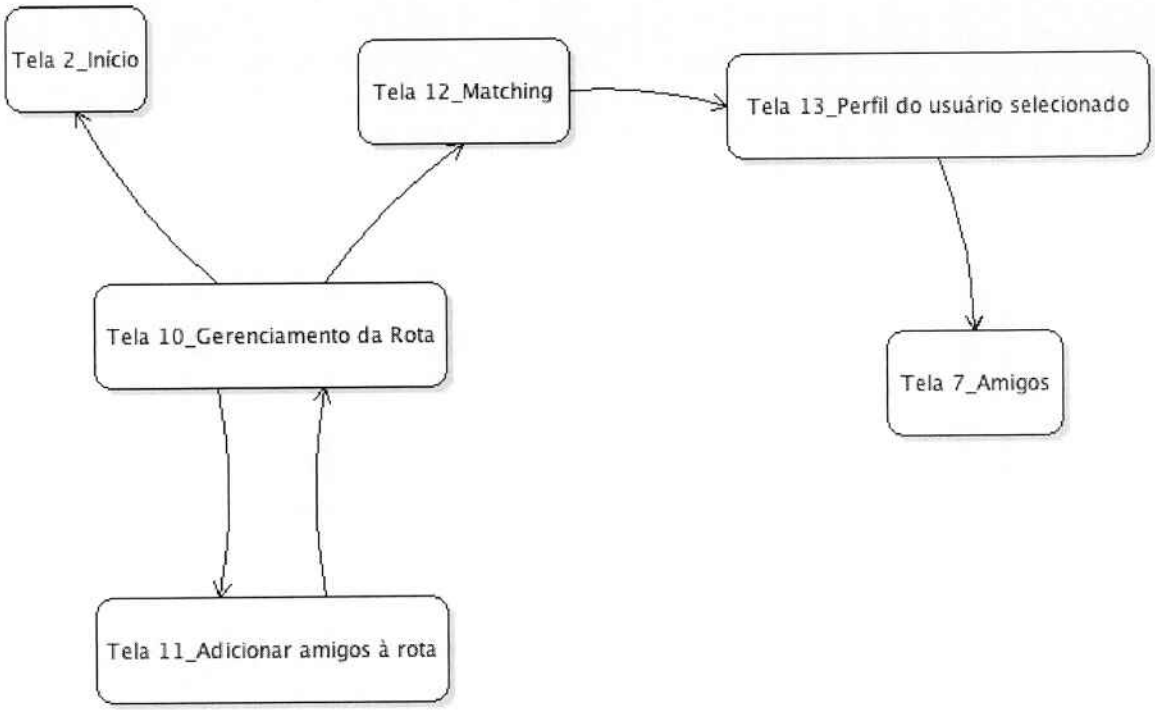

Figura 5 – Diagrama de Navegação - Parte 5

2.4. Interface Homem Computador

Nesta seção, são apresentados os protótipos das principais telas do sistema, em cada um dos possíveis estados e os casos de uso relacionados identificados. A lista completa dos protótipos de tela se encontra no APÊNDICE B – PROTÓTIPOS DE TELAS. As telas principais são: Tela 2, Tela 3, Tela 6, Tela 8, Tela 10, Tela 12 e Tela 14, e estão apresentadas nas Figuras 6 a 14.

Tela 2 - Início

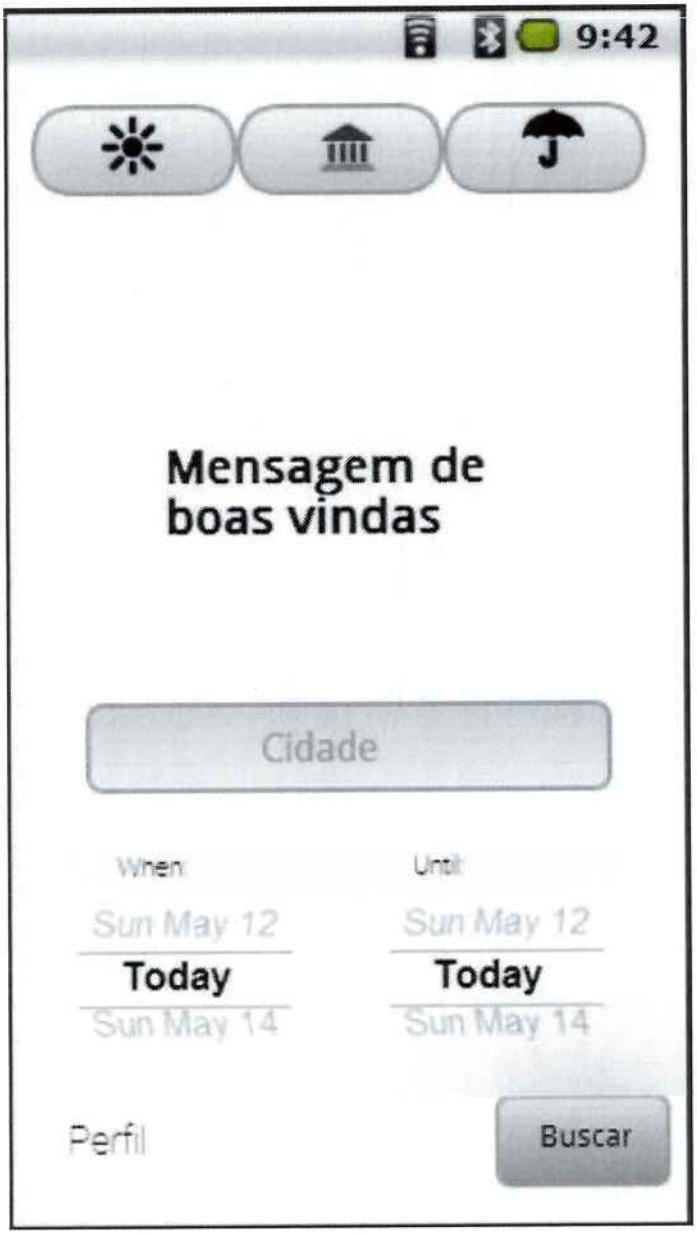

Figura 6 – Tela 2 - Início

Descrição: Essa é a tela inicial do aplicativo. Através dela, o usuário pode gerar um roteiro para uma determinada cidade, em um período estabelecido de dias e selecionando alguns filtros de interesse. O usuário, antes de gerar esse roteiro, pode acionar o *side bar* (menu que aparece ao deslizar a tela da esquerda para a direita) e ir para tela de realizar o *login* clicando no botão *Login*, ou pode ir para sua tela de perfil clicando no botão Perfil, caso ele já esteja logado.

Tela 3 - Roteiros

Figura 7 - Tela 3 - Roteiros

Descrição: Essa tela mostra o roteiro de uma determinada cidade com as informações dos respectivos lugares, ao lado da foto. O usuário pode, ao clicar no sinal de +, adicionar um novo evento ao roteiro; ao clicar no sinal X, pode excluir um evento do roteiro. O usuário pode visualizar os eventos do roteiro no mapa ao clicar no botão Mapa. Caso o usuário queira salvar esse roteiro, clica no botão Salvar Roteiro. Se este não estiver logado, o aplicativo irá para tela de *login*.

Tela 6 - Perfil

Figura 8 – Tela 6 - Perfil

Descrição: Nessa tela, o usuário visualiza o seu perfil com informações mais relevantes. Ao clicar em Viagens, o usuário vai para tela de gerenciamento de rotas. Ao clicar em Amigos, o usuário visualiza a lista de seus amigos e ao clicar em Configurações, o usuário vai para tela onde pode realizar as configurações do sistema (Tela 14).

Tela 8 - Bate-Papo

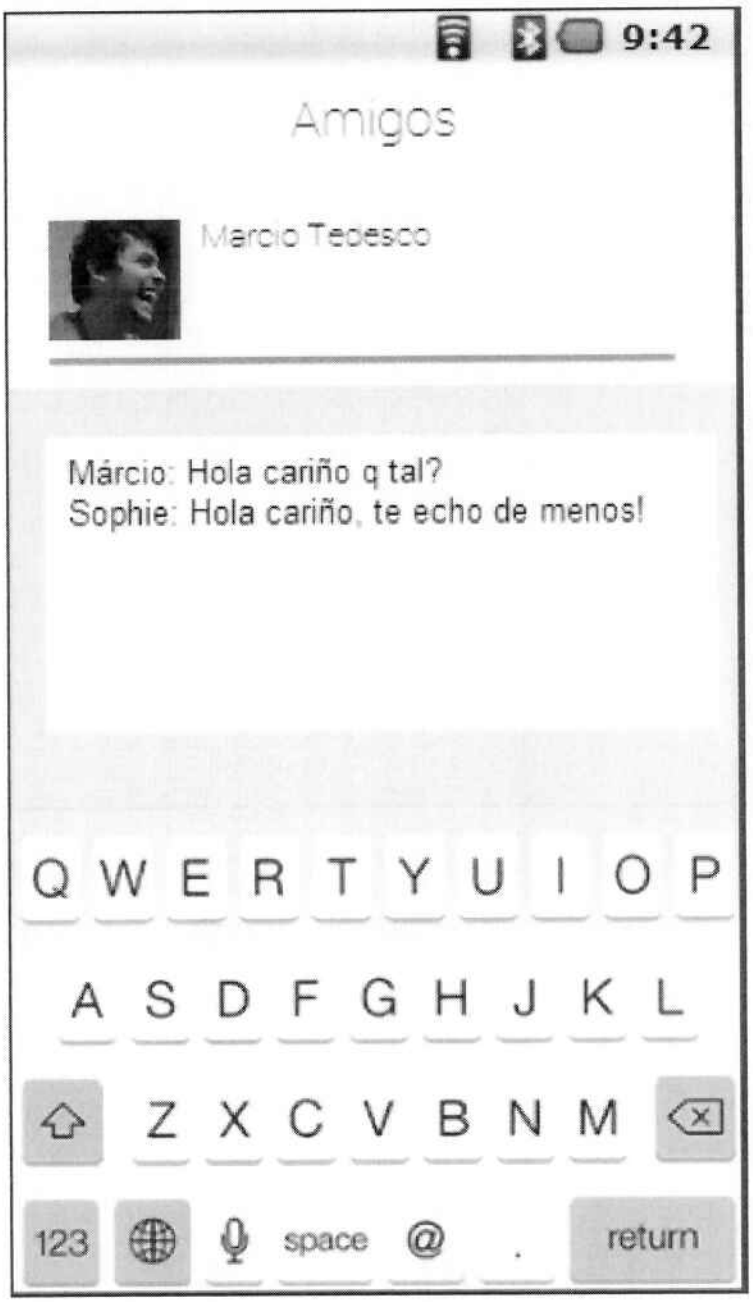

Figura 9– Tela 8 – Bate-papo

Descrição: Nessa tela, o usuário pode conversar com outro usuário.

Tela 10 - Gerenciamento da Rota

Figura 10– Tela 10 – Gerenciamento da Rota

Descrição: Nessa tela, o usuário visualiza a rota da viagem, ou seja, as cidades e a data para onde o usuário pretende viajar. Ao clicar no ícone , o sistema vai para tela de escolha de amigos que o usuário pode adicionar na viagem. Ao clicar no botão *Match*, o sistema procura, nas viagens de outros usuários, as intersecções entre as rotas e mostra na tela de *match*. Ao clicar no botão X, o usuário pode remover uma determinada cidade. Ao clicar no botão +, pode adicionar uma nova cidade ao roteiro.

Tela 12 - *Matching*

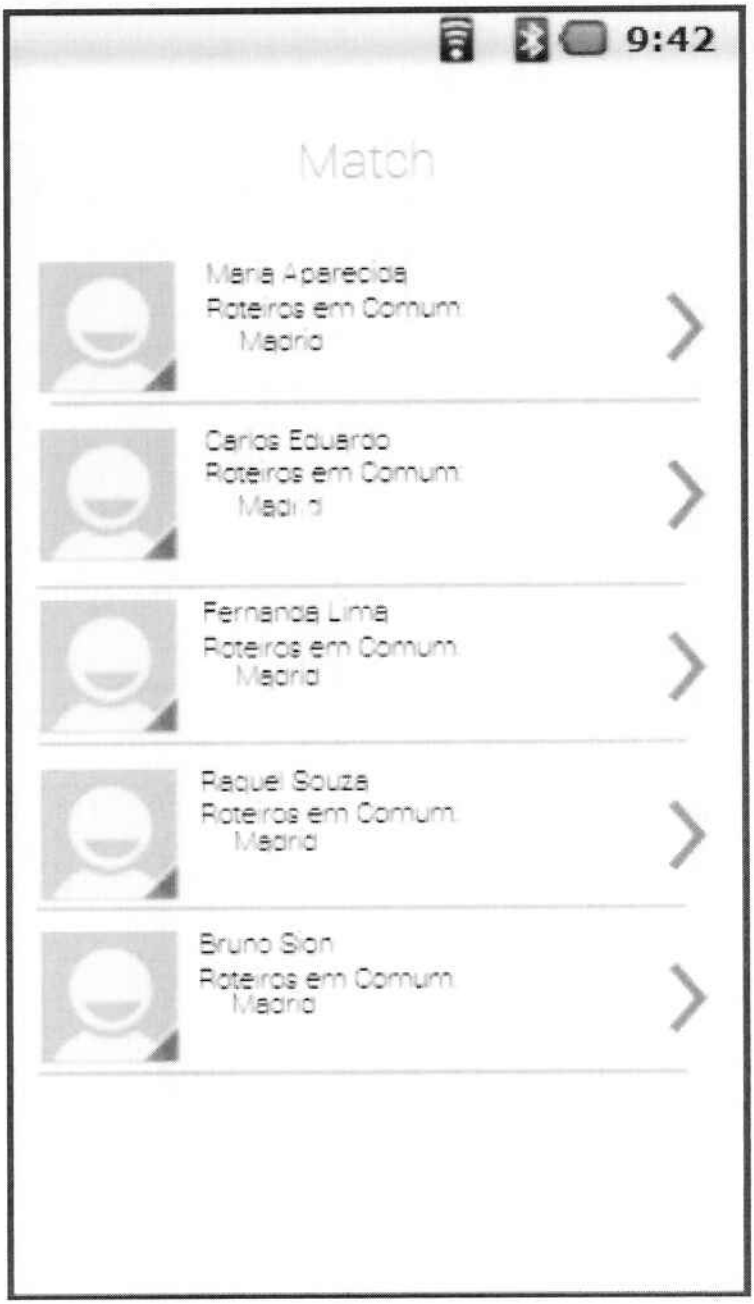

Figura 11 – Tela 12 – *Matching*

Descrição: A tela de *matching* lista os usuários que possuem, em sua rota, pelo menos um dia de roteiro coincidente, ordenados pela quantidade de roteiros em comum.

Tela 14 - Configurações

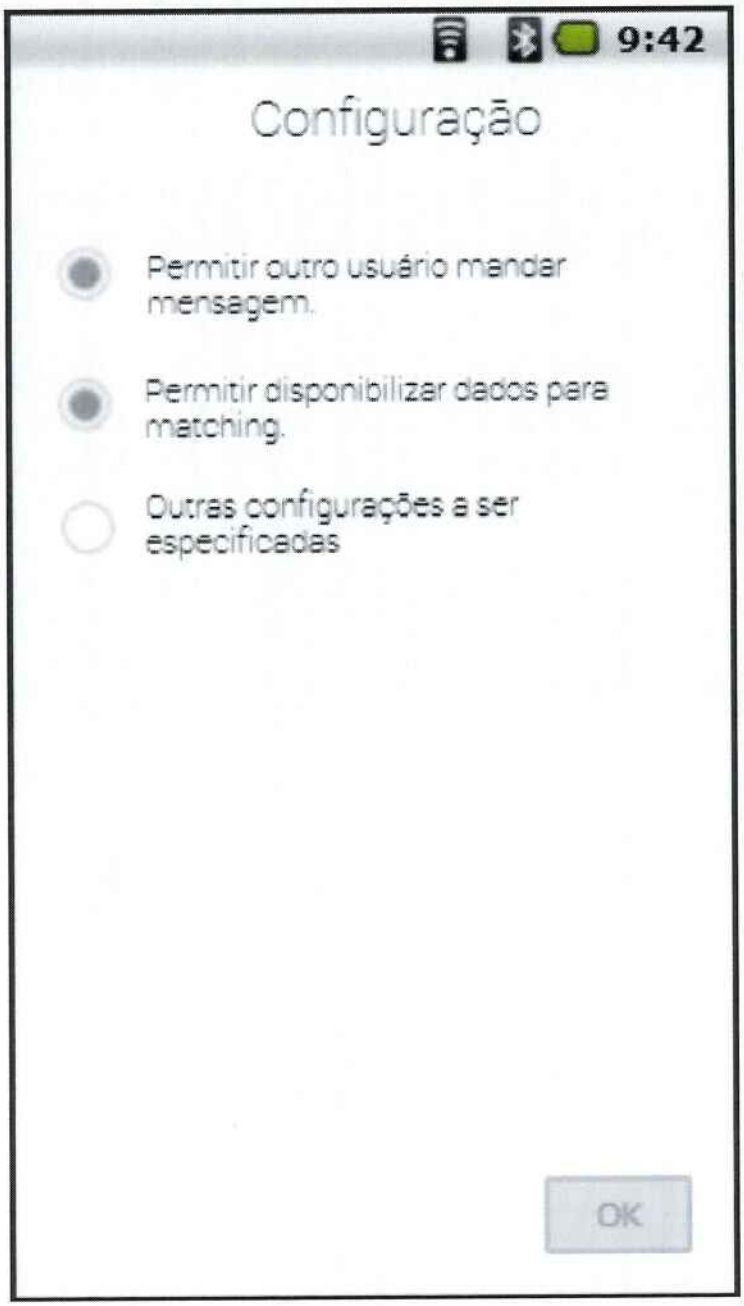

Figura 12 – Tela 14 – Configurações

Descrição: Nessa tela o usuário pode configurar as suas preferências de sistema.

2.5. Diagrama de classes simplificado

A Figura 13 apresenta o Diagrama de Classes simplificado do Atairu. O Diagrama de Classes completa se encontra no APÊNDICE C – DIAGRAMA DE CLASSES.

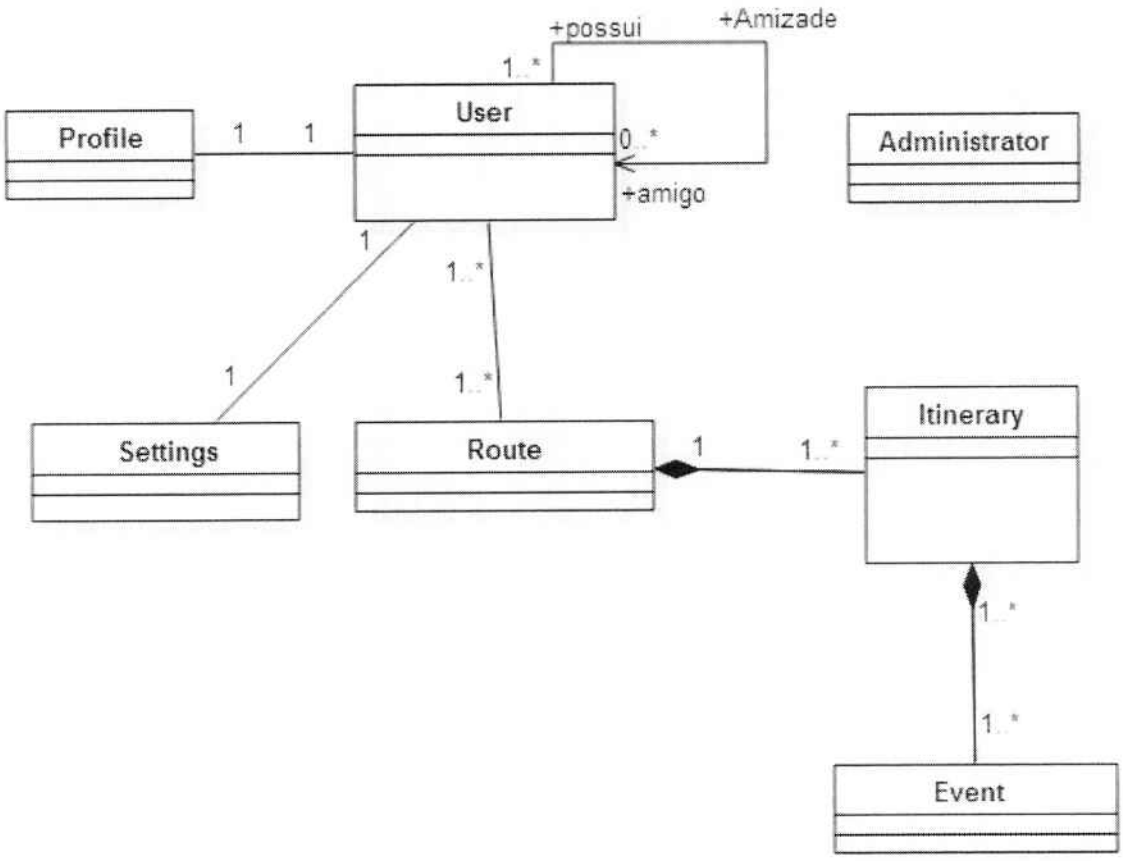

Figura 13 – Diagrama de Classes Simplificado

Para efeito de codificação, foi decidido utilizar nomes em inglês para que a API do Atairu, caso seja utilizada por outros sistemas não tenha barreira devido à língua portuguesa. Tem-se, a seguir, a descrição da responsabilidade das classes.

2.5.1. User

É a classe responsável por armazenar os dados cadastrais do usuário. São eles: nome, senha, país, cidadania, e-mail, foto e data de nascimento. Os métodos associados a esta classe são os responsáveis pela execução do *login*, *logout* e cadastro do usuário.

2.5.2. Profile

É a classe responsável por conhecer os dados relativos ao perfil social do usuário, como os interesses, e uma descrição pessoal. Associado a esta classe estão os métodos para gerenciamento dos atributos mencionados.

2.5.3. Settings

É a classe responsável por conhecer as configurações do usuário e as permissões para utilizar as funcionalidades do sistema. Associado a esta classe estão os métodos para modificação dessas configurações.

2.5.4. Route

É a classe responsável por conhecer o nome e a descrição de cada rota, os usuários que estão associados a esta rota e todos os roteiros (*Initeraries*) que pertencem a essa rota. Os métodos implementados por essa classe são r*elacionados com a modificação do seu nome e descrição, adição ou remoção de usuário associadas* à rota e adição, remoção e edição de *Itineraries*.

2.5.5. Itinerary

É a classe responsável por armazenar os dados do roteiro: datas de início e fim, a cidade, e possui um ou mais Eventos associados. Os métodos implementados por essa classe são relacionados com a modificação das datas de início e fim, e os Eventos associados.

2.5.6. Event

É a classe responsável por armazenar o nome, a descrição, o horário de funcionamento, a data de início, a data de fim e os *tags* que auxiliam na classificação do evento. Os métodos implementados por essa classe estão relacionados com a modificação dos seus atributos.

2.5.7. Administrator

É a classe responsável por armazenar os dados do administrador de sistema que são seu nome e sua senha e seus métodos estão relacionados com as tarefas administrativas.

2.6. Considerações sobre a Especificação do Sistema

A especificação do sistema seguiu o processo de metodologia descrita por Bezerra (2006), além da utilização do conhecimento que foi obtido pelo grupo durante os cursos de Engenharia de Software.

Com relação aos protótipos de telas descritas no APÊNDICE B – PROTÓTIPOS DE TELAS, eles foram usadas apenas como base para a implementação das telas na aplicação móvel. Foram feitas modificações do *design* das telas, durante a implementação, para se ter uma interface mais intuitiva ao usuário.

3. CONCEITOS E TECNOLOGIA

Este capítulo apresenta os conceitos relacionados com Arquitetura Orientada a Serviços (SOA), RESTfull Services APIs, iOS, Ruby on Rails, Plataforma Heroku e SGBD PostgreSQL, e as Provas de Conceito realizadas.

3.1. Arquitetura Orientada a Serviços (SOA – *Service Oriented Archtecture*)

A necessidade em fornecer e a demanda em consumir, em um mundo dinâmico, exigem a formulação conceitual de arquiteturas que possam moldar sistemas adequados que as auxiliem. Para os sistemas voltados para a disponibilização de serviços, SOA é a base para o desenvolvimento destes sistemas. SOA é um paradigma de arquitetura para organizar e usar competências que podem estar sob controle de domínios diferentes (OASIS, 2006).

O serviço, que caracteriza a unidade fundamental de SOA, é composto por diversas operações que satisfazem a sua execução; no caso do serviço de gerenciamento de usuários em um sistema, por exemplo, contém operações de cadastro, de remoção, edição etc.

Para organizar estas operações e o conjunto de serviços, SOA apresenta as seguintes características principais (FUGITA; HIRAMA, 2012):

- **Interoperabilidade** – permite a interação entre sistemas, independentemente da tecnologia utilizada pelas partes.
- **Padronização** – utilização de padrões que possibilitam a interoperabilidade entre sistemas, como o WSDL(*Web Service Definition Languague*) e SOAP(Simple Object Access Protocol).
- **Baixo acoplamento** – baixa dependência com outros componentes, possibilitando modificação, substituição ou melhoria sem maiores problemas.
- **Modularização** – desenvolvimento do sistema em módulos ou componentes, apresentando baixo acoplamento entre as partes.

A divisão em módulos tem, como finalidade, a melhor visualização do sistema para os desenvolvedores que irão utilizá-los, sem ter a necessidade de conhecer os detalhes internos, mas se importando com a funcionalidade e o resultado do serviço.

Para tanto, estes módulos precisam ter baixo acoplamento para que haja maior independência de cada serviço e, por consequência, seja possível ser reutilizado sem maiores problemas.

A tecnologia de implementação *Web Service*, orientada a serviços, proporciona a comunicação entre vários sistemas por causa da crescente utilização da internet, permitindo a construção de serviços nos moldes da arquitetura SOA.

O Atairu segue os padrões *Web Services*, com o propósito de ter a interoperabilidade entre plataformas, tendo em vista que é usado em *smartphones*, cujas plataformas de operação são variadas.

3.2. Restful Services APIs

Além do conceito de SOA, foi utilizado o conceito de *RESTful* na arquitetura de Atairu. A palavra *RESTful* significa que o sistema está nos moldes do estilo arquitetural *REST* (*Represantational State Transfer*), como descrito na dissertação de Fielding (2000). Quando há uma comunicação entre duas partes do sistema do tipo cliente-servidor, as requisições da parte do cliente têm por objetivo obter ou modificar o estado do servidor e isto é possível através de trocas de informações de suas representações de estados; esta dinâmica se deu origem ao termo *REST*.

A arquitetura *REST* aplica várias restrições para seus componentes e elementos de dados. Algumas destas restrições são:

- ***Client-Server***: separar a arquitetura em cliente e servidor com o objetivo de distinguir as tarefas de cada componente para que o desenvolvimento seja independente um do outro. O servidor cuida da parte de negócios e acesso ao banco de dados; e o cliente cuida da interface do usuário, a apresentação das informações.
- ***Stateless:*** qualquer requisição que seja feita pelo cliente deve conter individualmente todas as informações necessárias para que o servidor possa processar o serviço requisitado adequadamente. Portanto, o cliente, e não o servidor, deve guardar as informações necessárias para guardar o estado atual.
- ***Cacheable:*** como as requisições são independentes uma das outras, é possível que haja requisições do mesmo tipo, aumentando o

processamento desnecessariamente. Para não afetar o desempenho, é preciso que as informações possam ser guardadas em *cache*.

- ***Layered System:*** arquitetura em camadas para que seja possível a adição ou a exclusão de uma camada sem afetar as outras.

Aplicando as restrições de arquitetura para os padrões de *Web Services*, é necessário ter os seguintes requisitos: uma base URI (*Universal Resource Indicator*) que seja única; um formato padrão para os tipos de dados que serão utilizados, como por exemplo JSON ou XML; padronizações dos métodos HTTP em GET, PUT, POST ou DELETE, por exemplo, sendo que cada um destes métodos tem uma função distinta.

O desenvolvimento de Atairu utiliza a linguagem Ruby, seguindo os conceitos SOA e *RESTful*, para a implementação do *backend* e uma interface HTTP com métodos GET, PUT e POST.

3.3. iOS

O mercado de aplicativos móveis está atualmente polarizado entre as plataformas Apple iOS e Google Android. Tendo em vista esse mercado, realizou-se uma pesquisa (DREDGE, 2013) (EVANS, 2013) que auxiliasse a tomar a decisão da plataforma a ser utilizada para o desenvolvimento do *frontend* do Atairu.

Atualmente a fatia de mercado da plataforma Android é muito maior que a plataforma iOS. No entanto, o iOS ainda é a plataforma favorita para *startups* e testes de aceitação do mercado. Isso se deve principalmente aos seguintes fatores:

- **Baixa fragmentação.** Aplicativos desenvolvidos para a plataforma iOS funcionam perfeitamente numa pequena gama de iPhones e iPads: geralmente entre 6 e 8 diferentes dispositivos Apple. Para a plataforma Android, existem cerca de 12000 diferentes dispositivos no mercado, com uma grande variedade de tamanhos de tela, processadores e versões de Android que ainda estão em uso. Essa diversidade de versões e dispositivos provoca um problema chamado fragmentação, ou seja, torna o desenvolvimento muito mais complexo e o suporte pós-desenvolvimento muito custoso, pois para cada versão da

plataforma Android é necessário lidar com os *bugs* e detalhes próprios da versão.

- **APIs.** Tanto o Android quanto o iOS possuem uma biblioteca extensa e semelhante de software disponível aos seus desenvolvedores. No entanto, a maior parte do trabalho é feita em controladores: *iOS ViewController*, no iOS, e o *Android Activity*, na Android. A plataforma iOS, além deste conjunto de bibliotecas, semelhante nas duas plataformas, possui um conjunto extra e único de *frameworks* chamado Core Data Framework que facilita o *design* do sistema.
- **Ambiente.** O IDE (*Integrated Development Environment)* torna o desenvolvimento do aplicativo muito mais produtivo em ambas as plataformas. O Xcode da Apple é rápido, poderoso e prestativo, sem ser intrusivo. O *debugger* funciona sem problemas, e o simulador é rápido e responsivo. Já a IDE de desenvolvimento para a plataforma Android é o Eclipse que adiciona uma complexidade maior aos projetos além de operar de maneira mais lenta e ser menos intuitivo.
- **Facilidade em monetização.** Apesar da grande fatia de mercado do Android, em cerca de 82% do mercado, os usuários da plataforma são mais sensíveis ao preço, tornando mais difícil uma eventual monetização o produto. Os usuários iOS ainda são os que estão mais dispostos a comprarem aplicativos através da App Store da Apple.

A partir desses fatores optou-se pelo desenvolvimento do *frontend* do Atairu na plataforma iOS, utilizando a versão 5.1.1 do Xcode.

3.4. Ruby on Rails

O Rails é um *framework* que facilita o desenvolvimento de aplicações WEB, conhecido também por Ruby on Rails por utilizar a linguagem de programação Ruby. Esta linguagem foi criada por Yukihiro Matsumoto e ficou popular depois da criação do *framework* Rails. Considerada uma ferramenta para fazer aplicações web de acordo com Hartl e Prochaska (2008), utiliza quatro conceitos:

- *Model-View-Controller* (MVC) – utiliza o padrão de arquitetura que estrutura o sistema em modelo, apresentação e controle.

- *Don't Repeat Yourself* (DRY) – é uma prática de programação que diminui a duplicação desnecessária de código, futuros *bugs* e complexidade causada pela repetição constante.
- Convenção sobre configuração (*convention over configuration*) – busca convenções que possam diminuir configurações de sistema, isto é, codificações mais eficientes e menos trabalhosas.
- *Object-Relational Mapping* (ORM) – é a conversão de dados entre tipos diferentes do sistema em linguagem de programação orientada a objetos, isto é, os vários tipos de dados são transformados em objetos dentro do banco de dados.

Uma biblioteca que ajuda a testar as aplicações ao decorrer do desenvolvimento é o RSPED. Os testes podem ser feitos em um único módulo ou em quantos se julgarem necessários, pois a modificação de uma parte do código pode afetar outras. Desta forma, é possível verificar se os módulos desenvolvidos anteriormente foram afetados ou não, de maneira automatizada.

Ruby on Rails tem mais de 3400 contribuidores, além de sua equipe principal de desenvolvimento, possuindo uma vasta quantidade de documentação da ferramenta e de bibliotecas que ajudam a otimizar a construção de aplicações web. Tendo em vista esta facilidade oferecida pela ferramenta e a grande quantidade de documentação disponível para ser consultada, decidiu-se utilizar o Rails para implementar a aplicação web do projeto de formatura.

Foi utilizada a versão 2.1.2p95 do Ruby e a versão 4.1.4 do Rails.

3.5. Plataforma Heroku

Umas solução simples e prática, que foi encontrada para hospedar a aplicação em Ruby, foi o serviço *cloud Platform as a Service* (PaaS), o Heroku (HEROKU, 2014). Esta plataforma utiliza a palavra *dyno* como sendo uma unidade isolada de virtualização UNIX, que fornece o ambiente necessário para executar uma aplicação. A Figura 14 mostra os diferentes *dynos* disponíveis.

Dyno Size	Memory (RAM)	CPU Share	Multitenant	Compute (2)	Price/dyno-hour
1X	512MB	1x	yes	1x-4x	$0.05
2X	1024MB	2x	yes	4x-8x	$0.10
PX	6GB	100% (1)	no	40x	$0.80

Figura 14 – Configuração de escalabilidade da plataforma Heroku

Fonte: https://devcenter.heroku.com/articles/dyno-size

O sistema Atairu utiliza o dyno 1X que fornece o poder de processamento de 512MB de RAM e 1 CPU *share* (uso da porcentagem mínima de CPU, definida pelo Heroku), além de um pequeno banco de dados (10 KB) para testar a aplicação durante seu desenvolvimento. Esta configuração de processamento é a recomendada para aplicações com baixa requisição de *web service*s como a deste trabalho que será usado apenas para testes. São fornecidos, como padrão, 750 dyno horas de graça por aplicação, portanto não há custos provenientes para o desenvolvimento deste trabalho.

Uma característica deste serviço *cloud* é a utilização do Git, ferramenta de versionamento e repositório de código, na execução do *deploy* das aplicações, o que adiciona o benefício de gerenciamento e de versionamento das aplicações. Outro benefício é a facilidade em configurar variáveis de ambiente por meio de um arquivo externo, sem a necessidade de codificação para configurar.

3.6. SGBD PostgreSQL

Neste trabalho, foi utilizado o PostgreSQL como SGBD, versão 9.3.4. Ele satisfaz a necessidade do sistema quanto ao quesito banco de dados e, além disto, a utilização deste SGBD é requerida pela plataforma Heroku que hospeda a aplicação web do sistema. Este PaaS (*Platform as a Service*) foi escolhido devido à sua automatização quanto ao processo de *deploy* de aplicações em Ruby, isto é,

não há necessidade de configurar ou adaptar os sistemas que são hospedados nesta plataforma, basta apenas fazer *deploy* da aplicação.

O PostgreSQL não é o mais usado dentre os *open sources*, porém tem mostrado crescente adesão principalmente por ser considerada mais robusta por causa de seus padrões mais rigorosos. As suas principais características, são apresentadas a seguir:

- É uma ferramenta poderosa de gerenciamento de banco de dados relacional;
- Na questão de confiança e integridade dos dados é a melhor opção para desempenho de *procedures* customizadas no banco de dados (TEZER, 2014).

3.7. Provas de Conceito

Para implementar o sistema Atairu foi feito uma busca de dados e informações que fossem relevantes à rede social para viajantes. As informações referentes aos eventos e locais são obtidas através de bancos de dados já existentes e que permitem o acesso aos seus dados através de APIs.

Foram encontradas duas APIs com essa funcionalidade, focada em informações de locais: Foursquare (FOURSQUARE, 2014) e TripAdvisor. A prova de conceito dessas APIs foi executada apenas com o API do Foursquare, sendo que não foi possível testar o API do TripAdvisor por restrições legais de acesso a seus dados.

O Facebook (API do Facebook) (FACEBOOK, 2014) foi escolhido para fazer prova de conceito, devido à grande informação disponível com relação às preferências do usuário. Para a fonte de dados de previsões do tempo, foi escolhido o API do Forecast IO (FORECAST, 2014).

3.7.1. Provas de Conceito - Foursquare

O Atairu utiliza o *Foursquare* para busca de informações referentes aos eventos. O *Foursquare* usa o conceito de *venue* que significa local onde algo acontece ou ocorrência de algum evento; portanto, sua API retorna as informações

referentes aos eventos em formato JSON. As *venues* podem ser locais fixos ou eventos sazonais.

O API permite uma busca no banco de dados do Foursquare para obtenção de informações que incluem dicas, fotos, *check-in-counts* (contagem de *check-in*) e *hereNow* (quem se encontra no local agora). As pesquisas podem ser feitas ao redor de um local ou através da cidade inteira. Tudo isto pode ser requisitado pelo serviço de *venues,* sem a necessidade de o usuário estar autenticado na Foursquare e disponível a altas taxas de requisições: cinco mil requisições por hora, podendo requisitar mais se necessário.

3.7.2. Utilização dos Dados do Foursquare

Uma das condições de acesso à API do *Foursquare* é que, se os seus dados forem armazenados em algum servidor ou dispositivo, não é permitido manter um banco de dados separado com informações melhoradas da *venues*, isto é, toda e qualquer melhoria na informação deve ser enviada primeiramente ao banco de dados do Foursquare, para então ser acessado via *Web Services*. Outra condição de uso é de mencionar que a fonte dos dados é o Foursquare.

Este trabalho implementa a sugestão de roteiros e isto não quebra as condições de uso da API, pois é feito uma melhoria quanto a seleção de *venues* e não na informação que ela possui. Portanto, a fonte de dados escolhida para a busca por locais é o Foursquare.

A utilização da API exige a observação de regras de uso; mesmo que não precise de um usuário autenticado no sistema Foursquare para obter seus dados, as políticas de uso devem ser respeitadas. Tais políticas são:

- Quando houver utilização dos dados de seu banco de dados, é necessário explicitar de maneira apropriada de que a fonte é o Foursquare.
- O armazenamento dos dados em outro banco de dados é permitido, desde que seja feita a atualização periódica, para que não sejam disponibilizados dados desatualizados aos usuários.

- Não é permitida a combinação de informações de locais do *Foursquare* com as de outras fontes, visando melhorar ou modificar os dados fornecidos pelos serviços da API.
- Caso haja dados que precisem ser modificados para melhorar a qualidade da informação fornecida, deve ser enviada uma requisição de edição da informação para que haja a modificação primeiramente no *Foursquare*.
- Existe a restrição para disponibilizar, no máximo, 4 dicas ou fotos da mesma *venue*; caso contrário deve haver um direcionamento para a página do Foursquare.
- Não é permitido vender, alugar ou transferir os dados do Foursquare a terceiros; a obtenção dos dados deve ser feita diretamente por meio da API.

As restrições para o sistema são os casos de uso UC30, UC31 e UC32 especificados no Modelo de Casos de Uso da seção 2.2, os quais não foram implementados, pois vão contra as condições de uso do API do Foursquare. O Atairu, portanto utiliza os dados relacionados às *venues* obtidas apenas do Foursquare.

3.7.3. Prova de Conceito - Facebook

No que se refere à obtenção de dados pessoais, a fim de determinar o perfil das preferências do usuário, foi realizada uma prova de conceito do Facebook API, devido à quantidade de usuários cadastrados na rede social e à facilidade de obtenção de informações do usuário. Através da API, é possível acessar as preferências dos usuários que podem ser visualizadas em sua rede de amigos.

Os dados analisados foram:

- *likes* – indica o que o usuário do Facebook gostou, podendo ser uma página do Facebook ou algum perfil de uma figura pública.
- *about* e *interests* – frases genéricas, sem categorização do tipo de informação.
- *tagged_places* – lugares visitados pelo usuário, podendo ser casualmente um restaurante ou em uma viagem.

Estes dados não são precisos quanto ao gosto do usuário em relação a viagens, pois não apresentam uma padronização quanto ao tipo de dado informado, como o *likes*, *about* e o *interests*.

Além destes, foram obtidos, através da API, dados cadastrais como nome, país de origem, e-mail, data de aniversário, primeiro nome, sobrenome, gênero, id de usuário e foto. Para o propósito de descobrir a preferência do usuário, poderia ter sido implementado um sistema de reconhecimento de padrões baseado no conceito de *Machine Learning* da área de IA (Inteligência Artificial), tendo como base a análise de *tagged_places*; porém este método de obtenção da preferência do usuário é invasivo e, por isto, foi decidido não o utilizar para este projeto.

A obtenção dos dados do usuário somente é possível com a sua autorização, existindo, portanto, uma restrição de acesso. Para os dados cadastrais básicos como nome e e-mail não é necessário permissão especial; a permissão de autenticar em uma aplicação por meio do Facebook já é o suficiente para obter estes dados básicos. Portanto este API não é utilizado para a obtenção da preferência do usuário, mas é utilizado para obter os dados cadastrais básicos, pela aplicação móvel, com a finalidade de fazer apenas a autenticação do usuário no sistema.

3.7.4. Utilização dos Dados de Facebook

O Facebook oferece fluxos de *login* para diversos aparelhos e sistemas. Alguns são fáceis de serem implementados, utilizando seu SDK oficial, enquanto que outros precisam de codificação adicional para efetuar o *login* do usuário no Facebook.

Para garantir qualidade de experiência ao usuário, o Facebook requisita revisão de acesso aos dados mais restritos antes que a aplicação peça permissão de acesso a dados pessoais ao usuário pela aplicação. Esta revisão pode demorar de 3 a 7 dias para saber se será possível requisitar ao usuário a permissão de acesso de seus dados, classificados como sensíveis pelo Facebook.

As informações de acesso aos dados do usuário, sem ter o pedido de revisão ao Facebook são perfil público, email e amigos do usuário (*app friends*). O perfil público disponibiliza informações de descrição sobre o usuário (*about*). A lista de amigos do usuário neste caso é composta pelos amigos que utilizam a mesma

aplicação que o usuário e não a lista completa de amigos do Facebook. Para propósito de teste e de desenvolvimento de aplicações, não há restrição de acesso a essas informações.

Os demais atributos precisam passar pela revisão Facebook quanto à utilização destas informações na aplicação ou sistema, sendo classificados nas seguintes categorias:

- Propriedades estendidas do perfil - Propriedades sensíveis que podem ou não ser parte do seu perfil público.
- Permissões estendidas - Inclui as informações mais sensíveis do perfil. Uma dessas permissões permite a publicação de histórias no seu *profile* do Facebook. Todas as permissões estendidas aparecem em uma tela separada do fluxo de *login* para que o usuário possa decidir fornecer ou não essas informações.
- Permissão de página - Permite administrar qualquer página do Facebook que a pessoa gerencie.

3.7.5. Prova de Conceito - Forecast IO

A condição climática de uma região afeta nas atividades que podem ser feitas, portanto a previsão do tempo deve ser levada em conta no algoritmo de sugestão de roteiros. Para a obtenção de dados da previsão de tempo, foi executada uma prova de conceito do API do Forecast IO. Estes dados são utilizados no algoritmo de sugestão de roteiros.

O Forecast IO permite executar consultas para a maioria das localizações do mundo, retornando as seguintes informações:

- Condições climáticas atuais;
- Previsão do tempo por minuto em um período de 1 hora;
- Previsão do tempo por hora em um período de 48 horas.
- Previsão do tempo por dias em um período de 1 semana.

Há duas chamadas principais da API que podem ser feitas. A primeira chamada retorna a previsão do tempo atual para os próximos sete dias por meio da requisição *web* abaixo:

https://api.forecast.io/forecast/APIKEY/LATITUDE,LONGITUDE

A segunda chamada permite fazer uma consulta para um tempo específico, passado ou futuro, sendo na maioria dos casos 60 anos atrás até 10 anos no futuro. A requisição *web* para esta chamada segue abaixo:

https://api.forecast.io/forecast/APIKEY/LATITUDE,LONGITUDE,TIME

A decisão da escolha deste serviço, como fonte de dados de previsões do tempo, foi a simplicidade de utilização da API, não sendo necessário uma característica especial para incluir tal função no sistema Atairu.

3.8. Considerações Finais do Capítulo

Os conceitos apresentados neste capítulo ajudaram a definir as tecnologias e a arquitetura do sistema projetado. Para a escolha das tecnologias foram ponderados os seguintes quesitos:

- Curva de aprendizagem e relevância da tecnologia no mercado;
- Capacidade de reutilização da aplicação lógica por outros sistemas, através de uma API de serviços;
- Adaptabilidade ao conceito de *RESTful Services;*
- Facilidade de monetização.

A partir do levantamento das requisições de dados necessários para o funcionamento do sistema Atairu, foram realizadas provas de conceito das APIs de serviços que permitiram o seu fornecimento de maneira menos restritiva. Portanto, as tecnologias escolhidas para o desenvolvimento do sistema são:

- Tecnologia de implementação SOA: *Web Services*;
- *Frontend*: Plataforma iOS, Xcode versão 5.1.1;
- *Backend*: linguagem Ruby versão 2.1.2p95 e *framework* Rails versão 4.1.4;
- Servidor de aplicação *cloud*: Plataforma Heroku;
- SGBD: PostgreSQL versão 9.3.4;
- APIs: Foursquare, Facebook e Forecast IO.

4. DESENVOLVIMENTO

Este capítulo apresenta Divisão de Atividades, Arquitetura de Sistema, Modelagem de Dados, Algoritmo de Geração de Roteiros, *Backlog* de Implementação do *Backend* e Resultados de Implementação do *Backend*.

4.1. Divisão de Atividades

A divisão de atividades de um grupo é feito pelo gerente de projetos e este papel foi designado a Márcio Carlos Perin Tedesco, principalmente pelo seu conhecimento técnico e habilidade em organizar tarefas a serem executadas.

As atividades a serem realizadas depois do primeiro quadrimestre, no qual foi feita a especificação do sistema, foram agrupadas em três partes: elaboração da aplicação lógica (*backend*), elaboração da aplicação móvel iOS (*frontend*) e elaboração da monografia. Estas atividades foram distribuídas inicialmente de forma que dois dos integrantes desenvolvessem o *backend* e o outro integrante o *frontend*, ficando a monografia para ser escrita por todos os integrantes em paralelo à respectiva atividade.

No decorrer do desenvolvimento deste trabalho, foi decidido que deveria haver uma mudança na distribuição de atividades, em função do prazo de entrega da monografia. Com este intuito foi feita a transferência de responsabilidade de um dos integrantes da atividade *frontend* para se dedicar na elaboração da monografia. A divisão inicial e a divisão final das atividades são mostrada na Figura 15.

Devido a maior conhecimento da tecnologia Ruby e Rails e a agilidade em codificar de Mácio Carlos Perin Tedesco, foi decidido que a responsabilidade em finalizar a monografia ficaria para Tiago Lee.

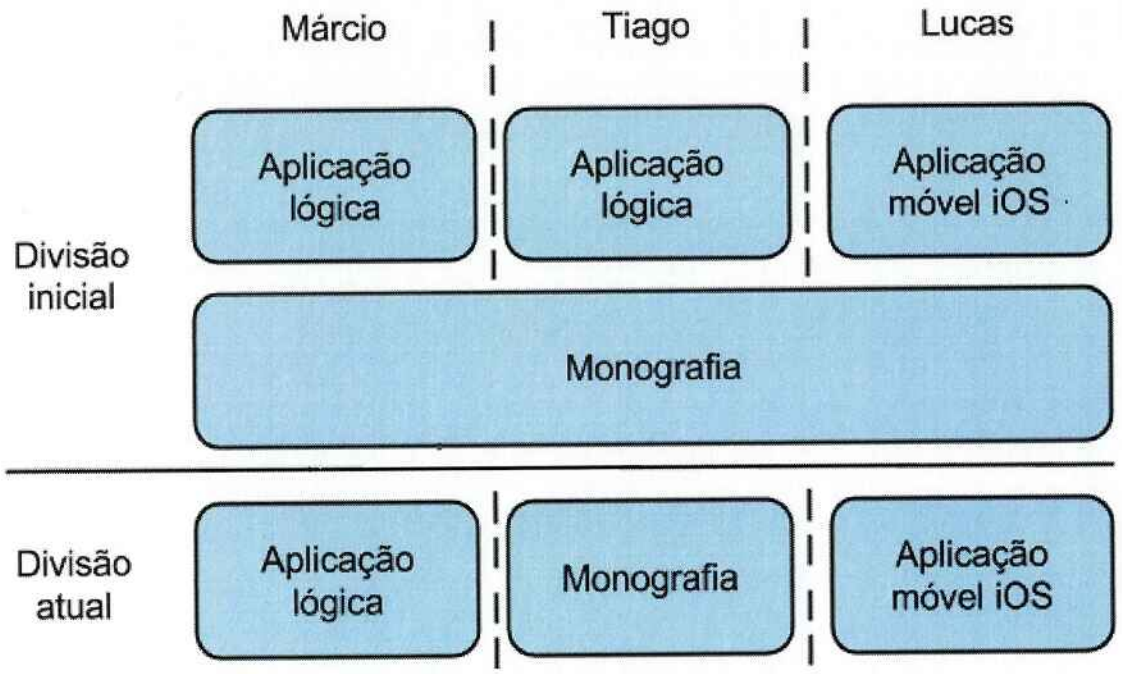

Figura 15 - Distribuição de atividades

4.2. Gerenciamento do projeto

O processo de gerenciamento de projeto escolhido teve como inspiração o *SCRUM* que é um processo desenvolvimento de software ágil, com característica iterativa e incremental (VIEIRA, 2014).

A unidade básica de desenvolvimento em *SCRUM* é um *sprint* que pode ter uma duração de tempo variável, porém, deve ser previamente definida. Para o projeto a duração estipulada de cada *sprint* foi de 10 dias. A definição das *features* (funcionalidades), implementadas em cada *sprint*, foi realizada em conjunto com os três membros do projeto logo no começo da etapa de desenvolvimento. As funcionalidades implementadas em cada *sprint* tanto do *backend* quanto do *frontend* foram casadas de maneira a ser possível realizar um teste integrado ao final de cada *sprint*. Portanto, ao final de cada *sprint* sempre foi agendado um *deploy* das modificações realizadas no *backend* ao servidor de lógica do sistema e a realização de um teste integrado com o *frontend*.

Eventuais dificuldades que levaram a atrasos foram gerenciadas através da alteração da quantidade de tarefas atômicas de cada *sprint* e aumento de tempo na dedicação ao projeto. Essa variação na distribuição do tempo dedicado ao *sprint* já

havia sido prevista devido à sazonalidade de provas e trabalhos durante o quadrimestre. As Figuras 16 e 17, apresentam as tabelas utilizadas no gerenciamento dos *sprints* do *backend* e do *frontend.*

RUBY ON RAILS - SPRINT SCHEDULE				
Sprint	**Focus**	**Sprint Start**	**Sprint End**	**Status**
Sprint #	*Main Features Included in Sprint (User Story Reference #)*	*Begins on Monday*	*Based on 1 or 2 week Sprints. Ends on Friday*	
0	User/Admn - Autenticaçao	9/10/2014	9/20/2014	Done
1	Profile - Tags	9/20/2014	9/30/2014	Done
2	Itinerary/Events	9/30/2014	10/10/2014	Done
3	Rotas/	10/10/2014	10/20/2014	Done
4	Amigos/Incluir Amigos/Busca Amigos	10/20/2014	10/30/2014	Done
5	Chat/Settings	10/30/2014	11/9/2014	Done
6	Match	11/9/2014	11/19/2014	Done

Figura 16 – *Backend Sprint Schedule*

iOS - SPRINT SCHEDULE				
Sprint	**Focus**	**Sprint Start**	**Sprint End**	**Status**
Sprint #	*Main Features Included in Sprint (User Story Reference #)*	*Begins on Monday*	*Based on 1 or 2 week Sprints. Ends on Friday*	
0	Preparação Ambiente	8/25/2014	8/31/2014	Done
1	Data Model/ Data Provider/Configuraçao Inicial	9/9/2014	9/19/2014	Done
2	Home/Login/Cadastro	9/20/2014	9/30/2014	Done
3	Meu Perfil / Perfil Amigo	10/1/2014	10/11/2014	Done
4	Rotas/	10/21/2014	10/31/2014	Done
5	Roteiros/Eventos	11/1/2014	11/11/2014	Done
6	Amigos/Incluir Amigos/Chat/Busca Amigos	11/12/2014	11/22/2014	Done
7	Match	11/23/2014	12/3/2014	Done
8	Identidade Visual e Design(Simples)			Done

Figura 17 – *Frontend Sprint Schedule*

Além dessas tabelas, uma tabela auxiliar, o *Sprint Backlog,* criada juntamente com as tabelas de gerenciamento dos *sprints*, definiu as tarefas atômicas a serem realizadas durante cada *sprint* a fim de implementar uma funcionalidade. Essa tabela

foi modificada dinamicamente ao longo do projeto e ajudou a rastrear as dificuldades do grupo e gerenciar o tempo dedicado a cada tarefa dentro do *sprint*. A Figura 18 apresenta um extrato dessa tabela. Cada entrada na tabela corresponde a uma ação atômica e seu status.

Além dessas tabelas, a comunicação de erros entre o *frontend* e o *backend* e as modificações a serem implementadas pelo *backend* foram realizadas através do sistema de criação e resolução de *issues* (erros), disponível na ferramenta utilizada para o gerenciamento de código e repositório Git, o qual é apresentada na Figura 19.

Sprint 2 - 30/09/2014	Criar model Profile	Done
	Criar associacao do model Profile com model user	Done
	Criar action create no Profiles controller	Done
	Criar action update no Profiles controller	Done
	Criar model Tags	Done
	Criar associacao do model Tag com o model profile	Done
	Criar action create(create-tag) no Tags controller	Done
	Criar action index (get-tags) no Tags controller	Done
	Criar action destroy(destroy-tag) no Tags controller	Done
	Documentar webservice Tags	Doing
	Criar action update do controller Registrations	Done
	Criar action show do controller Users	Done
	Revisar config/routes	Done
	Verificar se timeout do auth token do Devise funciona	Done
	Adicionar Profile à dashboard activeAdmin	Done
	Adicionar Tags à dashboard activeAdmin	Done
	Realizar Deploy Heroku	Done
	Teste integrado Mobile Sprint 2	Open
	Documentar webservice update-user-data	Done
	Documentar webservice get-user	Done
	Documentar webservice create-profile	Done
	Documentar webservice update-profile	Done
	Documentar webservice get-profile	Done
	Criar cenario e realizar testes SPEC get-user	Done
	Criar cenario e realizar testes SPEC create-profile	Done
	Criar cenario e realizar testes SPEC update-profile	Done
	Criar cenario e realizar testes SPEC get-profile	Done

Figura 18 – *Sprint Backlog*

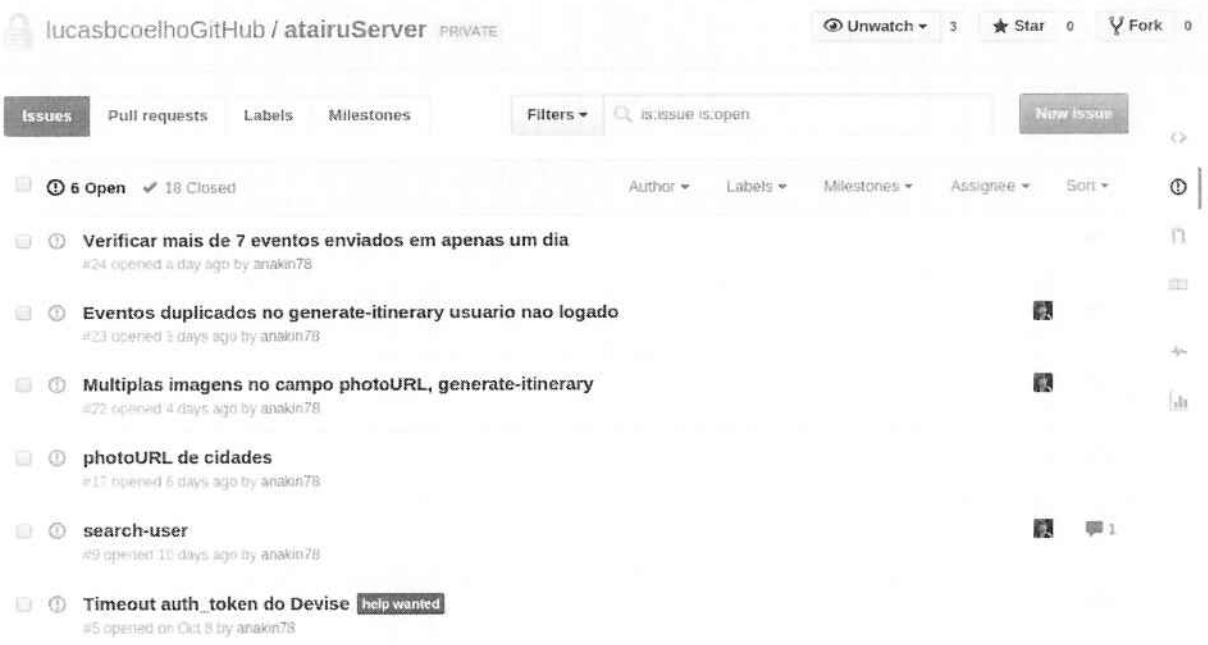

Figura 19 – Tela de *Issues*

4.3. Arquitetura de Sistema

A Figura 20 apresenta a arquitetura de sistema Atairu, através do Diagrama de Implantação, que apresenta os elementos de infraestrutura nos quais foi alocado o software.

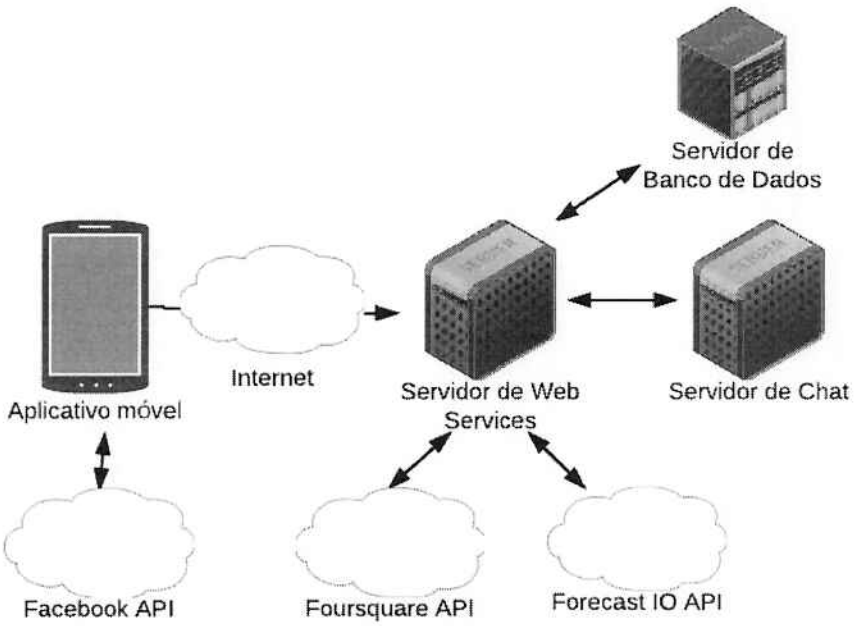

Figura 20 – Arquitetura de Sistema

O sistema é organizado em quatro camadas: camada de acesso aos dados, alocado no Servidor de Banco de Dados; camada de negócio, no Servidor de *Web Services* (contém o algoritmo e lógica do Atairu, além de expor os serviços web e gerenciar suas conexões) e no Servidor de Chat; a camada de interface, responsável pela apresentação do conteúdo no dispositivo móvel e a camada de sistemas externos, APIs das fontes de dados.

4.4. Modelagem de Dados

Nesta seção é descrito o modelo de dados construído para armazenar as informações do Atairu no banco de dados PostgreSQL.

O diagrama de Entidade-Relacionamento foi dividido em duas partes por razões de melhor visualização, as quais são apresentadas nas Figuras 21 e 22.

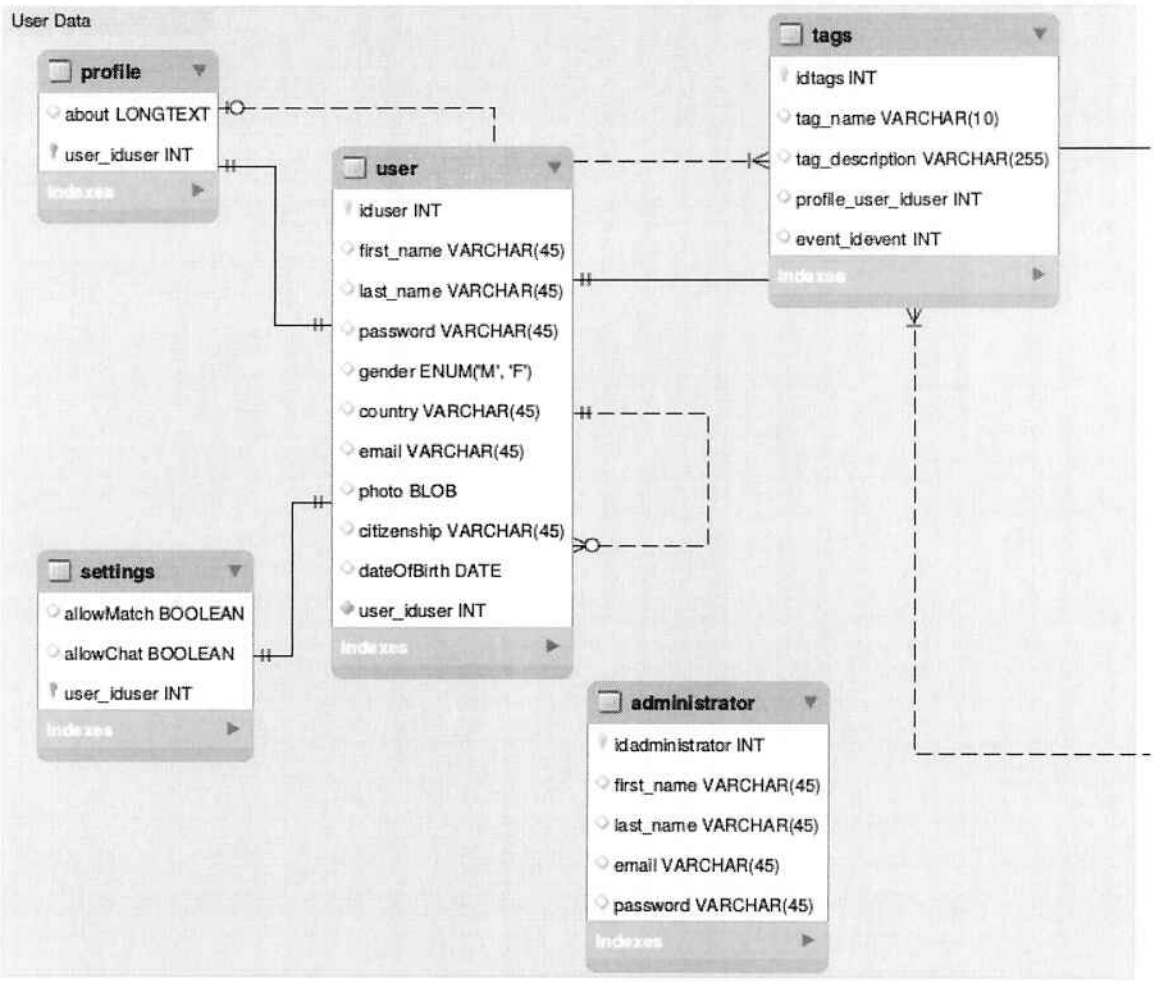

Figura 21 - Diagrama Entidade-Relacionamento - Usuário.

A Figura 21 apresenta as entidades que armazenam as informações referentes ao usuário que, devido ao aspecto de rede social do Atairu, é um grande concentrador de informações. A Figura 22 apresenta as tabelas que armazenam as informações envolvidas com a lógica do sistema. As principais entidades que fazem parte do modelo de dados do Atairu são: *User* e *Itinerary*.

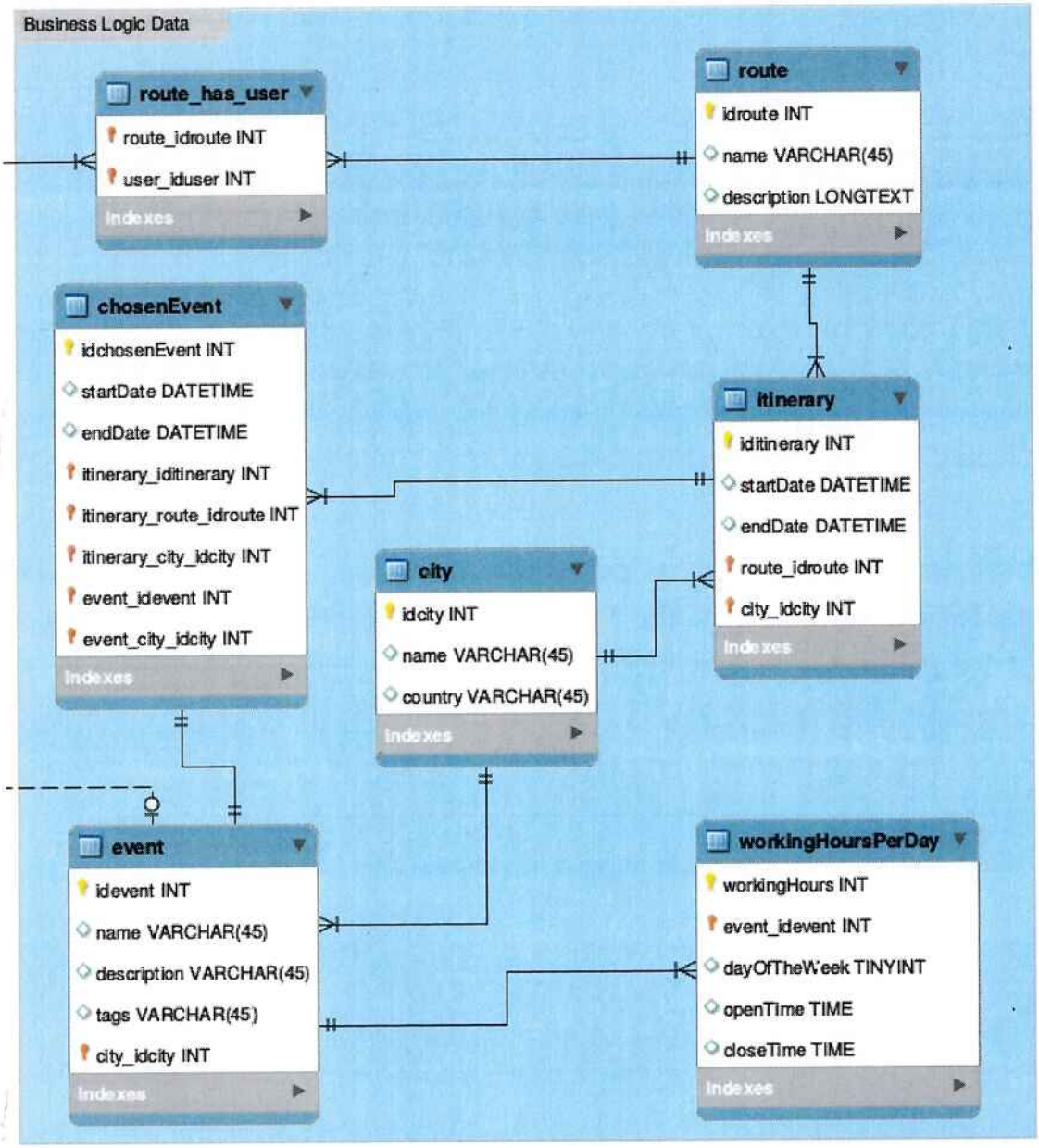

Figura 22 - Diagrama Entidade-Relacionamento - Lógica de Negócios.

A entidade *user* possui relacionamentos com entidades responsáveis por gerenciar seus dados, configurações no sistema, mensagens enviadas e recebidas e relacionamentos com outros *users*:

- *settings:* tabela que armazena as configurações do usuário. As duas principais configurações são a configuração que permite a presença do usuário nas buscas realizadas pela funcionalidade *Match* e a configuração que permite que o usuário seja encontrado através do mecanismo de busca de usuários;
- *message*: tabela que armazena as mensagens enviadas pelo usuário permitindo o acesso a um histórico de *chat* com cada usuário.
- *profile*: tabela que armazena as informações que podem ser visualizadas por outros usuários. Possui associação com a tabela de *tags*, de um para muitos.
- *friendship*: *join table* que armazena a relação de amizade entre dois *user_id*.

As entidades *tags* e *tagsCategories* fazem parte da lógica de um mapeamento de preferências do usuário, nomeadas de *tags,* para um sistema de categorias a serem buscadas na API do Foursquare e atuarem como um filtro de eventos relacionados às preferencias do usuário.

Associadas à entidade *Itinerary* estão as principais entidades relacionadas à lógica do aplicativo:

- *route:* é a entidade que relaciona os usuários com as informações que se referem à lógica do sistema. Armazena as informações a respeito da autorização de alteração de uma viagem, além de informações descritivas da viagem.
- *itinerary:* é a principal entidade da lógica de geração de roteiros. Armazena as informações a respeito das datas de começo e término de um roteiro em alguma cidade.
- *chosenEvent:* é a entidade que estabelece a ordem dos eventos em um determinado dia.
- *city:* é entidade que relaciona o eventos de uma cidade a um itinerário.
- *event:* é a entidade que acumula a maior parte dos dados do sistema. A tabela de eventos é carregada com eventos bem classificados e recomendados por muitos usuários da comunidade do Foursquare. Esses eventos são armazenados e, respeitando o termo de uso de

dados do Foursquare, são atualizados por uma sub-rotina com uma periodicidade menor que um mês.

4.5. Algoritmo de geração de roteiros.

O algoritmo de geração de roteiros utiliza conceitos de *location-awareness* e *context-awareness* (SCHILIT; ADAMS; WANT, 1994). Um dos grandes desafios que o desenvolvimento do Atairu enfrentou foi a obtenção do perfil do usuário. Portanto, para poder mapear as buscas de *venues* ao API do *Foursquare,* foi realizado um *mapping* entre as *tags* que são as preferências que o usuário pode associar ao seu perfil e entre as *Categories* de busca do Foursquare.

Sistemas de recomendação se utilizam da opinião de comunidades para ajudar os usuários a identificar itens úteis para um espaço de busca extenso (e.g., Amazon inventory, Netflix movies). A técnica utilizada por muitos destes sistemas é denominada *collaborative filtering* (CF), a qual analisa opiniões passadas da comunidade para encontrar correlações de usuários similares e itens para sugerir *k* items personalizados (e.g. filmes) para um usuário *u* (LEVANDOSKI, 2012).

Técnicas de recomendação existentes se utilizam de *ratings* representados por triplas (*user, rating, item*), no entanto, não utilizando a geolocalização como um parâmetro na geração de roteiros. O Atairu se utiliza da geolocalização dos eventos como um dos parâmetros para obtenção do próximo evento da lista de eventos que será apresentada ao usuário. Além disso, dados relativos ao *feedback* dos usuários que utilizam a ferramenta *Foursquare* permitiram a utilização desses dados como parâmetros para a construção de um filtro colaborativo.

4.5.1. Categories *Foursquare*

O *Foursquare* oferece uma classificação de suas *venues*, através de *Categories*. Para obter informações mais precisas das *venues* buscadas, realizou-se um mapeamento entre *tags*, preferências do usuário e categorias que melhor correspondem a essas preferências. A Figura 23 apresenta a tabela de *categories* do Foursquare.

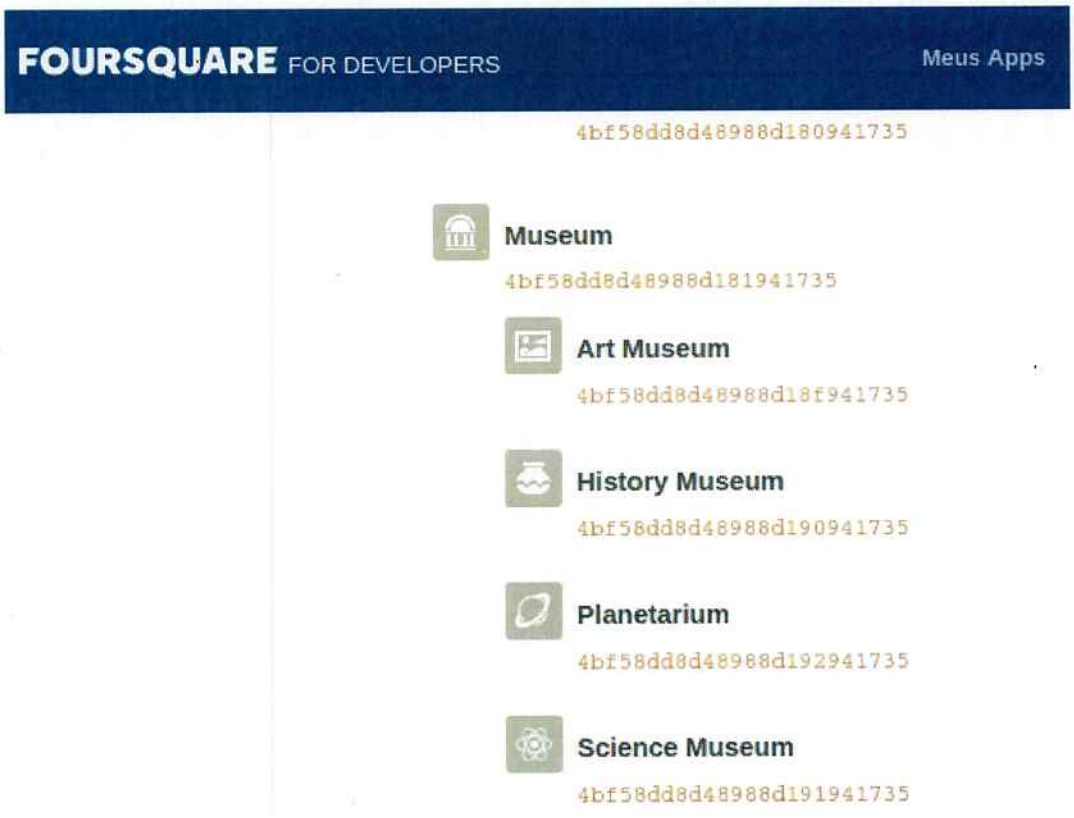

Figura 23 – Tela de *Categories* do Foursquare

4.5.2. TagsCategory

A cada *tag* criada para suprir uma preferência do usuário e contribuir para o desenho do seu perfil, foi associado um *array* de *foursquareCategoriesIDs*. Esses IDs são utilizados no algoritmo de geração de roteiros como um primeiro filtro para obtenção de listas de eventos mais adequados ao perfil do usuário.

Além disso, cada *tag* possui os atributos *tag_description,* e *tiers*. O atributo *tiers,* é um dos parâmetros utilizados na geração dos roteiros. As *tiers* é um *array* que aceita o conjunto de valores de 1 a 3 com combinações entre si. Esses valores correspondem ao seguinte mapeamento:

1 -> manha

2 -> tarde

3 -> noite

Esse conjunto de *tiers* define quais *tags* fazem sentido para a lógica de geração, dependendo do momento do dia. A *tag nightlife* só tem sentido durante a

noite, portanto é atribuída a seu conjunto de *tiers* o valor 3. As Figuras 24, 25 e 26 apresentam como são mapeadas as *Tags* para um conjunto de *foursquareIDs* através da classe *TagsCategory* que também guarda as informações sobre o nome e descrição da *Tag* relacionada assim como as *tiers* as quais essa *TagsCategory* pertence.

```
TagsCategory.create([{
        tag_name: 'nightlife',
        tag_description: 'Pra quem curte balada e uma noite agitada',
        #[Nightlife Spot]
        foursquareCategoriesID: ["4bf58dd8d48988d116941735", "4bf58dd8d48988d11b941735",
            "4bf58dd8d48988d11f941735", "4d4b7105d754a06376d81259"],

        tiers: [3]
    },
    {
        tag_name: 'most_common',
        tag_description: 'Pra quem curte visitar os pontos turísticos mais visitados',
        foursquareCategoriesID: [],
        tiers: [1,2]
    },
    {
        tag_name: 'zen',
        tag_description: 'Pra quem curte uma programação mais tranquila e relaxada',
        foursquareCategoriesID: ["52e81612bcbc57f1066b7a27","52e81612bcbc57f1066b7a28",
            "4bf58dd8d48988d1e2941735","4bf58dd8d48988d163941735","4d954b16a243a5684b65b473","4bf58dd8d48988d162941735",
            "4bf58dd8d48988d166941735","4bf58dd8d48988d12f941735","4bf58dd8d48988d131941735"],

        tiers: [1,2]
    },
    {
        tag_name: 'outdoor',
        tag_description: 'Pra quem curte uma programação ao ar livre',
        foursquareCategoriesID: ["52e81612bcbc57f1066b79ee","5267e4d8e4b0ec79466e48c5",
            "53e0feef498e5aac066fd8a9","4bf58dd8d48988d162941735"],
                        # Street Art, Street Fair, Street Food Gathering, , Other Great Outdoors
        tiers: [1,2,3]
    },
```

Figura 24 – *Mapping Tags x foursquareCategoriesID*(1)

4.5.1. Lógica

A lógica de geração de roteiros se utiliza dos seguintes conceitos para a geração de roteiros personalizados:

- *Data Mining*: os dados são obtidos do API do *Foursquare* no momento da requisição; o algoritmo extrai as informações a respeito dos eventos e realiza uma verificação de consistência para campos nulos ou informações inválidas.
- *Data Analytics*: através do DataMining, obtém-se os dados a respeito da utilização de usuários, comentários e frequência de *check-ins* nessa *venue.* Um *check-in* é quando uma *venue* é visitada por um usuário logado na aplicação que executa essa funcionalidade. Com esses dados foi elaborado um algoritmo que determina a relevância de uma *venue* para o contexto em que o usuário se encontra e seu perfil.

```
{
    tag_name: 'museum',
    tag_description: 'Pra quem faz questão de visitar os melhores museus',
    foursquareCategoriesID: ["4bf58dd8d48988d1e2931735","4bf58dd8d48988d181941735"],
                            # Art Gallery, Museum
    tiers: [1,2]

},
{
    tag_name: 'art',
    tag_description: 'Pra quem curte um teatro e faz questão de assistir uma boa peça',
    foursquareCategoriesID: ["4bf58dd8d48988d1e2931735","507c8c4091d498d9fc8c67a9","52e81612bcbc57f1066b79ee",
        "4bf58dd8d48988d1e5931735"],
                            # Art Gallery, Public Art, Street Art, Music Venue
    tiers: [1,2]
},
{
    tag_name: 'entertainment_and_events',
    tag_description: 'Pra quem curte parques de diversao ou eventos para se divertir',
    foursquareCategoriesID: ["4bf58dd8d48988d182941735","5267e4d8e4b0ec79466e48c5",
        "4bf58dd8d48988d1f7941735"],
                            # Theme Park, Street Fair, FleaMarket
    tiers: [1,2,3]
},
{
    tag_name: 'historical',
    tag_description: 'Pra quem curte visitar monumentos e predios historicos',
    foursquareCategoriesID: ["52e81612bcbc57f1066b7a14","4deefb944765f83613cdba6e",
        "4bf58dd8d48988d129941735","4bf58dd8d48988d12d941735","52e81612bcbc57f1066b7a38",
        "52e81612bcbc57f1066b79ed"],
                            # Palace, Historic Site, City Hall, Monument / Landmark, Town Hall, Outdoor Sculpture
    tiers: [1,2,3]
},
```

Figura 25 – *Mapping Tags x foursquareCategoriesID*(2)

```
{
    tag_name: 'food and drink shop',
    tag_description: 'Pra quem gosta de ......',
    foursquareCategoriesID: ["4bf58dd8d48988d120951735","53e0feef498e5aac066fd8a9",
        "4bf58dd8d48988d10e951735","5370f356bcbc57f1066c94c2","4bf58dd8d48988d186941735",
        "50be8ee891d4fa8dcc7199a7","4bf58dd8d48988d1f9941735"]
                            # Food Court, Street Food Gathering, Fish Market, Beer Store, Liquor Store, Market,
                            #Food & Drink Shop
},
{
    tag_name: 'souvenir',
    tag_description: 'Pra quem curte comprar um presente ou uma lembranca da viagem',
    foursquareCategoriesID: ["52f2ab2ebcbc57f1066b8b1b","4bf58dd8d48988d128951735",
        "4bf58dd8d48988d1f7941735"],
                            # Souvenir Shop, Gift Shop, FleaMarket
    tiers: [1,2,3]
},
{
    tag_name: 'theatre',
    tag_description: 'Pra quem curte um teatro e faz questão de assistir uma boa peça',

    foursquareCategoriesID: ["4bf58dd8d48988d1f2931735","4bf58dd8d48988d135941735",
        "4bf58dd8d48988d136941735","4bf58dd8d48988d137941735"],
                            # Performing Arts Venue, Indie Theater, Opera House, Theater
    tiers: [3]
},
{
    tag_name: 'cinema',
    tag_description: 'Pra quem curte um cinema e faz questão de assistir um bom filme',
    foursquareCategoriesID: ["4bf58dd8d48988d17f941735"],
                            # Movie Theater
    tiers: [1,2,3]
},
```

Figura 26 – *Mapping Tags x foursquareCategoriesID*(3)

- *Location-awareness:* com a informação sobre a posição do evento que o usuário visitará, é possível determinar, em um determinado raio, quais os eventos mais adequados.

O algoritmo de geração de roteiros constrói uma pilha de eventos para cada dia do roteiro definido pelo usuário. Se o usuário não estiver logado, o roteiro gerado estabelece como parâmetro a relevância e *feedback* do evento que é calculado a partir de dados estatísticos do API do *Foursquare*. A Figura 27 apresenta a relação de *Cities* populadas no banco de dados do Atairu, para utilização em testes.

```
City.create([{ name: 'Roma', country: 'Itália', lat: "41.872117", long: "12.479095"},
    { name: 'Paris', country: 'França', lat: "48.856583", long: "2.349014"},
    { name: 'Rio de Janeiro', country: 'Brasil', lat: "-22.90812", long: "-43.198414"},
    { name: 'Lisboa', country: 'Portugal', lat: "38.719805", long: "-9.148865"},
    { name: 'Madrid', country: 'Espanha', lat: "40.411405", long: "-3.703423"},
    { name: 'Londres', country: 'Inglaterra', lat: "51.506392", long: "-0.128059"}
])
```

Figura 27 - cidades disponíveis no protótipo inicial

Se o usuário estiver logado, o algoritmo de geração de roteiros considera as informações sobre suas preferências mapeadas através de *tags* e calcula um roteiro mais adequado para o seu perfil.

Além disso, com as informações da posição geográfica dos eventos, é disponibilizada uma previsão do tempo para cada dia do itinerário. Os dados para esta previsão são extraídos no momento da geração do algoritmo de roteiros através do API *Forecast IO*.

4.6. *Chat Server* e funcionalidade de *chat*

O Atairu oferece uma ferramenta de *chat* simples para facilitar a comunicação entre seus usuários. O *chat server* está alojado em um servidor próprio e dedicado somente a este recurso.

Para o desenvolvimento do servidor de *chat* do Atairu foi utilizado o *framework* de servidor de mensagens Faye. O Faye é um sistema de *messaging* do tipo *publishing-subscribe* criado por James Colgan e distribuído sob o tipo de licença MIT (ROSEN, 2004). O Faye provê um servidor de mensagens que realiza o gerenciamento de canais e redirecionamento das mensagens recebidas e enviadas.

Sistemas de *messaging* do tipo *publishing-subscribe* operam através de canais. Um canal é o meio pelo qual todas as mensagens são enviadas e recebidas pelos usuários que estão subscritos a este canal. Um usuário subscrito a um canal passa a receber as mensagens deste canal e passam a ter permissão para enviar

mensagens a este canal. Para a implementação do Atairu, apenas canais bilaterais são permitidos, ou seja, as conversas de *chat* ocorrem entre dois usuários, não se permitindo *chats* em grupo.

O histórico de mensagens é recuperado toda vez que um usuário é subscrito a um canal. Como o Atairu implementa sistema de mensagens dois a dois esse evento acontece quando é inicializada uma conversa de *chat*. Esta funcionalidade permite o envio de mensagens mesmo que um dos usuários esta *offline*.

4.7. Algoritmo de *Matching*

O *Matching* é uma funcionalidade do Atairu que permite aos usuários cadastrados a busca e contato de outros usuários que tenham viagens que coincidam em um ao mais dias e que serão realizados na mesma cidade. É necessária uma permissão do usuário para que ele seja listado como potencial contato de outros usuários, através desse mecanismo de busca.

4.8. *Backlog* de Implementação do *Backend*

Para o desenvolvimento da API de *web services* do Atairu, foi construído um *backlog*, listando todos os serviços a serem implementados na API, juntamente com a descrição de suas funcionalidades. Por meio do gerenciamento da implementação destes serviços em um documento compartilhado entre os integrantes do grupo, foi possível acompanhar a execução destas atividades.

A relação de atividades desenvolvidas no processo de implementação da API de *RESTfulServices* do Atairu é apresentado a serguir.

1. **Estudo Web Services:** Com o intuito de desenvolver uma API de *web services*, foi verificado que a arquitetura REST era a mais adequada para este sistema, tendo como base a utilização dos vários métodos dos recursos alocados na aplicação lógica.
2. **Web Service Users - Login:** Realiza a autenticação do usuário no sistema.
3. **Web Service Users - Logout:** Verifica se o usuário está logado no sistema, realizando um *check* duplo, através do seu e-mail passado como parâmetro no corpo da requisição e do *token* de autenticação

passado no *header*. Se o usuário existe e o *token* de autenticação está *ok*, realiza o *logout* do usuário no sistema.

4. **Web Service Users - Cadastro de Usuário:** Cria um novo usuário no sistema e logo em seguida o autentica no sistema.
5. **Web Service Users - Pegar Dados do Usuário:** Retorna os dados do usuário.
6. **Web Service Users - Atualizar Cadastro:** Atualiza o cadastro de um usuário.
7. **Web Service Users - Facebook id:** Verifica se já existe algum usuário registrado com o *facebook_id* enviado. Se sim, retorna o usuário, se não, envia uma mensagem de erro.
8. **Web Service Users - Busca de Usuários:** Busca usuários pelo nome ou sobrenome ou e-mail. A identificação do e-mail é automática, pela presença do '@'.
9. **Web Service Users - Retornar Dados de um Usuário:** Retorna os dados de um usuário e um booleano *isFriend*. *isFriend* é *true* se o usuário buscado é amigo do requisitante e *false* caso contrário.
10. **Web Service Tags - Listar Tags:** Retorna todas as tags relacionadas ao usuário.
11. **Web Service Tags - Criar Tag:** Cria uma associação entre uma *tag* e um *user*.
12. **Web Service Tags - Exluir Tag:** Exclui uma associação entre uma *tag* e um *user*.
13. **Web Service Rotas - Listar Rotas:** Retorna todas as rotas associadas ao usuário logado.
14. **Web Service Rotas - Criar Rota:** Cria uma rota com os atributos *name* e *description*.
15. **Web Service Rotas - Retornar Rota:** Retorna rota especificada pelo identificador.
16. **Web Service Rotas - Excluir Rota:** Exclui uma rota do sistema.
17. **Web Service Rotas - Adicionar roteiro:** Adiciona um roteiro (*Itinerary)* a uma rota, especificados por um indentificador.

18. **Web Service Rotas - Excluir roteiro:** Remove uma associação entre um roteiro (*Itinerary)* e uma rota.
19. **Web Service Rotas - Adicionar Usuário:** Cria uma associação entre um usuário e uma rota.
20. **Web Service Rotas - Excluir Usuário:** Exclui uma associação entre um usuário e uma rota.
21. **Web Service Rotas - *Match* de Rota:** Verifica a existência de rotas que possuam ao menos uma data coincidente e retorna a lista de rotas (*Matchs)* e roteiros (*Itineraries*) relacionados.
22. **Web Service Rotas - Retornar Usuários Associados à Rota:** Retorna lista de usuários associados a uma determinada rota e o id do seu administrador.
23. **Web Service Roteiros - Pegar Roteiros:** Retorna roteiro (*Itinerary)* especificado pelo identificador. Observação: este roteiro (*Itinerary)* deve, obrigatoriamente, pertencer a uma rota que está associada ao usuário.
24. **Web Service Roteiros - Excluir Roteiro:** Exclui um roteiro (*Itinerary).*
25. **Web Service Roteiros - Gerar Roteiro:** Retorna um roteiro (*Itinerary)* a partir dos seguintes parâmetros: cidade, datas que compreendem a estadia do usuário na cidade e interesses do usuário.
26. **Web Service Roteiros - Listar Eventos:** Retorna um *array* com todos os Eventos que estão associados ao roteiro (*Itinerary)* especificado pelo identificador.
27. **Web Service Roteiros - Atualizar Roteiro:** Atualiza roteiro (*Itinerary)* especificado pelo identificador.
28. **Web Service Eventos - Listar Eventos:** Retorna lista de eventos cadastrados para uma cidade.
29. **Web Service Eventos - Adicionar Evento no Roteiro:** Adiciona um evento a um roteiro (*Itinerary).*
30. **Web Service Eventos - Excluir Evento no Roteiro:** Exclui um evento que pertence a um roteiro (*Itinerary).*

31. **Web Service Eventos - Atualizar Evento do Roteiro:** Atualiza um evento que pertence a um roteiro (*Itinerary)*.
32. **Web Service Perfil - Criar perfil usuário:** Cria o perfil completo de um usuário logado.
33. **Web Service Perfil - Pegar Perfil Usuário:** Retorna o perfil completo de um usuário especificado pelo identificador.
34. **Web Service Perfil - Atualiza Perfil Usuário:** Atualiza o perfil do usuário logado, retornando o perfil atualizado.
35. **Web Service Perfil - Pegar Perfil Simples Usuário:** Retorna um perfil simplificado de um usuário especificado pelo identificador.
36. **Web Service Amigos - Listar Amigos:** Retorna a lista de amigos de um usuário logado no sistema.
37. **Web Service Amigos - Listar usuários que o tem como amigo:** Retorna a lista de *users* que possuem o usuário logado na sua lista de amigos.
38. **Web Service Amigos - Adicionar Amigo:** Cria uma relação de amizade entre o usuário especificado pelo identificador e o usuário logado no sistema.
39. **Web Service Amigos - Excluir Amigo:** Exclui um usuário especificado pelo identificador da lista de amigos do usuário logado no sistema.
40. **Web Service Configuração - Lista Configuração:** Retorna uma lista das configurações (*Settings)* do usuário.
41. **Web Service Configuração - Atualizar Configurações:** Atualiza as configurações do aplicativo de um usuário logado no sistema.
42. **Web Service Mensagens - Recuperar Mensagens:** Recupera todas as mensagens enviadas pelos usuários em um determinado canal.
43. **Web Service Mensagens - Armazenar Mensagens:** Armazena a mensagem enviada pelo usuário a um determinado canal.

4.9. Resultados de Implementação do *Backend*

A API de *web services* do Atairu fornece métodos para obter e manipular os recursos do sistema, que podem ser, usuários, rotas, roteiros, eventos, perfil de

usuário, amigos, configurações e mensagens. A Figura 28 apresenta os recursos e os seus métodos.

Recurso	Método
Users - C.1	login
	logout
	register
	get-user
	update-user-data
	has-facebook-id
	search-users
	get-user-data
Tags - C.2	get-tags
	create-tag
	delete-tag
Rotas - C.3	get-routes
	get-route
	destroy-route
	create-route
	match-route
	add-itinerary
	delete-itinerary
	add-user
	delete-user
	get-route-users
Roteiros/Itinerarie s - C.4	get-itinerary
	destroy-itinerary
	generate-itinerary
	list-events
	update-itinerary
Eventos - C.5	add-event
	remove-event
	update-event
Perfil - C.6	create-profile
	update-profile
	get-profile
Amigos - C.7	get-friends
	get-inverse-friends
	request-friendship
	destroy-friendship
Configuração - C.8	get-settings
	update-settings
Mensagens - C.9	retrieve-messages
	store-message

Figura 28 – Recursos e métodos

A especificação dos *web services* (descrição, parâmetros, mensagens de sucesso, mensagens de erro) do Atairu estão descritos em APÊNDICE D – DESCRIÇÃO DO API DO ATAIRU.

4.10. Testes

Os testes foram realizados iterativamente ao longo do projeto em, principalmente, 3 etapas:

1. Realização de testes locais isolados de *backend* e *frontend:* localmente foram realizados testes de casos de sucesso e de erro após a implementação de cada funcionalidade. A biblioteca RSPEC que facilita a descrição e execução de testes foi utilizada durante a implementação

das funcionalidades da aplicação lógica, para confirmar as respostas e as mensagens de erro esperadas (VIEIRA, 2014).

2. Realização de testes locais integrados: após o sucesso na verificação dos testes locais isolados, foram realizados testes integrados entre o *backend* e o *frontend*. A aplicação *frontend* realizava uma requisição de um *web service* e o *backend* devolvia a resposta esperada ou uma mensagem de erro.
3. Realização de teste entre *frontend* e servidor de *Web Services:* o sucesso na execução dos testes integrados locais leva ao *deploy* da aplicação lógica ao servidor de *web services.* Após o *deploy*, foiram executados testes funcionais e não-funcionais, levando em conta principalmente o tempo de resposta entre a aplicação *frontend* e o servidor de *Web Services.*

A execução desses testes, de forma iterativa e incremental durante a etapa de desenvolvimento do sistema, permitiu uma baixa taxa de retrabalho e uma alta correspondência aos resultados esperados.

4.11. Considerações Finais do Capítulo

A implementação descrita nas seções anteriores permitem afirmar a construção de um sistema que cumpre a especificação descrita no Capítulo 2 – Especificação do Sistema. O padrão de arquitetura REST, utilizada na implementação dos *web services*, contribuiu para o desenvolvimento de uma arquitetura distribuída e de baixo acoplamento. Isto permitirá que outros sistemas possam se conectar à API do Atairu, como no caso do desenvolvimento de uma aplicação móvel com a tecnologia Android para futuros trabalhos.

5. CONSIDERAÇÕES FINAIS

5.1. Conclusão

A realização deste trabalho permitiu o aprendizado de conceitos de arquitetura de sistemas *web services* como SOA e RESTful API. O primeiro conceito foi abordado no decorrer do curso, porém a utilização da teoria na prática se mostrou mais clara neste projeto. No caso do segundo conceito, foi a primeira vez que o grupo se deparou com este modelo de arquitetura; porém sua aprendizagem e aplicação neste projeto se mostraram bastante colaborativa.

O benefício obtido no aprofundamento do conhecimento do grupo se deve ao maior tempo de dedicação, iniciado em Janeiro de 2015 e finalizado em Dezembro de 2015, e pelo gerenciamento de projeto feito por Márcio Carlos Perin Tedesco.

Os cursos de Engenharia de Software foram essenciais para este trabalho, fornecendo as capacidades necessárias para a especificação do sistema Atairu e enriquecendo as experiências do grupo com relação ao ciclo de planejamento e desenvolvimento de um software.

O grupo acredita que este sistema será de grande ajuda para os viajantes e irá tornar o planejamento e a própria viagem mais agradável. Para tanto, o grupo pretende continuar a agregar mais valor ao sistema Atairu, com o propósito de levar ao mercado o que há de melhor para uma rede social para viajantes. Há muito trabalho a ser feito e desafios a serem conquistados para que este sistema se torne uma referência para os viajantes do mundo, mas as habilidades obtidas ao decorrer do curso de Engenharia da Computação será a chave para vencer esta nova etapa.

5.2. Contribuições

Este trabalho teve como objetivo desenvolver um sistema com a capacidade de auxiliar os viajantes a obterem roteiros de uma forma mais fácil, por meio do algoritmo de sugestão de roteiros personalizada, e a conhecerem novas pessoas, por meio do *chat* e do sistema de *matching*. Outros sistemas como o TripAdvisor e o Gogobot tentam ajudar seus usuários a organizarem suas viagens, porém não tem a junção de funcionalidades que o sistema Atairu possui. O primeiro sistema contém uma base de dados mais completa, sendo a mais popular do mercado em relação a

informações de lugares, e a segunda ajuda a montar um roteiro de viagens, porém apresenta baixa usabilidade e uma interface não intuitiva.

A grande contribuição deste trabalho é o algoritmo de sugestão de roteiros, que é a principal funcionalidade. Porém, sem as outras funcionalidades, que permitem a troca de experiências entre as pessoas, não seria possível ter um sistema completo. Tudo isto envolveu a análise de tecnologias atuais como linguagens de programação de aplicação móvel (iOS) e lógica (Ruby), gerenciador de banco de dados (PostgreSQL), *framework* (Rails) e *PaaS* (Heroku), a análise de conceitos de arquitetura voltada a *web services* como o SOA e REST e conceitos relacionados ao gerenciamento da informação como *context-aware*, *location-aware, proximate selection.*

Este novo conceito de sistema voltado para viagens certamente será o propulsor de ideias inovadoras, simples e poderosas, sendo uma contribuição importante para que novas ideias possam surgir por meio deste trabalho.

5.3. Trabalhos Futuros

O desenvolvimento deste trabalho mostrou-se bastante satisfatório, porém não foi possível implementar a funcionalidade *gamification* por conta dos prazos. Esta funcionalidade se mostra bastante promissora para atrair mais usuários ao sistema e para melhorar a contribuição das informações em tempo real entre os viajantes, visto que os usuários poderiam enviar um *feedback* de uma *venue* sem violar as condições de uso da API do Foursquare.

Uma melhoria a ser feita de tempos em tempos é na funcionalidade sugestão de roteiros, com o objetivo de sempre aprimorar e se adequar à necessidade dos usuários. A implementação de um algoritmo de *Machine Learning* que possa identificar padrões de preferências pode complementar esta funcionalidade e, para isto, será necessário um estudo mais profundo quanto aos dados a serem considerados.

Outras modificações a serem implementadas que foram identificadas são:

- Adicionar paginação ao sistema de envio de mensagens (*chat*) e adição de *emoticons*;

- Desenvolver novos mecanismos para realizar a consistência dos dados com relação aos eventos;
- Desenvolver uma *engine* para obtenção de dados relacionados aos eventos de maneira mais independente da API do *Foursquare*;
- Desenvolver uma versão para Android, com o objetivo de alcançar a maior fatia do *market-share mobile;*
- Especificar e desenvolver o acesso modularizado para promover a comercialização do Atairu;
- Especificar a precificação por módulo. A ideia inicial é permitir o uso gratuito para geração de roteiros que não utilizam as preferências do usuário. Para a utilização das funcionalidades como *matching,* busca de usuários cadastrados e salvar viagens, o usuário deverá comprar o acesso;
- Melhorar quesito de escalabilidade e gerenciamento de histórico do servidor de *chat*
- Implementação de *PushNotifications* para eventos provocados pelo sistema (e.g. nova mensagem de chat recebida).

REFERÊNCIAS

BEZERRA E. O Processo de Desenvolvimento de Software. In:_____. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro: ELSEVIER, 2006, cap.2, pág. 19-32

DREDGE, S. *If Android is so popular, why are many apps still released for iOS first?*2013. Disponível em: <http://www.theguardian.com/technology/appsblog/2013/aug/15/android-v-ios-apps-apple-google>. Acesso em: 3 de setembro de 2014.

EVANS, J. *Android vs. iOS Development: Fight!* 2013. Disponível em: <http://techcrunch.com/2013/11/16/the-state-of-the-art/>. Acesso em: 3 de setembro de 2014.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. Tese de Doutorado, University of California, Irvine, 2000

FUGITA, H. S.; HIRAMA, K. Entendendo SOA. In:_____. SOA: modelagem, análise e design. Rio de Janeiro: Elsevier, 2012, cap 2, p. 7-40.

HARTL, M.; PROCHAZKA, A. *Introduction: Why Rails?* In: _____. RailsSpace: Building a Social Networking Website with Ruby on Rails. Crawfordsville: RR Donelley, 2008, cap 1.

IND., F. Facebook Social Network. 2014. Disponível em: http://www.facebook.com. Acesso em: 30 de novembro de 2014.

IND., F. Forecast IO. 2014. Disponível em: https://forecast.io/ Acesso em: 30 de novembro de 2014.

IND., F. Foursquare. 2014. Disponível em: https://pt.foursquare.com/ Acesso em: 30 de novembro de 2014.

IND., G. Gogobot. 2014. Disponível em: http://www.gogobot.com. Acesso em: 30 de novembro de 2014.

IND., H. Heroku Cloud Application Plataform. 2014. Disponível em: http://www.heroku.com. Acesso em: 30 de novembro de 2014.

IND., T. TripAdvisor LLC. 2014. Disponível em: http://www.tripadvisor.com. Acesso em: 30 de novembro de 2014.

ROSEN, L.; OPEN SOURCE LICENSING, Prentice Hall PTR, 1st ed. 2004, p. 85.

LEVANDOSKI, J. J. et al. (2012, abril) LARS: A Location-Aware Recommender System. Microsoft Research.

OASIS *SOA REFERENCE MODEL TD.Reference Model for Service for Service Oriented Architecture* 1.0. OASIS Open, 2006. 31 p.

ORT, E. *Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools* 2005 Disponível em: http://www.oracle.com/technetwork/articles/javase/soaterms-138190.html. Acesso em: 4 de novembro de 2014.

PLASNKY, R. Definição, restrições e benefícios do modelo de arquitetura REST 2014. Disponível em: <http://imasters.com.br/desenvolvimento/definicao-restricoes-e-beneficios-modelo-de-arquitetura-rest/>. Acesso em: 24 de novembro de 2014.

SARACUT, F. *Should You Develop for iOS or Android First? Discover What 7 Mobile App Experts Say.* 2013. Disponível em: <http://blog.mobiversal.com/ios-or-android-first.html>. Acesso em: 3 de setembro de 2014

SCHILIT, B.; ADAMS, N.; WANT, R. (1994, Dezembro). Context-Aware Computing Applications. Santa Cruz, CA: IEEE, 1994, p. 85-90.

TEZER, O. S. *SQLite vs MySQL vs PostgreSQL: A Comparison of Relational Database Management Sytems* 2014. Disponível em: <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>. Acesso em: 24 de novembro de 2014.

VIEIRA, D. Scrum: A Metodologia Ágil Explicada de uma forma Definitiva 2004 Disponivel em: <http://www.mindmaster.com.br/scrum/> Acessado em: 30 de novembro de 2014

GLOSSÁRIO

Evento: engloba tanto as atrações como museus, teatros e parques, como festas, festivais, mostras etc.

Roteiro do dia: é a lista de eventos sugeridos pelo sistema, ou escolhidos pelo usuário, a serem percorridos durante um dia um uma determinada cidade.

Rota: conjunto de cidades as quais o usuário pretende visitar.

Roteiro: é o conjunto de roteiros do dia para o período de estadia na cidade. Cada cidade que faz parte da rota de um usuário possui um roteiro com um ou mais roteiros do dia.

***Web Crawler*:** sistema que faz buscas pela internet com o propósito de juntar dados ao sistema.

Deploy:é a instalação de uma aplicação em um servidor de aplicações.

APÊNDICE A – DESCRICÃO DE CASOS DE USO

Dentre os casos de uso listados na seção 2.5.1, foram selecionados os mais relevantes para o funcionamento do Aitaru e são descritos em maior detalhe. São eles:

- UC3 - Gerar sugestão de roteiros
- UC16 - Editar rota
- UC21 - Adicionar contato à rota
- UC22 - Executar *Matching*
- UC23 - Adicionar um usuário à lista de amigos

Caso de Uso UC3: Gerar sugestão de roteiros
Descrição: A partir da Tela 2 - Início, o usuário seleciona filtros de acordo com os seus interesses, uma data de chegada à cidade, a cidade e o número de dias que ficará na cidade. Ao pressionar o botão Buscar o sistema retorna em outra tela, Tela 3 - Roteiros, os roteiros prontos organizados por dia.
Evento Iniciador: Usuário requisita sugestão de roteiros.
Atores: Usuário logado ou usuário deslogado.
Pré-condição: Usuário estar na tela 2 - Início
Sequência de Eventos:

1. Usuário seleciona as atividades que tem interesse através de um ou mais filtros listados na tela.
2. Sistema informa que o filtro foi selecionado.
3. Usuário clica no botão Cidades Disponíveis.
4. Sistema retorna a lista de cidades disponíveis.
5. Usuário seleciona a cidade desejada.
6. Usuário seleciona a quantidade de dias de estadia na cidade selecionada
7. Usuário clica no botão Buscar.
8. Sistema realiza a busca e retorna uma lista de roteiros para os dias de estadia para a cidade selecionada.

Pós-Condição: roteiros para os dias de estadia do usuário na cidade desejada disponíveis na tela (Tela 3)

Extensões: não há.

Inclusões: não há.

Caso de Uso UC16: Adicionar roteiro a partir de uma rota.

Descrição: Neste caso de uso é descrito o processo de adição de um roteiro a uma rota.

Evento Iniciador: Usuário seleciona adicionar roteiro a uma rota.

Atores: Usuário Logado.

Pré-condição: Usuário estar na tela 10 - Gerenciamento de Rota.

Sequência de Eventos:

1. Usuário seleciona o botão + de adição de roteiro.
2. Sistema apresenta a tela 2 - Início.
3. Usuário requisita sugestão de roteiros (Caso de Uso - Gerar sugestão de roteiro)
4. Sistema apresenta a Tela 3 - Roteiros.
5. Usuário seleciona o botão Salvar Roteiro.
6. Sistema apresenta a tela 10 – (nome da tela) com o roteiro adicionado à rota.

Pós-Condição: Informações de rota editadas e salvas pelo sistema. O sistema apresenta as novas informações adicionadas na tela 10 - Gerenciamento de Rota.

Extensões: não há.

Inclusões:

1. Caso de Uso 3 - Gerar sugestão de roteiro (passo 3)

Caso de Uso UC21: Adicionar contato à rota

Descrição: Esse caso de uso descreve o processo de adição de um contato da lista de amigos na rota, permitindo assim que esse contado possa visualizar e editar a rota. O ator que inicia esse caso de uso é o usuário do sistema.

Evento Iniciador: Adicionar contato com o qual vai compartilhar a rota.

Atores: Usuário logado

Pré-condição: Usuário estar na tela 10 - Gerenciamento da rota.

Sequência de Eventos:

1. Usuário seleciona o ícone de adicionar contato.

2. Sistema exibe a lista de contatos disponíveis.
3. Usuário seleciona um ou mais contatos.
4. Usuário clica no botão OK.
5. Sistema dá permissão aos contatos selecionados para visualizar e editar a rota.

Pós-Condição: Rota replicada no perfil dos contatos selecionados e sistema volta para a tela de visualização da rota.

Extensões: Não há.

Inclusões: Não há.

Caso de Uso UC22: Executar *Matching*

Descrição: Esse caso de uso descreve o processo de *matching* da rota do usuário. A partir dos dados das cidades da rota e do período de permanência em cada cidade, o sistema procura algum outro usuário que se apresente na mesma cidade com período de estadia em comum.

Evento Iniciador: Usuário seleciona o botão de *Match*.

Atores: Usuário logado

Pré-condição: O usuário estar na tela 10 - Gerenciamento da Rota.

Sequência de Eventos:

1. Sistema verifica no banco de dados do servidor possíveis usuários que apresentam características semelhantes de viagem.
2. Sistema exibe a lista de usuários que possuam algum *match*, ordenando pela quantidade de *matchs*.
3. Usuário clica no botão Voltar.
4. Sistema retorna lista de *matching*.

Pós-Condição: Resultado do algoritmo de *matching apresentado.*

Extensões:

1. Sistema não encontra nenhum usuário que possua algum *match*, retornando assim, um aviso de não encontrado ninguém e retorna para a tela 10. (Passo 3)
2. Usuário seleciona um ou mais usuário da lista e clica no botão adicionar contato. (Passo 3)

Inclusões: Não há.

Caso de Uso UC23: Adicionar um usuário à lista de amigos

Descrição: Esse caso de uso descreve o processo de adicionar um usuário à lista de amigos, a partir da integração com o Facebook, Twiter ou Gmail e também por sistema de pesquisa através do nome.

Evento Iniciador: Selecionar buscar contatos.

Atores: Usuário logado.

Pré-condição: O usuário estar na tela 7 - Amigos.

Sequência de Eventos:

1. Usuário clica no botão Buscar.
2. Sistema apresenta a tela 16 - Buscar amigos.
3. Sistema busca os contatos do usuário a partir da integração com o Facebook.
4. Sistema exibe a lista de contatos do usuário.
5. Usuário clica no botão + de Adicionar Contato (Caso o contato já tenha o aplicativo)
6. Sistema faz a solicitação de amizade para o amigo.
7. Sistema mostra usuário na lista como pendente.

Pós-Condição: Contato adicionado à lista de amigos

Extensões:

1. Sistema não consegue fazer busca de contatos a partir da integração com outra rede social: Sistema mostra uma lista vazia (Passo 3)
2. Sistema manda uma mensagem ao contato fazendo o convite para utilização do aplicativo, caso ele não esteja cadastrado no aplicativo. (Passo 5)
3. Usuário pesquisa usuários pela barra de busca: sistema retorna a lista de usuários cujo nome contém no texto de busca. (Passo 4)

Inclusões: Não há.

APÊNDICE B – PROTÓTIPOS DE TELAS

Este apêndice apresenta todos os protótipos de telas.

Tela 0 - *Login*

Figura 29 – Tela 0 – Login

Descrição: Essa é a tela de *login* do aplicativo. Através dela, o usuário pode realizar o *login* através de diferentes redes sociais ou clicar no botão Cadastrar para ir à tela de cadastro do usuário no sistema.

Tela 1 - Cadastro

Figura 30 – Tela 1 – Cadastro

Descrição: Essa é a tela de cadastro do aplicativo. Através dela, o usuário realiza um novo cadastro no sistema.

Tela 4 - Adicionar Evento

Figura 31 – Tela 4 – Adicionar Evento

Descrição: Nessa tela, o usuário adiciona o evento ao roteiro clicando no sinal +. Cada evento tem uma descrição com algumas informações relevantes no campo informação, mostrado na figura.

Tela 5 - Mapa do Roteiro

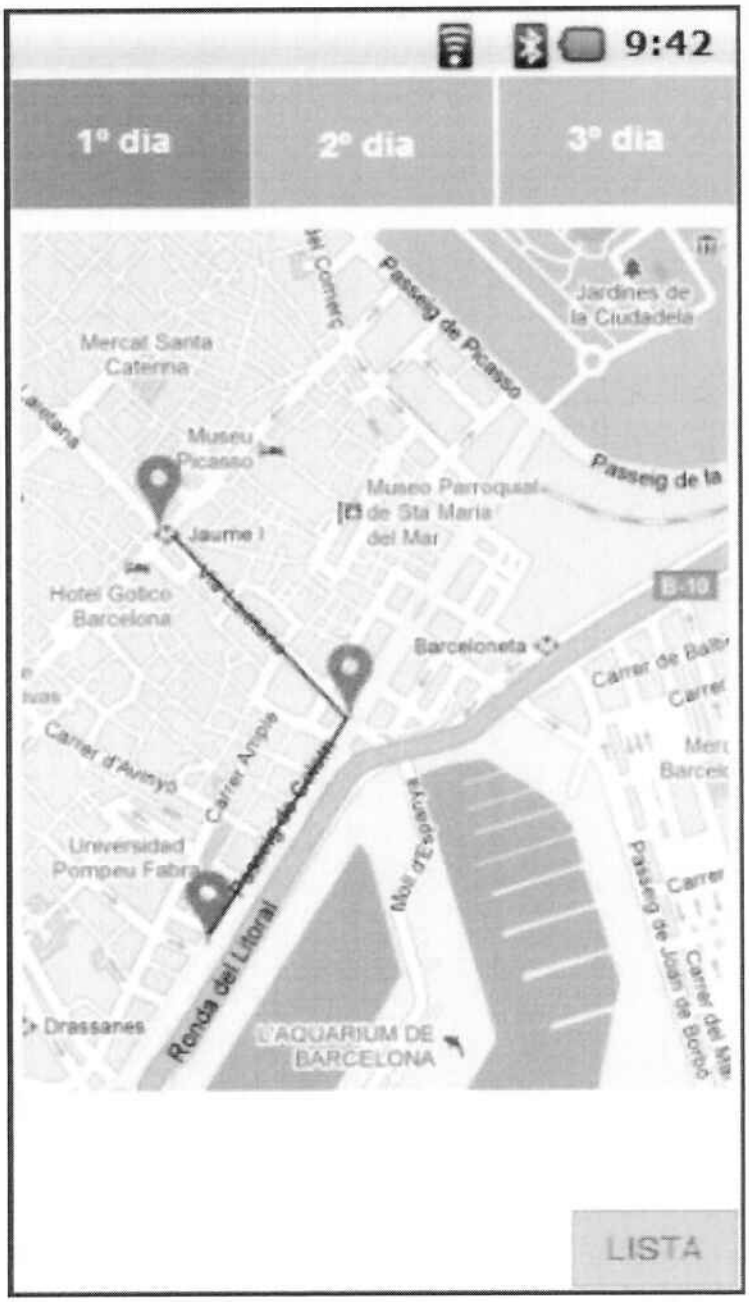

Figura 32 – Tela 5 – Mapa do Roteiro

Descrição: Nessa tela, o usuário visualiza o mapa do roteiro em um determinado dia. Ao clicar no botão Lista, o usuário volta para a tela de visualização do roteiro.

Tela 7 - Amigos

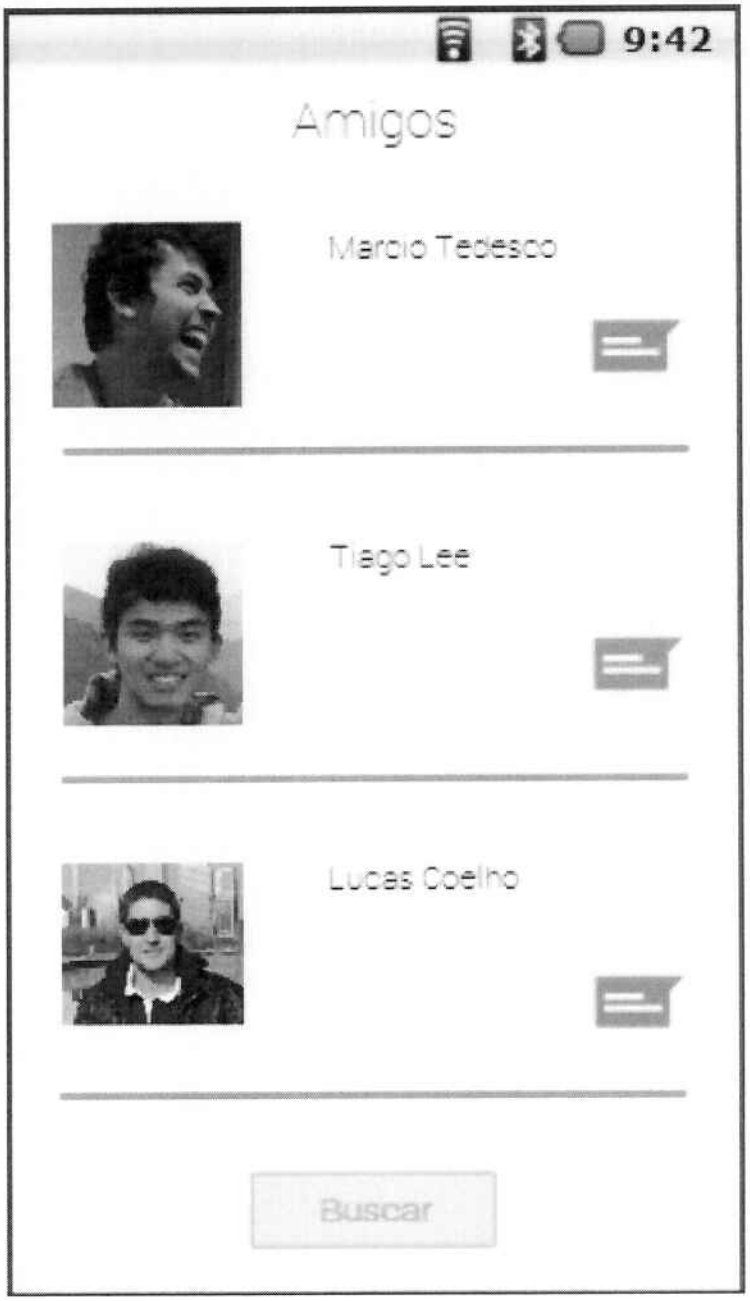

Figura 33 – Tela 7 – Amigos

Descrição: Nessa tela, o sistema mostra os amigos que já foram adicionados na sua lista. Ao clicar no ícone , o sistema vai para a tela de bate papo.

Tela 9 - Rotas

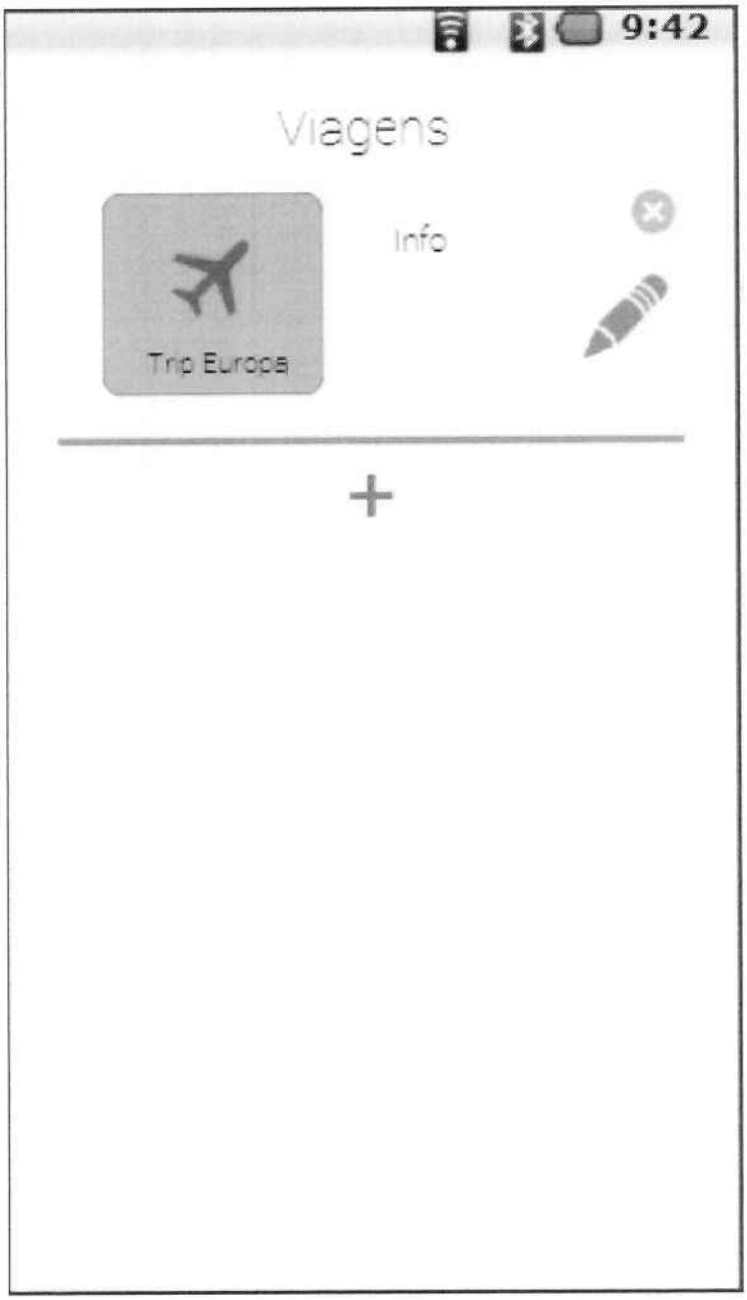

Figura 34 – Tela 9 – Rotas

Descrição: Para efeito de visualização das rotas pelo usuário, é utilizada a palavra viagens para representar as rotas somente na interface de usuário, por ser mais intuitivo a ele. Nessa tela, o usuário visualiza as suas rotas. Pode excluir a viagem associada, ao clicar no botão X, adicionar ao clicar no botão + ou editar ao clicar no botão com a imagem de um lápis.

Tela 11 - Adicionar amigos à rota

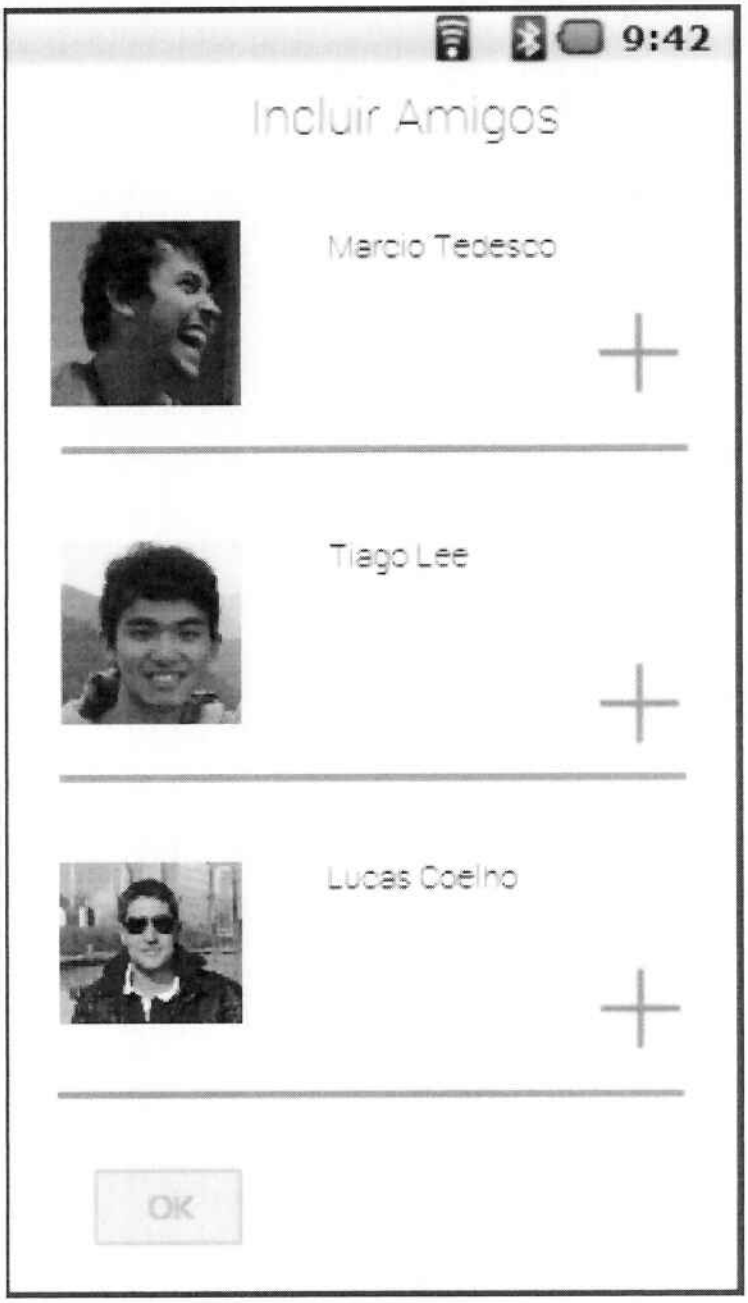

Figura 35 – Tela 11 – Adicionar Amigos à Rota

Descrição: Essa tela lista os amigos já existentes e o usuário pode adicionar ou remover os amigos que compartilham a rota.

Tela 13 - Perfil do usuário selecionado

Figura 36 – Tela 13 – Perfil do Usuário Selecionado

Descrição: Nessa tela, o usuário pode visualizar os detalhes dos roteiros em comum e também um resumo do perfil do outro usuário (Sobre, Interesses e foto).

Tela 15 - Seleção de rota

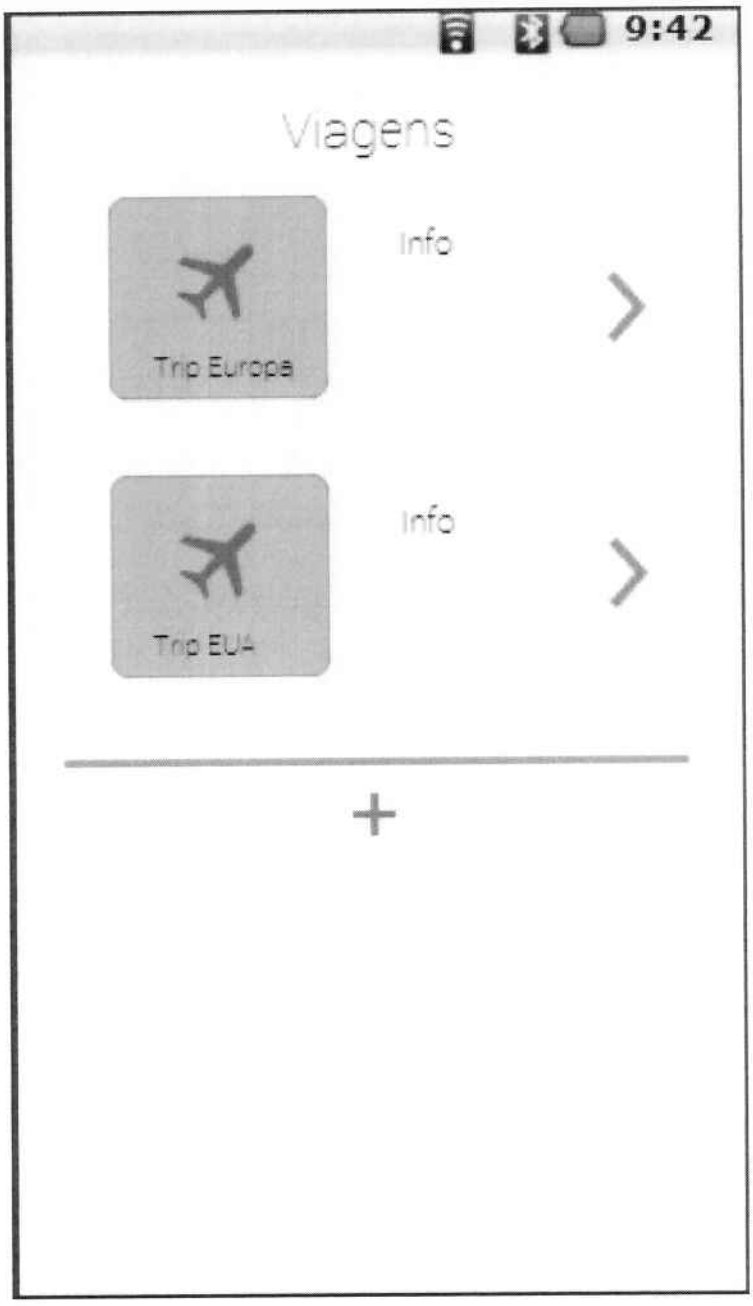

Figura 37 – Tela 15 – Seleção de Rota

Descrição: Essa tela apresenta a lista de rotas já existentes do usuário e a opção de criação de uma nova rota.

Tela 16 - Buscar Amigos

Figura 38 – Tela 16 – Buscar Amigos

Descrição: Nessa tela, o sistema faz integração com o Facebook, listando os amigos, e permite convidá-lo a experimentar o aplicativo, caso ele não possua. Também o usuário pode realizar buscas por nome de amigos.

APÊNDICE C – DIAGRAMA DE CLASSES

Este apêndice apresenta o Diagrama de Classes com os seus métodos e atributos.

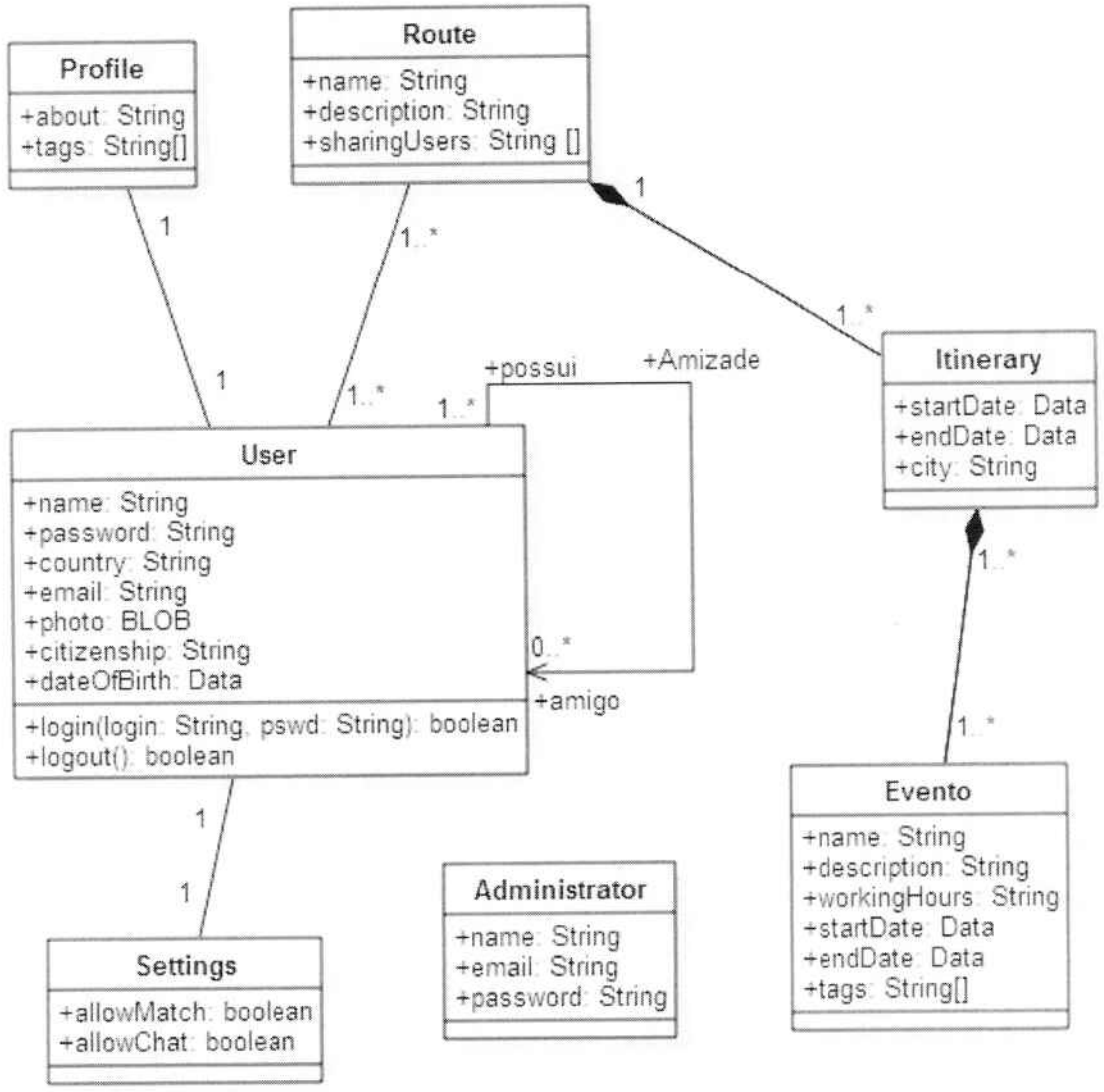

Figura 39 – Diagrama de Classes

APÊNDICE D – DESCRICÃO DO API DO ATAIRU

Esse apêndice apresenta a lista dos *web services* implementados. Para cada serviço, são apresentados a descrição, os parâmetros de entrada, a resposta e, quando for o caso, os dados envolvidos.

D.1 Users

D.1.1 Users / Login

D.1.1.1 Descrição

Realiza a autenticação do usuário no sistema.

Método	**POST**
URL	**/api/v1/login**

D.1.1.2Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
user[password]	O Password não pode passar de 15 caracteres (maxlength=15).	*string*	Obrigatório	character varying(255)

D.1.1.3 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user]	Retorna o usuário recém logado com um authentication_token recém gerado..	*string[]*		
success	Retorna *true*.	*boolean*		
info	Mensagem: *"O usuário foi logado com sucesso."*	*string*		

D.1.1.3Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: *"E-mail ou authenticaion token inválidos. Você tem certeza que está logado?"*	*string*		

D.1.2 Users / Logout

D.1.2.1 Descrição

Verifica se o usuário está logado no sistema, realizando um *check* duplo através do seu e-mail passado como parâmetro no corpo da requisição e do *token* de autenticação passado no *header*. Se o usuário existe e o *token* de autenticação está *ok*, realiza o *logout* do usuário no sistema.

Método	**POST**
URL	**/api/v1/logout**

D.1.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.1.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o login	*string*	Obrigatório	20

D.1.2.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		
info	Mensagem: *"O usuário foi deslogado com sucesso."*	*string*		

D.1.2.5 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: *"E-mail ou authentication token inválidos. Você tem certeza que está logado?"*	*string*		

D.1.3 Users / Cadastro de Usuário

D.1.3.1 Descrição

Cria um novo usuário no sistema e logo em seguida o autentica no sistema.

Método	**POST**
URL	**/api/v1/register**

D.1.3.2Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[first_name]	Primeiro nome do usuário.	*string*	Opcional	character varying(255)
user[last_name]	Último nome do usuário.	*string*	Opcional	character varying(255)
user[gender]	Valores possíveis: *female* ou *male*.	*enum*	Opcional	-
user[country]	País do usuário.	*string*	Opcional	character varying(255)
user[citizenship]	Cidadania do usuário.	*string*	Opcional	character varying(255)
user[dateOfBirth]	Data de Nascimento do usuário no formato ISO6801: YYYY-MM-DD	*date*	Opcional	10
user[password]	Senha do usuário	*string*	Obrigatório	character varying(255)
user[password_confirmation]	Confirmação de senha do usuário.	*string*	Obrigatório	character varying(255)
user[email]	E-mail do usuário	*string*	Obrigatório	character varying(255)
user[photoURL]	URL para a foto do usuário	*string*	Opcional	character varying(255)
user[facebook_id]	Id facebook do usuário para login no facebook	*string*	Opcional	character varying(255)

D.1.3.3 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user]	Retorna o usuário recém cadastrado.	*string*		
success	Retorna *true*.	*boolean*		
info	Mensagem:"*O usuário foi cadastrado com sucesso*"	*string*		

D.1.3.4 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Retorna os erros que surgiram durante o processo de registro.	*string*		

D.1.4 Users / Pegar Dados do Usuário

D.1.4.1 Descrição

Retorna os dados do usuário.

Método	**POST**
URL	**/api/v1/get-user**

D.1.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.1.4.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.1.4.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user][id]	Retorna id do usuário na tabela.	*integer*		
data[user][first_name]	Primeiro nome do usuário.	*string*		character varying(255)
data[user][last_name]	Último nome do usuário.	*string*		character varying(255)
data[user][gender]	Valores possíveis: *female* ou *male*.	*enum*		-
data[user][country]	País do usuário.	*string*		character varying(255)
data[user][citizenship]	Cidadania do usuário.	*string*		character varying(255)
data[user][dateOfBirth]	Data de Nascimento do usuário no formato ISO6801: YYYY-MM-DD	*date*		10
data[user][photoURL]	URL para a foto do usuário	*string*		
success	Retorna *true*	*boolean*		

D.1.4.5 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: "*E-mail ou authenticaion token inválidos. Você tem certeza que está logado?*"	*string*		

D.1.5 Users / Atualizar Cadastro

D.1.5.1 Descrição

Atualiza o cadastro de um usuário.

Método	**POST**
URL	**/api/v1/update-user-data**

D.1.5.2Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[first_name]	Primeiro nome do usuário.	*string*	Opcional	character varying(255)
user[last_name]	Último nome do usuário.	*string*	Opcional	character varying(255)
user[gender]	Valores possíveis: *female* ou *male*.	*enum*	Opcional	-
user[country]	País do usuário.	*string*	Opcional	character varying(255)
user[citizenship]	Cidadania do usuário.	*string*	Opcional	character varying(255)
user[dateOfBirth]	Data de Nascimento do usuário no formato ISO6801: YYYY-MM-DD	*date*	Opcional	10
user[email]	E-mail do usuário	*string*	Obrigatório	character varying(255)
user[photoURL]	URL para a foto do usuário	*string*	Opcional	

D.1.5.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.1.5.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		
info	Mensagem: "*Os detalhes de User which e-mail is user1@example.com foram atualizados com sucesso.*"	*string*		

D.1.5.5 Resposta - Erro - HTTP 422

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*.	*boolean*		
info	Mensagem: "*Tentativa de atualização inválida.*"	*string*		

D.1.6 Users / Verificar Registro do Facebook Id

D.1.6.1 Descrição

Verifica se já existe algum usuário registrado com o facebook_id enviado. Se sim, retorna o usuário, se não, envia uma mensagem de erro.

Método	**GET**
URL	**/api/v1/has-facebook-id**

D.1.6.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
facebook_id	Id facebook do usuário para login no facebook	*string*	Obrigatório	character varying(255)

D.1.6.3 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user]	Retorna o usuário recém logado com um authentication_token recém gerado..	*string[]*		
success	Retorna *true*.	*boolean*		

D.1.6.4 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: *"Nenhum usuário com este facebook_id foi encontrado "*	*string*		

D.1.7 Users / Busca de Usuários

D.1.7.1 Descrição

Busca usuários pelo nome ou sobrenome ou e-mail. A identificação do e-mail é automática, pela presença do '@'.

Método	**GET**
URL	**/api/v1/user/search-users**

D.1.7.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
text	Texto de input para algoritmo de busca	*string*	Obrigatório	TEXT
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.1.7.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o login	*string*	Obrigatório	20

D.1.7.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user[], isFriend]	Retorna um *array* de usuários com a informação se cada usuário é ou não amigo do usuário requisitante (isFriend = *true* ou *false*)	*string[][], boolean*		
success	Retorna *true*.	*boolean*		

D.1.8 Users / Retornar Dados de um Usuário

D.1.8.1 Descrição

Retorna os dados de um usuário e um booleano isFriend. isFriend é *true* se o usuário buscado é amigo do requisitante e *false* caso contrário.

Método	**GET**
URL	**/api/v1/user/get-user-data**

D.1.8.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user_id	ID do usuário buscado.	*integer*	Obrigatório	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.1.8.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.1.8.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user, isFriend]	Retorna um usuário com a informação se cada usuário é ou não amigo do usuário requisitante (isFriend = *true* ou *false*)	*string[], boolean*		
success	Retorna *true*.	*boolean*		

D.2 Tags

D.2.1 Tags / Listar Tags

D.2.1.1 Descrição

Retorna todas as *tags* relacionadas ao usuário.

Método	**GET**
URL	**/api/v1/tags/get-tags**

D.2.1.2 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.2.1.3 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.2.1.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
data[tag][]	Retorna *array* de *tags* associadas a esse usuário	*array*		
success	Retorna *true*	*boolean*		1

D.2.1.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.2.2 Tags / Criar Tag

D.2.2.1 Descrição

Cria uma associação entre uma *tag* e um *user*.

Método	**POST**
URL	**/api/v1/tags/create-tag**

D.2.2.2 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.2.2.3 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
tag[id]	O tag[id] corresponde a uma *tag*	*Integer*		

D.2.2.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*	*boolean*		1
info	Retorna uma mensagem de sucesso	*string*		

D.2.2.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.2.3 Tags / Excluir Tag

D.2.3.1 Descrição

Remove uma associação entre uma *tag* e um *user*.

Método	**GET**
URL	**/api/v1/tags/delete-tag**

D.2.3.2 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.2.3.3 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
tag[id]	O tag[id] corresponde a uma *tag*	*Integer*		

D.2.3.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*	*boolean*		1
info	Retorna uma mensagem de sucesso	*string*		

D.2.3.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3 Rotas

D.3.1 Rotas / Listar Rotas

D.3.1.1 Descrição

Retorna todas as rotas associadas ao usuário logado.

Método	**GET**
URL	**/api/v1/route/get-routes**

D.3.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.1.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.1.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
data[routes][]	Retorna *array* de *routes*	*String[]*		1
success	Retorna *true*	*boolean*		1

D.3.1.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.2 Rotas / Criar Rota

D.3.2.1 Descrição

Cria uma rota com os atributos *name* e *description*.

Método	**POST**
URL	**/api/v1/route/create-route**

D.3.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[name]	Descriçao route[name] = Nome da rota	*string*	Obrigatório	--
route[description]	Descriçao route[description] = Descrição da rota	*string*	Opcional	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.2.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true.*	*boolean*		1
info	Mensagem "Route criado com sucesso."	*string*		

D.3.2.5 Resposta - Erro - HTTP 422

success	Retorna *false.*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.3 Rotas / Retornar Rota

D.3.3.1 Descrição

Retorna rota especificada pelo identificador.

Método	**GET**
URL	**/api/v1/route/get-route**

D.3.3.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Descriçao route[id] = Id da rota	*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.3.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.3.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[routes][]	Retorna *array* de *routes*	*String[][]*		1
success	Retorna *true*	*boolean*		1

D.3.3.5 Resposta - Erro - HTTP 422

success	Retorna *false.*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.4 Rotas / Excluir Rota

D.3.4.1 Descrição

Exclui uma rota do sistema.

Método	**DELETE**
URL	**/api/v1/route/destroy-route**

D.3.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Descriçao route[id] = Id da rota	*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.4.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.4.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		1
info	Mensagem "Route excluído com sucesso."	*string*		

D.3.4.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.5 Rotas / Adicionar itinerário

D.3.5.1 Descrição

Adiciona um roteiro (*Itinerary)* a uma rota especificada por um indentificador.

Método	**POST**
URL	**/api/v1/route/add-itinerary**

D.3.5.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Descriçao route[id] = Id da rota	*integer*	Obrigatório	--
itinerary[id]	Descrição itinerary[id] = Id do itinerário	*integer*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.5.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.5.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		1
info	Mensagem "Associação criada com sucesso."	*string*		

D.3.5.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.6 Rotas / Excluir itinerário

D.3.6.1 Descrição

Remove uma associação entre um itinerário e uma rota.

Método	**DELETE**
URL	**/api/v1/route/delete-itinerary**

D.3.6.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Descriçao route[id] = Id da rota	*integer*	Obrigatório	--
itinerary[id]	Descrição itinerary[id] = Id do itinerário	*integer*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.6.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.6.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		1
info	Mensagem "Associaçao removida com sucesso."	*string*		

D.3.6.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.7 Rotas / Adicionar usuário

D.3.7.1 Descrição

Cria uma associação entre um usuário e uma rota.

Método	**POST**
URL	**/api/v1/route/add-user**

D.3.7.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Id da rota	*string*	Obrigatório	--
user[email]	E-mail do usuário que está adicionando o *user* à rota	*string*	Obrigatório	
user_id	Id do *user* que é adicionado à rota	*integer*	Obrigatório	

D.3.7.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.7.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		1
info	Mensagem "Associação criada com sucesso."	*string*		

D.3.7.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.8 Rotas / Excluir usuário

D.3.8.1 Descrição

Exclui uma associação entre um usuário e uma rota.

Método	DELETE
URL	**/api/v1/route/delete-user**

D.3.8.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Id da rota	*string*	Obrigatório	--
user[email]	E-mail do usuário que está adicionando o *user* à rota	*string*	Obrigatório	
user_id	Id do *user* que é adicionado à rota	*integer*		

D.3.8.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.8.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true.*	*boolean*		1
info	Mensagem "Associação removida com sucesso."	*string*		

D.3.8.5 Resposta - Erro - HTTP 422

success	Retorna *false.*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.3.9 Rotas / Match de Rota

D.3.9.1 Descrição

Verifica a existência de roteiros que possuam ao menos uma data coincidente e retorna a lista de rotas (*Matchs)* e roteiros(*Itineraries*) relacionados.

Método	**POST**
URL	**/api/v1/route/match-route**

D.3.9.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Id da rota	*string*	Obrigatório	--
user[email]	E-mail do usuário que está adicionando o *user* à rota	*string*	Obrigatório	

D.3.9.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.9.4 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
data[users[cityName]][]	Retorna o nome da cidade onde aconteceu o *match*	*string*		
data[users[matchDates][]]	Retorna as data onde aconteceu o *match*	*array*		
data[users[hits][]]	Retorna o número de hits(número de datas encontradas)	*string*		
success	*True* ou *False* dependendo do sucesso ou não ao encontrar ao menos um *match*.	*boolean*		

D.3.11 Rotas / Retornar usuários associados à rota

D.3.11.1 Descrição

Retorna lista de usuários associados a uma determinada rota e o id do seu administrador.

Método	**GET**
URL	**/api/v1/route/get-route-users**

D.3.11.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
route[id]	Descriçao route[id] = Id da rota	*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.3.11.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticaçao do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.3.11.4 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
data[users[]]	Retorna *array* com os dados de todos os usuários associados à rota, exceto os dados de autenticação. Se não existem usuários relacionados à rota devolve um *array* vazio []	*string*		
data[admin_id]	Retorna id do usuário administrador, criador da rota	*integer*		
success	Retorna *true*.	*boolean*		1

D.4 Roteiros

D.4.1 Roteiros / Pegar Roteiros

D.4.1.1 Descrição

Retorna roteiro (*Itinerary)* especificado pelo identificador. Observação: este roteiro (*Itinerary)* deve, obrigatoriamente, pertencer a uma rota que está associada ao usuário.

Método	**GET**
URL	**/api/v1/itinerary/get-itinerary**

D.4.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
itinerary[id]	Descrição itinerary[id] = Id do itinerário	*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.4.1.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.4.1.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[itinerary]	Retorna roteiro	*String[]*		1
success	Retorna *true*	*boolean*		1

D.4.1.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.4.2 Roteiros / Excluir Roteiro

D.4.2.1 Descrição

Exclui um roteiro (*Itinerary*).

Método	DELETE
URL	**/api/v1/itinerary/destroy-itinerary**

D.4.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
itinerary[id]	Descrição itinerary[id] = Id do itinerário	*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.4.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.4.2.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
info	Mensagem "Itinerary excluido	*String[]*		1

	com sucesso."			
success	Retorna *true*	*boolean*		1

D.4.2.5 Resposta - Erro - HTTP 422

success	Retorna *false.*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.4.3 Roteiros / Gerar Roteiro

D.4.3.1 Descrição

Retorna um roteiro (*Itinerary)* a partir dos seguintes parâmetros: cidade, datas que compreendem a estadia do usuário na cidade e interesses do usuário.

Método	**POST**
URL	**/api/v1/itinerary/generate-itinerary**

D.4.3.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
city[id]	Id único da cidade.	*string*	Obrigatório	--
startDate	Data e horário de início	*string*	Obrigatório	
endDate	Data e horário de término	*string*	Obrigatório	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.4.3.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.4.3.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[itinerary]	Retorna roteiro	*string[]*		1
success	Retorna *true*	*boolean*		1

D.4.3.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.4.4 Roteiros / Listar Eventos

D.4.4.1 Descrição

Retorna um *array* com todos os Eventos que estão associados ao roteiro (*Itinerary)* especificado pelo identificador.

Método	**GET**
URL	**/api/v1/itinerary/list-events**

D.4.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
itinerary[id]		*string*	Obrigatório	--
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.4.4.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.4.4.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[events][]	Retorna *array* de eventos	*string[][]*		1
success	Retorna *true*	*boolean*		1

D.4.4.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.4.5 Roteiros / Atualizar Roteiro

D.4.5.1 Descrição

Atualiza roteiro (*Itinerary)* especificada pelo identificador.

Método	**PUT**
URL	**/api/v1/itinerary/update-itinerary**

D.4.5.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
itinerary[id]	Descrição itinerary[id] = Id do itinerário	*string*	Obrigatório	--
startDate	Data e horário de início	*string*	Opcional	
endDate	Data e horário de término	*string*	Opcional	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.4.5.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.4.5.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		1
info	Retorna uma mensagem de sucesso.	*string*		

D.4.5.5 Resposta - Erro - HTTP 422

success	Retorna *false*.	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		

D.5 Eventos

D.5.1 Eventos / Listar Eventos

D.5.1.1 Descrição

Retorna a lista de eventos cadastrados para uma cidade.

Método	**GET**
URL	**/api/v1/events/list-events**

D.5.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
idCity	Identificador único da cidade	*string*	Obrigatório	--

D.5.1.3 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
isSuccess	Retorna 1 se for sucesso ou 0 se houver algum erro.	*boolean*		1
message	Mensagem de sucesso ou erro.	*string*		
events	Retorna um *array* de eventos da cidade	*array*		

D.5.1.4 Events

Parâmetro	Descrição	Type	Status	Tamanho
name	Nome do evento	*string*		1
description	Descrição do evento.	*string*		
idEvent	Retorna o id único do evento	*string*		
address	Endereço do Evento	*string*		
latitude	Latitude	*int*		
longitude	Longitude	*int*		
workingHours	Horário de funcionamento	*string*		
startDate	Data e horário de início	*string*		
endDate	Data e horário de término	*string*		
urlImages	Lista de url das imagens	*array*		

D.5.2 Eventos / Adicionar Evento no Roteiro

D.5.2.1 Descrição

Adiciona um evento a um roteiro (*Itinerary*).

Método	**POST**
URL	**/api/v1/events/add-event**

D.5.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
idItinerary	Identificador único do roteiro	*string*	Obrigatório	
idEvent	Identificador único do evento	*string*	Obrigatório	--
startDate	Data e horário de início	*string*	Obrigatório	
endDate	Data e horário de término	*string*	Obrigatório	

D.5.2.3 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
isSuccess	Retorna 1 se for sucesso ou 0 se houver algum erro.	*boolean*		1
message	Mensagem de sucesso ou erro.	*string*		--
idEventChosen	Retorna o id único do evento escolhido	*string*		--

D.5.3 Eventos / Excluir Evento no Roteiro

D.5.3.1 Descrição

Exclui um evento que pertence a um roteiro(*Itinerary)*.

Método	**POST**
URL	**/api/v1/events/remove-event**

D.5.3.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
idEventChosen	Retorna o id único do evento	*string*	Obrigatório	--

D.5.3.3 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
isSuccess	Retorna 1 se for sucesso ou 0 se houver algum erro.	*boolean*		1
message	Mensagem de sucesso ou erro.	*string*		--

D.5.4 Eventos / Atualizar Evento do Roteiro

D.5.4.1 Descrição

Atualiza um evento que pertence a um roteiro (*Itinerary*).

Método	**POST**
URL	**/api/v1/events/update-event**

D.5.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
idEventChosen	Retorna o id único do evento	*string*	Obrigatório	--
startDate	Data e horário de início	*string*	Obrigatório	
endDate	Data e horário de término	*string*	Obrigatório	

D.5.4.3 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
isSuccess	Retorna 1 se for sucesso ou 0 se houver algum erro.	*boolean*		1
message	Mensagem de sucesso ou erro.	*string*		--

D.6 Perfil

D.6.1 Perfil / Criar perfil usuário

D.6.1.1 Descrição

Cria o perfil completo de um usuário logado.

Método	**POST**
URL	**/api/v1/profile/create-profile**

D.6.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[about]	Texto com a descriçao pessoal do usuário	*string*	Opcional	character varying(255)
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.6.1.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.6.1.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		
info	Mensagem: *"Perfil pertencente ao usuário #{@user.email} foi criado com sucesso."*	*string*		

D.6.1.5 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: "Tentativa de criação inválida no controller #{_object}."	*string*		

D.6.2 Perfil / Pegar perfil usuário

D.6.2.1 Descrição

Retorna o perfil completo de um usuário especificado pelo identificador.

Método	**GET**
URL	**/api/v1/profile/get-profile**

D.6.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.6.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.6.2.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data	Retorna o Perfil do usuário	*Hash[attribute] [value]*		
success	Retorna true	*boolean*		

D.6.2.5 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*string*		
info	Retorna a mensagem: "Tentativa de GET inválida no controller #{_object}."	*string*		

D.6.3 Perfil / Atualizar perfil do usuário

D.6.3.1 Descrição

Atualiza o perfil do usuário logado, retornando o perfil atualizado.

Método	**PUT**
URL	**/api/v1/profile/update-profile**

D.6.3.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
user[about]	Texto com a descrição pessoal atualizada do usuário	*string*	Opcional	character varying(255)

D.6.3.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.6.3.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
info	Mensagem: "Os detalhes de perfil pertencente ao usuário #{User.email} foram atualizados com sucesso."	*string*		
success	Retorna true	*boolean*		

D.6.3.5 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*string*		
info	Retorna a mensagem: "Tentativa de GET inválida no controller #{_object}."	*string*		

D.6.4 Perfil / Pegar perfil simples usuário

D.6.4.1 Descrição

Retorna um perfil simplificado de um usuário especificado pelo identificador.

Método	**GET**
URL	**/api/v1/profile/get-profile-simple**

D.6.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
idUser	Id do usuário	*string*	Obrigatório	

D.6.4.3 Resposta

Parâmetro	Descrição	Type	Status	Tamanho
isSuccess	Retorna 1 se for sucesso ou 0 se houver algum erro.	*boolean*		1
message	Mensagem de sucesso ou erro.	*string*		
name	Nome do usuário.	*string*		
lastName	Sobrenome do usuário.	*string*		
urlImage	Url da imagem do usuário	*string*		

D.7 Amigos

D.7.1 Amigos / Listar amigos

D.7.1.1 Descrição

Retorna a lista de amigos de um usuário logado no sistema.

Método	GET
URL	/api/v1/friendship/get-friends

D.7.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.7.1.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.7.1.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user][]	Returna um *array* de *users* com todos os amigos e suas informações	*string*		
success	Retorna *true*.	*boolean*		

D.7.1.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: *"Não existe nenhum(a) friendship associado ao usuário user[email]"*	*string*		

D.7.2 Amigos / Listar users que o tem como amigo

D.7.2.1 Descrição

Retorna a lista de *users* que possuem o usuário logado na sua lista de amigos.

Método	**GET**
URL	**/api/v1/friendship/get-inverse-friends**

D.7.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.7.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticaçao do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.7.2.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
data[user[][]]	Retorna um array de users com todos os amigos e suas informações	*string*		
success	Retorna *true*.	*boolean*		

D.7.2.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem: *"Não existe nenhum(a) inverse-friendship associado ao usuário user[email]"*	*string*		

D.7.3 Amigos / Adicionar Amigo

D.7.3.1 Descrição

Cria uma relação de amizade entre o usuário especificado pelo identificador e o usuário logado no sistema.

Método	**POST**
URL	**/api/v1/friendship/request-friendship**

D.7.3.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
friend[id]	Id do usuário a ser relacionado ao usuário logado.	*string*	Obrigatório	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.7.3.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.7.3.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		
data[id]	Id na tabela Friendship	*Integer*		
data[user_id]	Id do usuário logado	*Integer*		
data[friend_id]	Id do amigo adicionado	*Integer*		
info	Mensagem de sucesso	*string*		

D.7.3.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem de erro	*string*		

D.7.4 Amigos / Excluir Amigo

D.7.4.1.Descrição

Exclui um usuário especificado pelo identificador da lista de amigos do usuário logado no sistema.

Método	**DELETE**
URL	**/api/v1/friendship/destroy-friendship**

D.7.4.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
friend[id]	Id do usuário a ser relacionado ao usuário logado.	*string*	Obrigatório	
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)

D.7.4.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticaçao do usuário retornado ao realizar o login	*string*	Obrigatório	20

D.7.4.4 Resposta - Sucesso - HTTP 200

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*.	*boolean*		
info	Mensagem de sucesso	*string*		

D.7.4.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Mensagem de erro	*string*		

D.8 Configuração

Setting[id]	Descrição	Type	Values	Default
1	Permissão para *match*	*boolean*	"1" ou "0"	"0"
2	Permissão para aparecer nas buscas de amigos	*boolean*	"1" ou "0"	"0"

D.8.1 Configuração /Listar configurações

D.8.1.1 Descrição

Retorna uma lista das configurações (*Settings)* do usuário.

Método	**GET**
URL	**/api/v1/settings/get-settings**

D.8.1.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.8.1.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.8.1.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
data[]	Retorna a lista de *settings* do usuário.	*boolean*		1
success	Retorna *true*.	*boolean*		

D.8.1.4 Resposta - Erro - HTTP 401

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		
info	Retorna mensagem de erro	*string*		

D.8.2 Configuração / Atualizar Configurações

D.8.1 Descrição

Atualiza as configurações do aplicativo de um usuário logado no sistema.

Método	**PUT**
URL	**/api/v1/settings/update-settings**

D.8.2.2 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
setting[id]	ID definido na tabela de settings em D.8	*integer*	Obrigatório	1
setting[value]	Aceita os aclores "0" ou "1" que correspondem a true ou false	*boolean*	Obrigatório	1
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	

D.8.2.3 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.8.2.4 Resposta Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
info	Mensagem de sucesso	*string*		
success	Retorna true	*boolean*		

D.8.3.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*string*		
info	Mensagem de erro	*string*		

D.9 Mensagens

D.9.1 Mensagens / Recuperar Mensagens

D.9.1.1 Descrição

Recupera todas as mensagens enviadas pelos usuários em um determinado canal.

Método	**GET**
URL	**/api/v1/message/retrieve-messages**

D.9.1.2 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.9.1.3 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
channel	Canal no qual as mensagens foram enviadas	*string*	Obrigatório	character varying(255)

D.9.1.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
data[][]	Retorna *array* de mensagens envidas pelo *user* no *channel*	*array*		
success	Retorna *true*	*boolean*		1

D.9.2 Mensagens / Armazenar Mensagem

D.9.2.1 Descrição

Armazena a mensagem enviada pelo usuário a um determinado canal.

Método	**POST**
URL	**/api/v1/message/store-message**

D.9.2.2 Parâmetros - HEADER

Parâmetro	Descrição	Type	Status	Tamanho
X-AUTH-TOKEN	Corresponde ao *token* de autenticação do usuário retornado ao realizar o *login*	*string*	Obrigatório	20

D.9.2.3 Parâmetros

Parâmetro	Descrição	Type	Status	Tamanho
user[email]	O user[email] é no formato de email. Ex: user1@example.com	*string*	Obrigatório	character varying(255)
message	O texto da mensagem	*text*	Obrigatório	character varying(TEXT)
channel	Canal no qual a mensagem foi enviada	*string*	Obrigatório	character varying(255)
timestamp	Data e hora no qual a mensagem foi enviada	*datetime*	Obrigatório	

D.9.2.4 Resposta - Sucesso

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *true*	*boolean*		1
data[][]	Retorna	*string[][]*		
info	Retorna uma mensagem de sucesso	*string*		

D.9.2.5 Resposta - Erro

Parâmetro	Descrição	Type	Status	Tamanho
success	Retorna *false*	*boolean*		1
info	Retorna uma mensagem que identifica a origem do erro	*string*		