UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

YASMIN GOMES PEGORARO

APLICATIVO VIVACURITIBA: SUGESTÃO DE ATRAÇÕES E ROTEIROS TURÍSTICOS PARA CURITIBA

CURITIBA

2022

YASMIN GOMES PEGORARO

APLICATIVO VIVACURITIBA: SUGESTÃO DE ATRAÇÕES E ROTEIROS TURÍSTICOS PARA CURITIBA

VivaCuritiba Application: Suggestion of Attractions and Tourist Itineraries for Curitiba

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Profª. Drª. Ana Cristina Barreiras Kochem Vendramin

CURITIBA

2022

YASMIN GOMES PEGORARO

APLICATIVO VIVACURITIBA: SUGESTÃO DE ATRAÇÕES E ROTEIROS TURÍSTICOS PARA CURITIBA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 08/dezembro/2022

Ana Cristina Barreiras Kochem Vendramin
Professora Doutora
Universidade Tecnológica Federal do Paraná

Anelise Munaretto Fonseca
Professora Doutora
Universidade Tecnológica Federal do Paraná

Daniel Fernando Pigatto
Professor Doutor
Universidade Tecnológica Federal do Paraná

CURITIBA
2022

AGRADECIMENTOS

Agradeço à minha família, em especial meus pais Alberto e Marli e meu irmão Felipe, pelo apoio e incentivo incondicionais.

Agradeço à minha orientadora Profª Drª Ana Cristina Barreiras Kochem Vendramin, pelo apoio e dedicação oferecidos para a execução deste Trabalho de Conclusão de Curso.

Agradeço também aos amigos que fiz durante o decorrer desta trajetória e a tornaram muito mais leve.

RESUMO

PEGORARO, Yasmin Gomes. APLICATIVO VIVACURITIBA: SUGESTÃO DE ATRAÇÕES E ROTEIROS TURÍSTICOS PARA CURITIBA. 62 f. Trabalho de Conclusão de Curso de Graduação - Departamento Acadêmico de Informática e Departamento Acadêmico De Eletrônica, Universidade Tecnológica Federal do Paraná, Curitiba, 2022.

Este trabalho tem como objetivo o desenvolvimento de um sistema para dispositivos móveis com sistema operacional Android para geração de roteiros entre pontos turísticos na cidade de Curitiba. O sistema utiliza informações fornecidas pelo usuário, tais como datas e horários, meio de transporte a ser utilizado, além de um conjunto de pontos turísticos de interesse, para gerar um roteiro turístico otimizado. Esse roteiro utiliza um algoritmo de busca pelo menor caminho entre os pontos selecionados, dados meteorológicos de previsão do tempo e os horários de funcionamento de cada local. O sistema também fornece aos usuários sugestões de atrações turísticas geradas a partir de um sistema de recomendação colaborativa baseado em memória, o qual é alimentado com dados históricos de escolhas feitas pelos usuários do sistema.

Palavras-chave: roteiro turístico; sistema de recomendação; dispositivo móvel; sistema operacional Android.

ABSTRACT

PEGORARO, Yasmin Gomes. VIVACURITIBA APPLICATION: SUGGESTION OF ATTRACTIONS AND TOURIST ITINERARIES FOR CURITIBA. 62 f. Trabalho de Conclusão de Curso de Graduação - Departamento Acadêmico de Informática e Departamento Acadêmico De Eletrônica, Universidade Tecnológica Federal do Paraná, Curitiba, 2022.

This work aims to develop a system for mobile devices with Android operating system to generate itineraries between tourist attractions in the city of Curitiba. The system uses information provided by the user, such as dates and times, means of transport to be used, and a set of tourist points of interest, to generate an optimized tourist itinerary. This itinenary uses an algorithm to search for the shortest path between the selected points, weather forecast data and the opening hours of each location. The system also provides users with tourist attractions suggestions generated from a memory-based collaborative recommendation system, which suggestions are based on historical data of choices made by the system's users.

Keywords: tourist itinerary; recommendation system; mobile device; Android operating system.

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

LISTA DE SIGLAS E ACRÔNIMOS

API	*Application Programming Interface*
AWS	*Amazon Web Services*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IDLE	*Integrated Development and Learning Environment*
JSON	*JavaScript Object Notation*
JSX	*JavaScript Syntax Extension*
REST	*Representational State Transfer*
SDK	*Software Development Kit*
SMS	*Short Message Service*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O uso de *smartphones* para a execução de tarefas cotidianas é uma constante cada vez mais presente, em uma parcela cada vez maior da população. A partir desta constatação, abre-se um leque das mais variadas opções de funcionalidade que um sistema projetado para dispositivos móveis pode oferecer aos usuários. O planejamento de um roteiro turístico é uma delas.

A elaboração de um roteiro de viagem é uma atividade que habitualmente requer pesquisa acerca do destino, seja para descobrir atividades disponíveis no local, restaurantes, restrições ou ainda formas de acesso. Como resultado final, pode-se obter uma lista com os pontos de interesse a serem visitados organizados em datas e horários, de acordo com a previsão da viagem.

Com a popularização dos *smartphones*, na qual nota-se a democratização do acesso a diversos recursos tecnológicos, a grande maioria dos modelos já dispõe de funcionalidades básicas que podem auxiliar na tarefa de criação de roteiros turísticos, como o acesso à internet por meio de redes de conexão sem fio e a geolocalização, além da perceptível evolução da capacidade de processamento destes dispositivos. Nesse sentido, encontra-se a oportunidade para a criação de aplicações para objetivos específicos que utilizem essas funcionalidades.

É possível encontrar serviços que reúnam parcialmente essas funções, porém com um enfoque menor em turismo e maior em criação de roteiros de maneira geral. Dentre eles destaca-se o Google Maps, muito utilizado para a obtenção de direções e pesquisa a respeito de um local.

Postas estas elucidações, o presente trabalho tem como objetivo o desenvolvimento de um aplicativo, chamado VivaCuritiba, capaz de sugerir locais de visitação a turistas na cidade de Curitiba e produzir um roteiro turístico otimizado, através de informações simples fornecidas pelo usuário, sendo elas: datas e horários desejados para a realização do roteiro, meio de transporte a ser utilizado para o deslocamento entre os pontos turísticos, além da seleção dos próprios locais a serem visitados, a partir de uma lista preestabelecida e inserida na aplicação.

O aplicativo contará também com um sistema de recomendação colaborativa que utilizará a lista das atrações turísticas selecionadas e buscará um conjunto de opções mais relevantes com base na semelhança desta lista com a seleção de outros usuários, que alimentam a base de dados remota do sistema a cada geração de um roteiro novo. Com isso, busca-se melhorar a experiência turística na cidade através de sugestões customizadas, facilitando o planejamento e tornando a experiência mais positiva para o usuário.

O projeto será desenvolvido para *smartphones* para a obtenção e processamento dos dados citados e utilizará recursos básicos dos dispositivos para fornecer sugestões relevantes que farão parte de um itinerário customizado e otimizado a ser fornecido ao usuário como resultado final de sua utilização. A escolha de dispositivos móveis como plataforma de execução do projeto se deu devido justamente à sua mobilidade, tornando-o compatível com o desloca-

mento envolvido em uma atividade turística e permitindo que seja acessível a consultas durante a execução da rota.

1.1 Motivação

Pode-se definir turismo como "o movimento temporário de pessoas a destinos fora de seus locais normais de moradia e trabalho, as atividades realizadas durante a estada nestes locais, e as instalações criadas para atender as necessidades destas pessoas."(MATHIESON; WALL, 1982). É notório que atividades turísticas possuem um impacto econômico relevante, sendo inclusive a principal atividade em algumas localidades, fomentando a economia regional e impulsionando a geração de emprego e circulação de riquezas. Considerando estes conceitos, é possível afirmar que ferramentas que auxiliem turistas a enriquecer e desfrutar de uma estadia mais agradável se colocam como fatores de incentivo à atividade turística e, consequentemente, aos indicadores socioeconômicos atrelados a ela.

De acordo com o Plano Municipal de Turismo de Curitiba (CURITIBA, 2015), o fluxo turístico em Curitiba cresceu num ritmo superior à média brasileira e paranaense e de um modo constante ao longo da década de 2000. O número estimado de turistas que visitaram a cidade em 2019 é de 5,8 milhões, período pré-pandemia da COVID-19. Analisando os números de indicadores como o número de acessos à página institucional do Curitiba Turismo e o número de atendimentos a turistas nos postos de informações da cidade entre 2020 e 2021, é perceptível a tendência de aumento na procura por esses serviços dado o início da retomada das atividades após a interrupção causada pela pandemia. Isso indica o retorno da expressividade do setor turístico na configuração da economia local em um cenário pós-pandêmico (CURITIBA, 2022).

Baseando-se nas informações expostas anteriormente, este projeto possui como finalidade o desenvolvimento de um sistema para auxiliar turistas que desejem visitar a cidade de Curitiba, com foco na criação de roteiros turísticos.

1.2 Objetivo Geral e Objetivos Específicos

O objetivo geral do presente trabalho é desenvolver um sistema para sugerir pontos turísticos com base nas seleções feitas pelo usuário e por fim gerar um roteiro otimizado a partir dos dados fornecidos.

Como objetivos específicos do projeto, definem-se:

- Desenvolver um sistema que ofereça sugestões de pontos turísticos da cidade de Curitiba que sejam relevantes ao usuário;
- Desenvolver um sistema que seja capaz de otimizar uma rota;

- Desenvolver um aplicativo para dispositivos móveis para que o usuário tenha acesso ao sistema proposto através de um aparelho celular;
- Implementar uma base de dados com informações sobre os pontos turísticos disponíveis no sistema para seleção do usuário;
- Implementar uma base de dados com informações a respeito das preferências dos usuários do sistema para alimentar o sistema de recomendação.

1.3 Estrutura do Documento

Este documento divide-se em seis capítulos. O primeiro capítulo fornece a motivação, objetivos e justificativa para realização do presente trabalho. O segundo capítulo apresenta a fundamentação teórica utilizada para o desenvolvimento do sistema proposto, além de trabalhos previamente realizados que se relacionam com o projeto proposto. O terceiro capítulo descreve o projeto de *software* criado durante o planejamento do aplicativo VivaCuritiba. O quarto capítulo compreende a descrição dos recursos de *software* utilizados no desenvolvimento. O quinto capítulo fornece detalhes acerca do desenvolvimento e utilização do sistema, descrevendo o sistema de recomendação implementado e sua comunicação com o aplicativo VivaCuritiba, apresentando a interface gráfica do aplicativo e demonstrando o seu uso. O sexto capítulo apresenta as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos base e fundamentações teóricas utilizados no concebimento deste trabalho. São detalhados também os trabalhos relacionados ao projeto proposto neste documento.

2.1 Sistemas de Recomendação

Com a notável popularização da utilização de recursos *on-line* através da internet, constata-se também o aumento da quantidade de informações e opções disponíveis para consulta e consumo. Esse grande volume de dados que teve o acesso facilitado trouxe também uma dificuldade de escolha, devido ao fato de existirem muitas alternativas disponíveis. Visando endereçar essa questão conceberam-se os sistemas de recomendação, que podem ser definidos como ferramentas e técnicas de *software* que fornecem sugestões úteis a um usuário (RICCI; ROKACH; SHAPIRA, 2011). Tais sugestões podem estar inseridas em um contexto específico, como por exemplo qual filme assistir, qual música ouvir, qual produto comprar, entre muitos outros com que as pessoas se deparam no cotidiano.

Existem diferentes métodos de predição utilizados na construção de sistemas de recomendação (ISINKAYE; FOLAJIMI; OJOKOH, 2015), como filtragem baseada em conteúdo e filtragem colaborativa. Este trabalho possui enfoque na recomendação colaborativa baseada em memória.

2.1.1 Recomendação Colaborativa

Sistemas de recomendação baseados em recomendação colaborativa, ou filtragem colaborativa, utilizam informações fornecidas por outros usuários do sistema para filtrar, dentre as opções disponíveis, itens que tenham sido escolhidos por usuários de preferências semelhantes (ISINKAYE; FOLAJIMI; OJOKOH, 2015). De modo geral, esses sistemas buscam padrões para predizer alternativas mais assertivas ao usuário.

Dentre os métodos de filtragem colaborativa, destacam-se a filtragem baseada em modelo e a filtragem baseada em memória (LEVINAS, 2014). A recomendação feita a partir de um modelo utiliza técnicas de *machine learning*[1] na tentativa de prever qual seria a classificação dada por um item que ainda não tenha sido avaliado por um determinado usuário. É uma abordagem interessante para sistemas em que a lista de opções disponíveis recebe constantes atualizações, já que a predição feita é baseada em reconhecimento de padrões, aplicando o que foi aprendido a qualquer item no contexto do sistema (COLLEGE, 2022b). Existe, porém, um custo computacional maior envolvido neste método de filtragem se comparado com a recomen-

[1] Do inglês aprendizado de máquina, é um ramo da inteligência artificial que associa análise de dados e algoritmos (IBM, 2022).

dação colaborativa baseada em memória (LEVINAS, 2014). Esse segundo método de filtragem colaborativa se baseia no grau de satisfação dos usuários com determinada opção para traçar a similaridade entre eles, através do histórico de escolhas realizadas por essas pessoas no universo de opções (COLLEGE, 2022a). O resultado almejado consiste em encontrar os itens mais similares com base nas preferências fornecidas ao sistema ao longo do tempo e recomendar os itens que ainda não tenham sido escolhidos. Para este projeto, considerando que a base de dados que fornece as opções de escolha do usuário é estática, o modelo baseado em memória foi escolhido.

2.1.2 Recomendação Baseada em Memória

Algoritmos para recomendação baseados em memória utilizam a similaridade entre os interesses de usuários para sugerir novos itens. Parte-se do pressuposto de que, se um usuário A demonstra preferências semelhantes a um usuário B, itens escolhidos por A que ainda não tenham sido escolhidos por B seriam sugestões mais próximas a algo que B optaria naturalmente.

Uma possível implementação para esse modelo de recomendação se utiliza de uma matriz que relaciona *m* usuários com *n* itens em um dado sistema. A intersecção entre um usuário e um item nessa matriz contém o grau de satisfação deste usuário com este determinado item, podendo ser uma nota atribuída ou a quantidade de vezes em que o item foi escolhido pelo usuário (MEDEIROS, 2013). A Tabela 1 demonstra um exemplo de pequena escala de uma matriz de usuários, na qual os números presentes na interseção entre linha e coluna representam a nota atribuída por um usuário à opção correspondente.

Tabela 1 – Matriz usuário x item de um sistema

	Opção 1	**Opção 2**	**Opção 3**	**Opção 4**
Usuário 1	8			10
Usuário 2	8	10		
Usuário 3		10	10	
Usuário 4			8	
Usuário 5	10	8		

Define-se também a matriz de similaridade, que representa o grau de compatibilidade entre as preferências dos usuários. Essa matriz pode ser obtida utilizando a similaridade por cosseno, que utiliza o valor do cosseno do ângulo formado entre dois vetores para determinar o nível de similaridade (MEDEIROS, 2013). Considerando uma matriz usuário-item de ordem *m x n*, pode-se obter a similaridade entre dois itens A e B através do cosseno entre os vetores de tamanho *n*, encontrados nas colunas de índices correspondentes a esses dois itens na matriz. A Equação 1 apresenta o cálculo da similaridade por meio do cosseno entre dois vetores (MEDEIROS, 2013).

$$\cos(\mathbf{A},\mathbf{B}) = \frac{\mathbf{AB}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} \mathbf{A}_i \mathbf{B}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{A}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{B}_i)^2}} \quad (1)$$

2.2 Problema do Caminho Mínimo - Algoritmo de Djiskstra

Em teoria dos grafos, existe a concepção do problema do caminho mínimo, cujo princípio recai sobre encontrar o caminho cujo custo seja o menor possível, ou seja, a soma do peso das arestas percorridas seja mínimo visitando todos os vértices de um determinado conjunto (BLACK, 2020). Considerando cada vértice do gráfico como um local e o peso de cada aresta como a medida de distância linear que separa cada dois desses vértices, é possível aplicar algoritmos de busca do menor caminho para se obter a rota mais curta para visitar um grupo de locais selecionados.

Para este projeto, propôs-se a implementação do algoritmo de Djikstra (CORMEN *et al.*, 2009), o qual soluciona o problema para grafos de aresta com pesos positivos, neste caso, a distância entre as atrações turísticas. O algoritmo consiste em analisar todas as arestas disponíveis saindo do vértice atual e escolher como próximo vértice aquele que dispuser da aresta de menor peso, ou menor distância, com o vértice atual. Ao final, retorna-se a sequência de vértices que compõe o caminho mínimo a ser executado (Algoritmo 1).

Algoritmo 1 – Algoritmo de Djisktra

1: sendo $V[G]$ o conjunto de vértices v pertencentes ao grafo $G = (V,E)$
2: sendo $d[v]$ a distância entre o vértice de origem a cada outro vértice v
3: **para todos** $v \in V[G]$ **faça**
4: $d[v] = \infty$ {I}nicializa-se com valor infinito (pior hipótese)
5: $prev[v] =$ null
6: **finaliza para**
7: $d[origem] = 0$
8: $Q = V[G]$
9: **enquanto** $Q \neq \varnothing$ **faça**
10: $u = extrairMin(Q)$
11: **para cada** v adjacente a u **faça**
12: **se** $d[v] > d[u] + peso(u,v)$ **então**
13: $d[v] = d[u] + peso(u,v)$
14: $prev[v] = u$
15: **finaliza se**
16: **finaliza para**
17: **finaliza enquanto**
18: return dist[], prev[]

Fonte: Adaptado de Cormen *et al.* (2009).

2.3 Trabalhos Relacionados

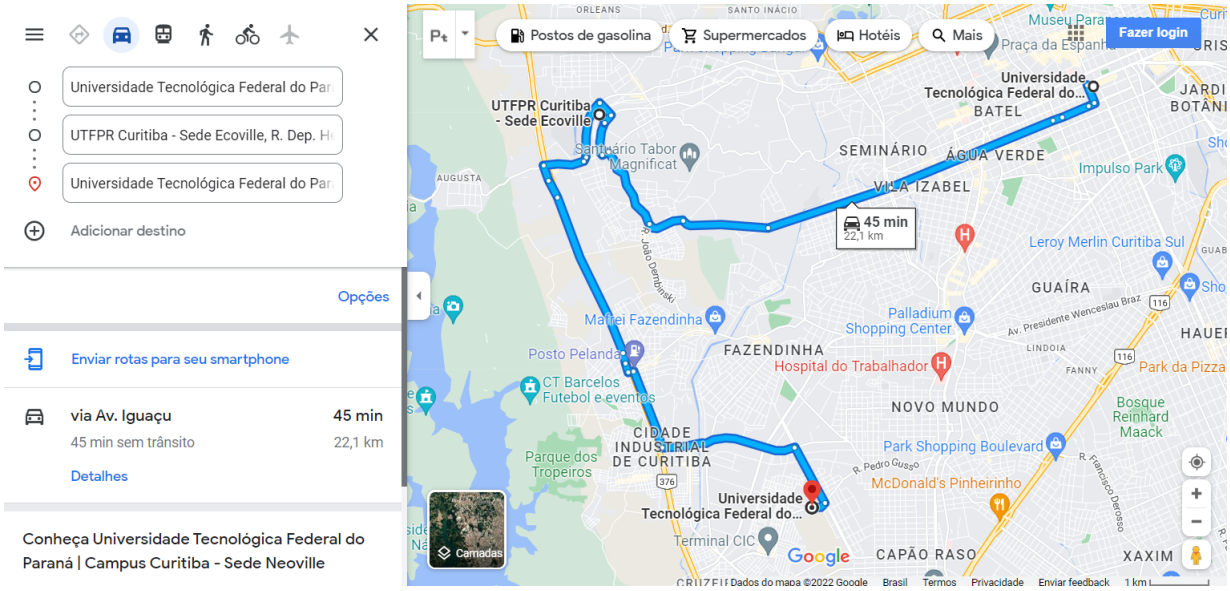
Essa seção apresenta os estudos relacionados ao presente trabalho.

2.3.1 Google Maps

Google Maps (GOOGLE, 2022b) é um serviço gratuito oferecido pela empresa Google que possui, dentre muitas funcionalidades, função de pesquisa e visualização de mapas. Lançado no ano de 2005, atualmente conta com dados da maior parte dos países. A ferramenta permite o planejamento de rotas a partir de dois ou mais pontos no mapa em uma ordem de visitação indicada pelo usuário, oferecendo também a escolha de meio de transporte a ser utilizado, sendo as opções oferecidas: a pé, carro, bicicleta, transporte público ou avião. A ferramenta considera dados de trânsito relevantes à seleção do usuário para fornecer a rota requerida.

A Figura 1 ilustra um exemplo de roteiro oferecido pela ferramenta, para um deslocamento de carro entre três pontos.

Figura 1 – Roteiro criado pelo Google Maps



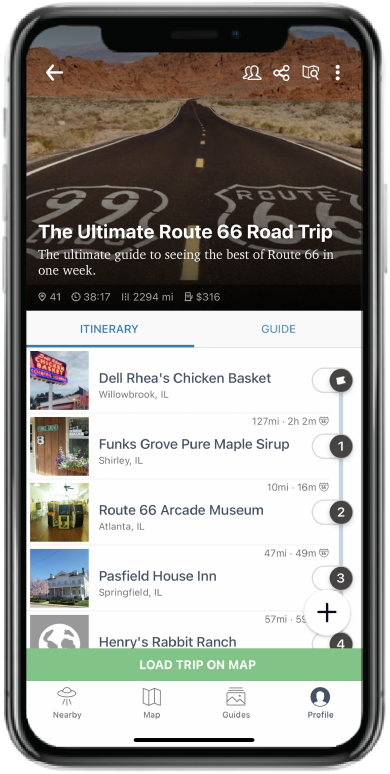
Fonte: Adaptado de Google (2022a).

2.3.2 *Roadtrippers*

Roadtrippers (ROADTRIPPERS, 2022) é um aplicativo disponível em versões para *desktop*, Android e iOS. Ele foi desenvolvido com o intuito de auxiliar o usuário a planejar uma viagem a partir dos pontos de partida e de chegada, oferecendo opções para que o viajante possa conhecer locais nas proximidades ao longo do trajeto, tendo acesso a avaliações deixadas por outros usuários e podendo salvar os lugares de interesse para futura referência. O aplicativo conta ainda com funcionalidades para a organização da viagem, como informações a respeito dos hotéis próximos e estimativas de consumo de combustível.

A Figura 2 exibe uma interface do aplicativo Roadtrippers em sua versão para *smartphones*, em que pode ser observado o itinerário definido pelo usuário.

Figura 2 – Itinerário criado através do Roadtrippers



Fonte: Roadtrippers (2021).

2.3.3 Descubra Curitiba

Descubra Curitiba é um aplicativo Android desenvolvido como Trabalho de Conclusão de Curso para obtenção do título de Bacharel em Engenharia de Computação na UTFPR, câmpus Curitiba, apresentado pelos estudantes Eduardo Nardi e José Augustinho e orientado pelas professoras doutoras Ana Cristina Kochem Vendramin e Anelise Munaretto Fonseca. O aplicativo calcula um itinerário turístico para a cidade de Curitiba considerando as entradas do usuário, as características de cada local e as condições climáticas do dia (NARDI; AUGUSTINHO, 2018).

A Figura 3 apresenta a tela inicial deste aplicativo.

Figura 3 – Tela inicial do aplicativo Descubra Curitiba

Fonte: Nardi e Augustinho (2018).

2.3.4 Discussões

A partir das elucidações dos trabalhos citados no decorrer deste capítulo, é possível levantar alguns pontos de discussão no que se refere aos diferenciais do aplicativo VivaCuritiba proposto neste documento.

Analisando a funcionalidade de rotas oferecida pelo Google Maps, o aplicativo VivaCuritiba tem como principal diferença a busca por uma rota otimizada a partir dos locais selecionados, cuja ordem de visitação será definida pela aplicação e não pelo usuário, em contraposição à ferramenta desenvolvida pela empresa Google.

O aplicativo *Roadtrippers* possui o mesmo enfoque em planejamento de roteiros e descoberta de atrações turísticas, porém suas funcionalidades atuam como uma interface que visa facilitar a tarefa de organização da rota de viagem, incumbindo o viajante de analisar as avaliações feitas por outros usuários e traçar sua própria rota. O aplicativo VivaCuritiba prevê funcionalidades que fornecem esses dados de forma automática ao usuário.

Em relação ao projeto Descubra Curitiba, o principal diferencial do aplicativo VivaCuritiba reside no sistema de recomendação colaborativa proposto com o fim de oferecer ao usuário opções que não tenham sido selecionadas anteriormente, buscando enriquecer o roteiro a ser percorrido baseado em um algoritmo de sugestão. Outro ponto de diferença está na possibili-

dade de selecionar vários meios de transporte para a execução da rota, enquanto o aplicativo Descubra Curitiba utiliza apenas rotas a serem percorridas de carro.

Pode-se concluir que o aplicativo VivaCuritiba possui similaridades com alguns trabalhos já executados, destacando-se o projeto Descubra Curitiba. Observa-se, porém, que o aplicativo VivaCuritiba apresenta alguns diferenciais relevantes que somam-se às funcionalidades já oferecidas aos usuários em outros projetos.

No próximo capítulo é descrito o projeto de *software* construído durante o planejamento do aplicativo VivaCuritiba, apresentando o escopo do sistema, os casos de uso e os requisitos funcionais e não funcionais.

3 PROJETO DE SOFTWARE

Para o desenvolvimento do aplicativo VivaCuritiba, as seguintes etapas foram seguidas:

- **Definição do escopo do projeto:** são estabelecidas as funcionalidades que serão incluídas na aplicação proposta e quais serão os casos de uso aplicados ao projeto;
- **Definição dos requisitos funcionais e não-funcionais:** nesta etapa é realizado o levantamento dos requisitos que guiarão a implementação, bem como a escolha das tecnologias mais adequadas para a execução do trabalho;
- **Estudo das tecnologias a serem utilizadas:** é realizado um levantamento e análise das tecnologias disponíveis cuja utilização no projeto é viável. Com base nesse estudo, são realizadas as escolhas de quais recursos proveriam uma utilização mais propícia ao sistema proposto, considerando critérios de facilidade de acesso/utilização, custo, documentação, possibilidade de aplicação no escopo do projeto e familiaridade da desenvolvedora com tal tecnologia;
- **Estudo de outros sistemas relacionados:** é o estudo bibliográfico e teórico realizado antes da fase de implementação do sistema, para que seja possível planejar a execução e mapear a forma de implementação de cada solução proposta;
- **Especificação da execução do projeto:** esta etapa inclui o planejamento sequencial de como se dará a implementação de cada elemento do projeto;
- **Implementação e testes:** etapa na qual é realizada a execução das fases planejadas anteriormente, além de testes para averiguação do funcionamento do sistema.

Este capítulo apresenta detalhes do projeto de *software* elaborado durante o planejamento do aplicativo VivaCuritiba. No próximo capítulo é apresentado o estudo das tecnologias que foram utilizadas na concepção do sistema. No Capítulo 5 são apresentadas especificações do processo de desenvolvimento dos componentes do sistema, além de um caso de utilização do aplicativo VivaCuritiba.

A seguir é apresentado o escopo do sistema e descrita sua modelagem através do diagrama de casos de uso e especificação dos requisitos funcionais e não funcionais.

3.1 Escopo do Sistema

O sistema proposto neste documento deve fornecer ao usuário um roteiro otimizado para a visitação de atrações turísticas na cidade de Curitiba. O acesso ao sistema se dá através do aplicativo VivaCuritiba.

A escolha dos pontos turísticos é feita a partir de uma lista com 15 atrações, conforme consta na Quadro 1. Essa lista contém o nome da atração, um imagem, uma breve descrição do local, *tags* que identificam as categorias às quais a atração pertence (parque, igreja, museu, compras) e um campo indicando se a atração é ao ar livre. Esse campo é considerado junto aos dados de previsão do tempo para evitar que locais abertos sejam escolhidos em caso de mau tempo.

Quadro 1 – Lista de atrações turísticas disponíveis no sistema.

Nome da Atração	**Categoria**	**Ao ar livre?**
Bosque Alemão	Parque	Sim
Igreja da Ordem	Igreja	Não
Jardim Botânico	Parque	Sim
Mercado Municipal de Curitiba	Compras	Não
Museu Oscar Niemeyer (Museu do Olho)	Museu	Não
Museu Paranaense	Museu	Não
Park Shopping Barigui	Compras	Não
Parque Barigui	Parque	Sim
Parque São Lourenço	Parque	Sim
Parque Tanguá	Parque	Sim
Parque Tingui	Parque	Sim
Passeio Público	Parque	Sim
Rua 24 Horas	Compras	Não
Shopping Estação	Compras	Não
Shopping Mueller	Compras	Não

Após selecionar as atrações, o usuário deve selecionar também as datas e horários nos quais deseja realizar o roteiro turístico.

A partir da seleção das atrações desejadas, o sistema fornecerá ainda sugestões de outras atrações que não tenham sido previamente selecionadas, baseadas nas escolhas de outros usuários com interesses semelhantes.

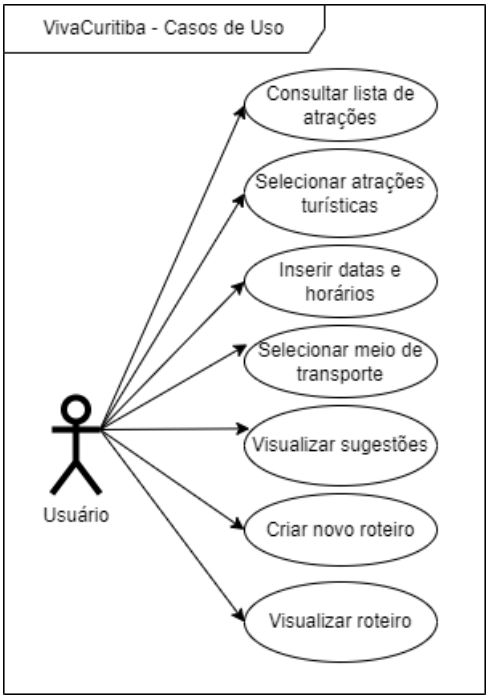
3.2 Modelagem do Sistema

Nesta seção são apresentados os casos de uso do sistema e seus respectivos fluxos, além da descrição dos requisitos funcionais e não funcionais.

3.2.1 Casos de Uso

A Figura 4 exibe o diagrama de casos de uso do aplicativo VivaCuritiba.

Figura 4 – Diagrama de casos de uso do aplicativo VivaCuritiba



Fonte: Autoria própria (2022).

3.2.1.1 Caso de Uso 1 - Consultar lista de atrações

O caso de uso “Consultar atrações” descreve o processo realizado pelo usuário para consultar as atrações turísticas disponíveis para compor o roteiro:

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para consulta de atrações disponíveis no sistema e seus respectivos detalhes;
- **Pré-condições:** o usuário deverá estar com o aplicativo aberto funcionando corretamente;
- **Pós-condições:** será exibida uma lista de atrações disponíveis;
- **Fluxo básico:**

    1. o usuário clica no botão Explorar ou Selecionar Atrações através do toque na tela;

2. é exibida na tela uma lista com itens clicáveis contendo todas as atrações turísticas disponíveis;
3. o usuário clica em uma atração através do toque na tela;
4. são exibidas na tela a descrição e uma imagem referentes ao ponto turístico selecionado.

3.2.1.2 Caso de Uso 2 - Selecionar atrações turísticas

O caso de uso “Selecionar atrações” descreve o processo de escolha dos pontos turísticos que irão compor o roteiro.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para selecionar atrações turísticas que irão compor o roteiro;
- **Pré-condições:** o usuário deverá estar com o aplicativo aberto, acessando a tela de criação de um novo roteiro;
- **Pós-condições**: os pontos turísticos de interesse estarão indicados no sistema;
- **Fluxo básico:**
    1. o usuário clica no botão Selecionar Atrações através do toque na tela;
    2. é exibida na tela uma lista com itens selecionáveis contendo todas as atrações turísticas disponíveis;
    3. o usuário seleciona atrações através do toque na tela.

3.2.1.3 Caso de Uso 3 - Inserir datas e horários

O caso de uso “Inserir datas e horários” descreve o processo no qual o usuário indica as datas e horários nos quais deseja executar o roteiro pela cidade de Curitiba.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para indicar datas e horários que irão compor o roteiro;
- **Pré-condições:** o usuário deverá estar com o aplicativo aberto, acessando a tela de criação de um novo roteiro;
- **Pós-condições:** as datas e horários em que o roteiro será executado estarão indicados no sistema;

- **Fluxo básico:**

    1. o usuário clica no botão Selecionar Datas e Horários através do toque na tela;
    2. o usuário insere datas e horários indicando início e fim do trajeto.

3.2.1.4 Caso de Uso 4 - Selecionar meio de transporte

O caso de uso “Selecionar meio de transporte” descreve o processo de escolha do meio de transporte com o qual o roteiro será executado.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para indicar o meio de transporte utilizado para percorrer o roteiro;
- **Pré-condições:** o usuário deverá estar com o aplicativo aberto, acessando a tela de criação de um novo roteiro;
- **Pós-condições**: o meio de transporte que será utilizado estará indicado no sistema;
- **Fluxo básico:**

    1. o usuário seleciona um dos meios de transporte disponíveis através do toque na tela;
    2. o meio de transporte escolhido fica indicado pela cor laranja.

3.2.1.5 Caso de Uso 5 - Visualizar sugestões

O caso de uso “Visualizar sugestões” descreve o processo que faz com que as sugestões obtidas pelo sistema sejam exibidas ao usuário.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para visualizar as sugestões oferecidas pelo sistema de pontos turísticos a serem visitados;
- **Pré-condições:**

    – o usuário deverá ter indicado quais pontos deseja visitar, datas, horários e meios de transporte;
    – o dispositivo móvel deverá estar conectado à internet;

- **Pós-condições**: as sugestões de pontos turísticos estarão visíveis ao usuário;

- **Fluxo básico:**

    1. o usuário clica no botão Criar Roteiro através do toque na tela;
    2. é exibida na tela uma lista com itens selecionáveis contendo as atrações turísticas sugeridas pelo algoritmo de recomendação.

3.2.1.6 Caso de Uso 6 - Criar novo roteiro

O caso de uso “Criar novo roteiro” descreve o processo que realiza a construção do roteiro a partir das inserções do usuário.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para criação de um novo roteiro a partir dos dados inseridos no sistema;
- **Pré-condições:**
    - o usuário deverá ter indicado quais pontos deseja visitar, datas, horários e meios de transporte;
    - as sugestões deverão ter sido exibidas pelo usuário;
    - o usuário deverá estar na tela que exibe as sugestões;
    - o dispositivo móvel deverá estar conectado à internet;
- **Pós-condições:** o roteiro sugerido ficará disponível para o usuário;
- **Fluxo básico:**

    1. o usuário clica no botão Adicionar Atrações ou Voltar;
    2. é exibida na tela uma lista com os pontos turísticos escolhidos, na ordem a serem visitados e com indicação de datas e horários.

3.2.1.7 Caso de Uso 7 - Visualizar roteiro

O caso de uso “Visualizar roteiro” descreve o processo que faz com que o roteiro criado pelo sistema seja exibido ao usuário.

- **Ator principal:** usuário;
- **Descrição:** caso de uso executado para visualização de um roteiro gerado pelo sistema;
- **Pré-condições:**

  - o usuário deverá ter indicado quais pontos deseja visitar, datas, horários e meios de transporte e dado o comando para a criação de um novo roteiro;
  - o usuário deverá acessar a tela de Roteiros Salvos;

- **Pós-condições**: o roteiro ficará visível para o usuário;
- **Fluxo básico:**

  1. o usuário clica no botão Criar Roteiro através do toque na tela ou seleciona um dos roteiros salvos disponíveis;
  2. é exibida na tela uma lista com os pontos turísticos escolhidos, na ordem a serem visitados e com indicação de datas e horários.

3.2.2 Requisitos Funcionais

Para este projeto, foram levantados os seguinte requisitos funcionais:

- O sistema deverá conter uma lista estática de atrações turísticas da cidade de Curitiba a serem utilizadas na elaboração dos roteiros;
- O sistema deverá permitir a seleção de atrações turísticas;
- O sistema deverá permitir a visualização dos detalhes referentes a cada atração turística;
- O sistema deverá permitir a seleção de datas e horários para a execução de um roteiro turístico;
- O sistema deverá conter uma lista de meios de transporte disponíveis para a elaboração do roteiro;
- O sistema deverá permitir a seleção de um meio de transporte a ser utilizado para percorrer o trajeto;
- O sistema deverá sugerir pontos turísticos a partir das seleções de um usuário;
- O sistema deverá apresentar ao usuário as sugestões indicadas;
- O sistema deverá utilizar dados de previsão do tempo para priorizar atrações turísticas que não sejam ao ar livre em caso de mau tempo;
- O sistema deverá utilizar dados de horário de funcionamento das atrações turísticas para a criação do roteiro;
- O sistema deverá gerar um roteiro otimizado a partir das seleções do usuário;

- O sistema deverá utilizar um algoritmo de melhor caminho para a criação do roteiro;
- O sistema deverá permitir que um roteiro gerado seja salvo;
- O sistema deverá permitir que um roteiro salvo seja consultado.

3.2.3 Requisitos Não Funcionais

Para este projeto, foram levantados os seguinte requisitos não funcionais:

- O aplicativo deve executar em um aparelho celular com Android versão 11 ou superior;
- O aplicativo deve utilizar conexão à internet para realizar suas funções;
- O aplicativo deve ser escrito na linguagem de programação JavaScript, fazendo uso da biblioteca *React Native*;
- O sistema de recomendação deve ser escrito na linguagem de programação Python;
- O sistema de recomendação deve ser hospedado em serviços AWS.

Dadas as especificações contidas neste capítulo, a seguir são apresentados os recursos de *software* utilizados no desenvolvimento do sistema.

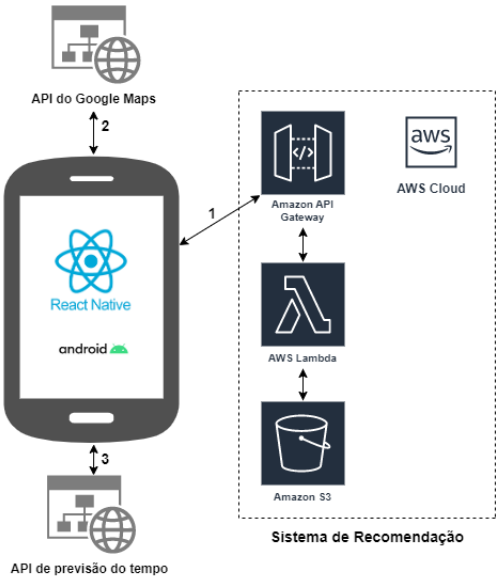
4 RECURSOS DE *SOFTWARE*

Este trabalho propõe o desenvolvimento de um sistema cuja interface com o usuário se dá através do aplicativo VivaCuritiba construído para ser executada em *smartphones*. Esse aplicativo possui como objetivo auxiliar no planejamento e execução de roteiros turísticos na cidade de Curitiba, provendo sugestões de pontos de visitação e apresentando uma rota otimizada.

O aplicativo foi desenvolvido com enfoque no sistema operacional para *smartphones* Android, devido à facilidade de acesso às ferramentas necessárias de desenvolvimento para tal plataforma.

A Figura 5 exibe uma visualização de como os componentes do sistema interagem entre si. Nela é possível visualizar a troca bilateral de dados entre o aplicativo e o sistema de recomendação, composto por serviços AWS que serão detalhados na subseção 4.2.4, além da comunicação com a *Application Programming Interface* (API) que fornece dados de previsão de tempo (subseção 4.2.6) e com a API do Google Maps. É indicada também a sequência da interação entre esses elementos: (1) primeiramente é realizada a requisição ao sistema de recomendação para que a seleção das atrações turísticas seja finalizada; (2) em seguida, o sistema interage com a API do Google Maps para obtenção de detalhes sobre cada atração, além de dados da distância entre os pontos turísticos; (3) por fim, é realizada a chamada para a API que fornece dados da previsão do tempo ao sistema.

Figura 5 – Diagrama de blocos do aplicativo VivaCuritiba



Fonte: Autoria própria (2022).

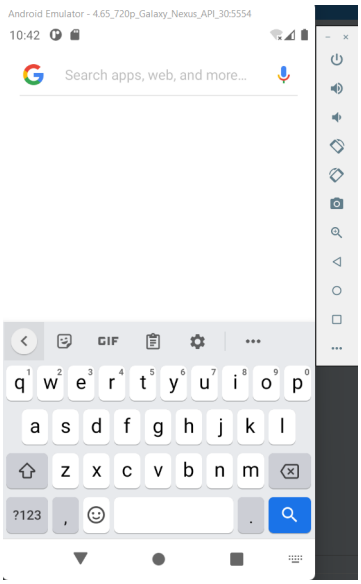
O objetivo desse capítulo é apresentar uma visão geral dos componentes do sistema proposto, começando pela descrição do funcionamento do sistema operacional Android (ver Seção 4.1) e posteriormente detalhando as tecnologias utilizadas (ver Seção 4.2).

4.1 Sistema Operacional Android

Android é um sistema operacional para dispositivos móveis lançado em 2008, cujo principal desenvolvedor é a empresa americana Google (ANDROID, 2022a). O Android está presente em diversos modelos de aparelhos de diferentes fabricantes, sendo a plataforma mais utilizada em todo o mundo (STATCOUNTER, 2022). Ele foi construído em código aberto baseado no núcleo Linux e oferece uma vasta gama de possibilidades de personalização.

O kit de desenvolvimento de *software* Android *Software Development Kit* (SDK) reúne ferramentas utilizadas por programadores no desenvolvimento de aplicações para o sistema operacional Android. O kit é instalado junto ao Android Studio, que possibilita a utilização de um simulador para executar a aplicação em diferentes modelos de *smartphone* e em diferentes versões do sistema operacional. Android Studio é o ambiente de desenvolvimento de *software* oferecido pela empresa Google e utilizado para os testes do aplicativo VivaCuritiba. A *Integrated Development Environment* (IDE) provê suporte para diversas versões do sistema operacional Android, além de fornecer um ambiente de simulação das aplicações desenvolvidas, como mostra a Figura 6. Para a visualização e testes do aplicativo VivaCuritiba, foram utilizados recursos de emulação de um *smartphone* com sistema Android presente no Android Studio, com o qual o *React Native* fornece uma integração para a execução de aplicações.

Figura 6 – Emulador disponível no Android Studio.

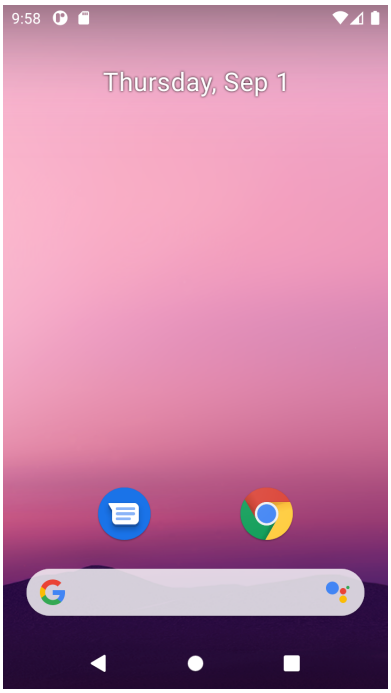


Fonte: Autoria própria (2022).

A interface de usuário do Android é baseada em manipulação direta, produzindo uma resposta a estímulos captados por sensores internos presentes em *smartphones*, como acelerômetros e sensores de proximidade (ANDROID, 2022b). O objetivo é transmitir uma sensação de fluidez durante a utilização do sistema. Na tela inicial padrão do Android estão presentes ícones de navegação, compostos por atalhos a aplicativos, além de controles para percorrer e acessar as diferentes telas. Ela entrega uma grande capacidade de customização, permitindo ao usuário

aplicar mudanças no *design* de acordo com suas preferências pessoais. No topo encontra-se a barra de status, na qual é possível verificar informações sobre o dispositivo e notificações recebidas. A Figura 7 apresenta um exemplo de tela inicial de um dispositivo Android.

Figura 7 – Tela inicial de um dispositivo Android



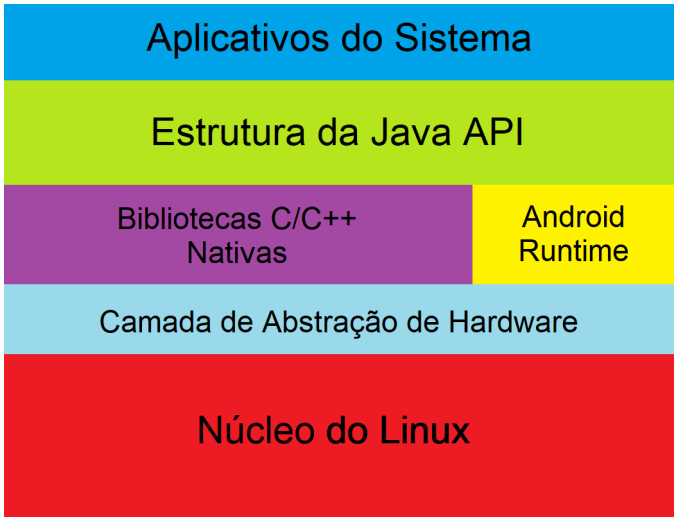
Fonte: Autoria própria (2022).

A arquitetura de um sistema Android é dividida em cinco camadas, como ilustra a Figura 8 (PROJECT, 2020):

- no nível mais baixo encontra-se o núcleo do Linux, o qual é a fundação da plataforma, permitindo que se utilizem recursos de segurança existentes e que os fabricantes de dispositivos se aproveitem do conhecimento previamente atestado no processo de desenvolvimento de *drivers* de *hardware*;
- acima encontra-se a camada de abstração de *hardware*, responsável por interfacear o *hardware* do dispositivo com a estrutura da Java API, através de módulos de biblioteca desenvolvidos para a interação com cada componente físico do dispositivo, como a câmera;
- a camada a seguir é constituída por dois módulos: o Android Runtime (a partir da versão 5.0 do sistema operacional), responsável por compilar arquivos de aplicativos no momento de sua instalação, e as bibliotecas C/C++ nativas, para componentes implementados em código nativo e que também são expostas a aplicativos de terceiros através das Java Framework APIs;
- as APIs tem como função expor os recursos do Android para sua utilização durante o desenvolvimento de aplicativos, de modo a simplificar esse processo;

- por fim, a camada mais alta da arquitetura Android consiste nos aplicativos do sistema, os quais podem ser utilizados como aplicativos pelos usuários e também fornecem aos desenvolvedores acesso a capacidades do sistema sem que seja necessário programar uma mesma funcionalidade novamente, como por exemplo o envio de SMS, que pode ser feito por um aplicativo de terceiros invocando o aplicativo de *Short Message Service* (SMS) do sistema.

Figura 8 – Camadas da arquitetura do sistema operacional Android



Fonte: Adaptado de Project (2020).

As características apresentadas nessa seção sobre o ambiente de desenvolvimento para aplicativos Android motivaram a escolha por esse sistema operacional, visto que o iOS, segundo sistema operacional mais popular entre os usuários, não oferece aos desenvolvedores tanta flexibilidade, sendo necessário utilizar um computador pessoal da marca Apple.

4.2 Tecnologias

Para a execução deste projeto foram selecionadas as tecnologias que se enquadram de forma mais apropriada ao sistema proposto e aos recursos disponíveis.

Para a implementação da aplicação móvel foi escolhida a linguagem de programação JavaScript, utilizando a biblioteca React-Native associada ao SDK do Android. Essa escolha foi motivada pelo fato de ser uma biblioteca multiplataforma, ou seja, é possível criar aplicações para diferentes sistemas operacionais utilizando o mesmo código-fonte, além da familiaridade da autora com o JavaScript.

Para o desenvolvimento do sistema de recomendação, foi necessária uma estrutura que fosse externa à aplicação e cujo acesso se desse de maneira remota, devido ao fato da interação dos usuários com o sistema depender da capacidade do aplicativo de alimentá-lo e consumi-lo.

Para a implementação do sistema de recomendação, foi utilizada a linguagem de programação Python, por oferecer bibliotecas que contêm funções utilizadas nesta operação.

O código do sistema de recomendação foi hospedado em um serviço AWS Lambda (AMAZON, 2022c) oferecido pela Amazon Web Services. Foi utilizado também o serviço de Amazon API Gateway (AMAZON, 2022a), responsável pela interface entre o sistema de recomendação e a aplicação móvel através de um *endpoint* de uma API RESTful, além do Amazon Simple Storage Service (AMAZON, 2022b), para o armazenamento dos dados dos usuários que são responsáveis por alimentar o sistema de sugestões.

O projeto conta também com uma integração à *One Call API* para a obtenção de dados de previsão do tempo que serão considerados para a construção do roteiro, caso o local de visitação seja classificado como "ao ar livre".

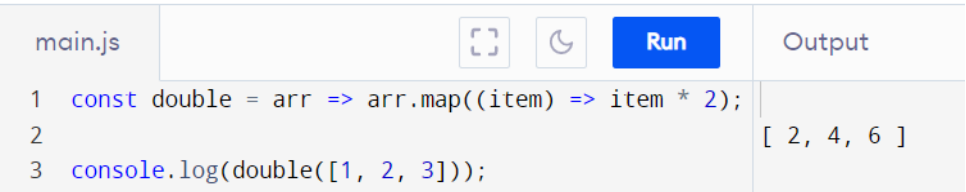
Essas tecnologias estão detalhadas nas seções a seguir.

4.2.1 JavaScript

JavaScript é uma linguagem de programação de alto nível, criada em 1995 (MOZILLA, 2022). Atualmente é a linguagem mais utilizada para a programação no "lado do cliente", atuando majoritariamente no desenvolvimento *front-end*, permitindo o funcionamento dinâmico de páginas web em diversos navegadores. Também é utilizada para desenvolvimento *back-end*, de jogos e aplicações para dispositivos móveis, o qual é o foco deste projeto.

Essa versatilidade é possível devido ao fato de ser uma linguagem multiparadigma, ou seja, suporta diferentes formas de funcionamento e estruturação. Dentre eles, encontra-se o paradigma de programação declarativa, utilizado na concepção da biblioteca React-Native e que pode ser definido como uma descrição do que o programa faz, em oposição à especificação de cada passo que será executado para obtenção do resultado. A Figura 9 apresenta um exemplo de código de um programa escrito de forma declarativa em JavaScript, o qual imprime na tela um vetor que contém o dobro dos valores inseridos em um vetor recebido como entrada.

Figura 9 – Programação declarativa em JavaScript



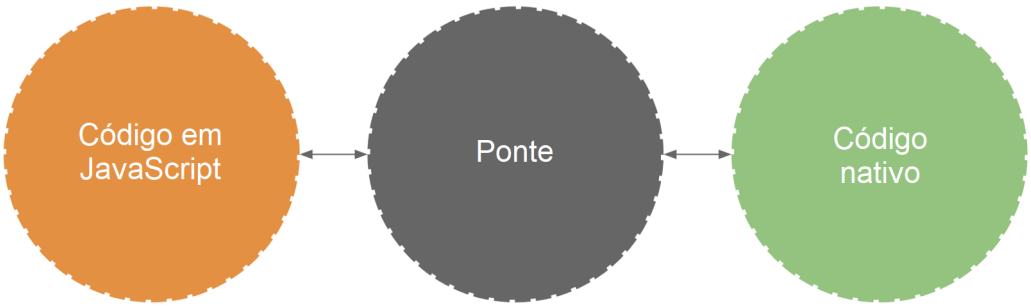
Fonte: Adaptado de Grachet (2022).

4.2.2 *React Native*

Criada em 2015 pelo Facebook, o React Native é uma biblioteca *open-source* utilizada para desenvolvimento de aplicações *mobile* que mapeia seus componentes aos componentes nativos de cada plataforma (PLATFORMS, 2022d). O React Native possui como ponto de partida o ReactJS (PLATFORMS, 2022c), outra biblioteca JavaScript criada em 2013 pela mesma empresa para a construção de interfaces de usuário. A biblioteca tem como princípio oferecer ao desenvolvedor e aos usuários uma experiência de desenvolvimento nativo em um ambiente multiplataforma, possuindo como lema a expressão *Learn once, write anywhere*, isto é, aprenda uma vez, escreva em qualquer lugar.

Para atingir seu objetivo, o React Native utiliza uma *bridge* (ponte) para traduzir o código escrito em JavaScript para um contexto nativo de diversas plataformas. Essa ponte é um conceito análogo a um servidor de mensagens em um ambiente de *backend* e representa a comunicação assíncrona e bilateral que ocorre entre duas tecnologias diferentes na execução de um mesmo aplicativo (GRACHET, 2022). Ela é escrita em C/C++, possibilitando a execução num contexto multiplataforma. A Figura 10 apresenta uma representação visual do conceito de ponte do React Native. Nela pode-se observar o modo com que a ponte atua como um interceptor na troca bilateral de dados entre o código escrito em JavaScript que faz uso da biblioteca React Native e a execução de comandos em código nativo de uma plataforma, traduzindo as informações trocadas para que a comunicação seja possível.

Figura 10 – Conceito de ponte do React-Native



Fonte: Adaptado de Grachet (2022).

O funcionamento de uma aplicação escrita em React Native se baseia em uma sintaxe chamada de *JavaScript Syntax Extension* (JSX), semelhante à utilizada em *HyperText Markup Language* (HTML), para renderizar seus componentes. É possível também adicionar código JavaScript na composição desses elementos, flexibilizando a implementação. Um dos principais diferenciais oferecidos por essa biblioteca consiste no fato de que, dentre todos os componentes existentes na interface, a aplicação irá renderizar novamente apenas aqueles em que houve alguma mudança (PLATFORMS, 2022e). Para isso há dois tipos de dado que controlam o comportamento de um componente: *props* e *state* Platforms (2022a). As *props* são parâmetros utilizados para a customização de um componente e são fixos durante todo o tempo de

execução de um programa, enquanto *state*, como o próprio nome indica, contém os dados que podem sofrer alteração, caracterizando uma mudança de estado.

Para o desenvolvimento de aplicações a serem executadas em ambiente Android, é possível utilizar o emulador (PLATFORMS, 2022f) disponível na ferramenta oficial para desenvolvimento nativo Android Studio, ou executar a aplicação em um *smartphone* que possua Android como sistema operacional.

A programação do aplicativo em *React Native* foi feita utilizando o Visual Studio Code para a edição dos arquivos JavaScript.

A seguir, são descritos os principais componentes *React Native* utilizados na concepção da aplicação móvel.

4.2.2.1 Componentes React Native

O processo de desenvolvimento de aplicações para dispositivos móveis através do *React Native* utiliza componentes para acessar as APIs nativas de cada sistema, assim como descrever o comportamento e aparência da interface do programa.

Neste projeto foram utilizados os componentes essenciais (*core components*) presentes na biblioteca, providos pelo time responsável pela manutenção do *React Native* (PLATFORMS, 2022b). Além desses componentes, existe também uma vasta gama de opções desenvolvidas pela comunidade que utiliza a biblioteca e compartilha componentes para reutilização. Neste projeto, foi utilizado o componente de comunidade chamado *Google Places AutoComplete* (SAFI; YASKEVICH; PONTES, 2015) para a obtenção do ponto inicial do roteiro através de um dado de entrada fornecido pelo usuário.

Os componentes do *React Native* utilizados são (PLATFORMS, 2022b):

- *View*: funciona como um contentor capaz de receber atributos para o controle da interface de usuário, como a estilização, acessibilidade, entre outros. É responsável por armazenar e distribuir informações na tela, podendo incluir outros componentes filhos em sua hierarquia;
- *Scrollview*: funciona como um contentor para visualização, acrescido da funcionalidade de rolagem na tela para acessar informações que não estão acessíveis na visualização inicial da tela;
- *TouchableOpacity*: tem como definição ser responsivo à interação com o usuário através de toques na tela, tornando-se menos opaco quando pressionado, de forma a fornecer um retorno ao usuário de que a ação (toque) foi capturada pela aplicação. Na prática, seu funcionamento é como o de um botão, podendo ser estilizado e receber uma função com comandos a serem executados em seu acionamento;

- *FlatList*: oferece diversas funcionalidades na utilização de listas em uma aplicação, como a configuração de múltiplas colunas, cabeçalho, rodapé e separadores, além de ser multiplataforma. São altamente customizáveis, podendo ser utilizados para a criação de diversos modelos de lista;
- *Modal*: utilizado para a criação de caixas de diálogo. Ele pode ser customizado e incluir outros componentes para compor os comportamentos desejados. Oferece 3 opções de animação: *slide, fade* e *none*;
- *Google Places AutoComplete* (SAFI; YASKEVICH; PONTES, 2015): utiliza serviços do *Google Maps* para capturar a entrada do usuário, no momento que está buscando o endereço (no presente projeto, o ponto inicial do roteiro), e apresentar sugestões baseadas no que foi inserido até agora, completando o restante do endereço, além de salvar o endereço selecionado para posterior utilização. É necessário possuir uma chave válida para a utilização da *Google Places API*.

4.2.3 Python

Python é uma linguagem de programação criada em 1991 e utilizada em diversos tipos de aplicações, como desenvolvimento web, *machine learning* e *data science* (INSTITUTE, 2022). Devido a sua grande abrangência de aplicabilidade, foram desenvolvidas e disponibilizadas inúmeras bibliotecas que podem ser empregadas em diferentes fins, reduzindo o volume de código e tornando-o mais legível.

Para este projeto, foram utilizadas bibliotecas para a leitura/escrita e manipulação dos dados que alimentam o sistema de recomendação. Para a execução do algoritmo, funções matemáticas para o processamento desses dados também foram adotadas através de bibliotecas para a realização dos cálculos necessários.

Para o desenvolvimento e testes do sistema de recomendação, utilizou-se o *Integrated Development and Learning Environment* (IDLE). O IDLE é um ambiente de desenvolvimento integrado para programas escritos em Python, o qual possui um editor de texto simples para a escrita do código fonte e permite visualizar o resultado da execução do programa através de uma janela de saída.

4.2.4 *Amazon Web Services* (AWS)

Amazon Web Services, também conhecido por seu acrônimo *Amazon Web Services* (AWS), é uma plataforma de serviços de computação em nuvem pertencente à Amazon, fundada em 2006 (AMAZON, 2022d). Seus produtos permitem o desenvolvimento de soluções de forma escalável e oferecem disponibilidade através da redundância existente em suas zonas divididas por regiões geográficas. O navegador Google Chrome foi utilizado para a interação com

os serviços AWS, através da versão web do console de gerenciamento oferecido pela empresa. Para este projeto foram utilizados três serviços: AWS Lambda, Amazon API Gateway e Amazon S3, que são detalhados nas seções a seguir.

4.2.4.1 AWS Lambda

Lançado no ano de 2014, AWS Lambda é um serviço que oferece a execução de código sem que o desenvolvedor necessite provisionar ou gerenciar a estrutura para tal, ou seja, de forma transparente ao usuário (AMAZON, 2022c). Seu funcionamento é acionado através de eventos, ou gatilhos, que são configurados na própria plataforma da AWS.

A escolha do AWS Lambda para a execução do *script* do sistema de recomendação considerou o suporte à linguagem de programação escolhida Python para o desenvolvimento do *script*, a facilidade de acesso e configuração desse serviço, além de sua garantia de disponibilidade.

No sistema descrito neste documento, foi utilizado o serviço de API Gateway para disparar a execução do código, o qual está descrito a seguir.

4.2.4.2 *Amazon API Gateway*

A empresa define o Amazon API Gateway como "um serviço da AWS para criação, publicação, manutenção, monitoramento e proteção de APIs *Representational State Transfer* (REST) e WebSocket em qualquer escala."(AMAZON, 2022a). Ele possibilita a criação de APIs RESTful baseadas no protocolo *HyperText Transfer Protocol* (HTTP)[1] para o acesso a serviços web através de seus *endpoints*. Neste projeto foi habilitado o método HTTP POST para interação com o sistema de recomendação, cujo código foi hospedado em uma instância do serviço AWS Lambda. Ele foi implementado para a inserção de novos dados de usuário na alimentação do sistema através do corpo da solicitação; a resposta dessa requisição contém os resultados obtidos após o processamento do sistema de recomendação, os quais foram inseridos na resposta da API para serem lidos pela aplicação móvel.

4.2.4.3 Amazon S3 (*Simple Storage Service*)

Amazon S3 é um serviço de armazenamento de dados (AMAZON, 2022b), que os organiza em unidades básicas de armazenamento, os *buckets*, ou baldes. O conteúdo desses *buckets* pode ser acessado pelo próprio console disponibilizado pela AWS, ou através de uma interface REST, como foi utilizado neste projeto.

[1] HTTP é um protocolo de comunicação de informação amplamente utilizado na transmissão de dados na *World Wide Web*

Para a construção da estrutura que reúne os dados para o sistema de recomendação, foi criado um *bucket* contendo três arquivos .csv (*comma-separated values)*:

- identificação de cada ponto turístico incluso no sistema;
- identificação de usuários;
- o grau de satisfação de cada usuário com cada ponto turístico.

4.2.5 *Google Maps API*

Criada e mantida pela empresa americana Google, a Google Maps API é um serviço que oferece diversas APIs relacionadas à geolocalização. Através dela é possível dispor de diversas funcionalidades oferecidas nas versões web e para dispositivos móveis, porém através da execução de requisições HTTP ao invés da interação com uma interface de usuário.

Para o aplicativo VivaCuritiba, os seguintes serviços da *Google Maps API* foram utilizados para os seguintes fins:

- *Directions API*: traçar as rotas entre um ponto de partida e um ponto de chegada (ver subseção 4.2.5.1);
- *Places API*: verificar se o local a ser visitado estará aberto para visitação no horário definido pelo roteiro (ver subseção 4.2.5.2);
- *One Call API*: para a utilização desta API, é necessário enviar como parâmetro valores de latitude e longitude do local. No aplicativo VivaCuritiba, esses dados são obtidos através da subseção 4.2.5.2 (ver subseção 4.2.6).

4.2.5.1 *Directions API*

A *Directions API* fornece informações a respeito da rota entre dois pontos selecionados. Sua escolha para este projeto foi baseada no fato de esta API ser capaz de fornecer a distância entre dois pontos de acordo com o meio de transporte escolhido. Deste modo, foi possível habilitar a customização do modo de locomoção para a rota a ser construída, sendo as opções disponíveis: carro, transporte público, bicicleta e a pé. Vale notar que para a cidade de Curitiba, a opção transporte público é equivalente à rede de ônibus disponível na cidade. Através desta chamada de API também foi possível recuperar a informação referente ao valor de identificação de cada local no sistema do *Google Maps*, chamado de *Place ID*, além das coordenadas geográficas do ponto em questão, que são utilizadas para a leitura de dados de previsão do tempo. A resposta obtida ao chamar a *Directions API* retornou a informação em formato JSON, sendo lida pela aplicação e processada de acordo com funcionamento do sistema. Os campos utilizados como entrada para a utilização desta API são:

- Ponto de origem;
- Ponto de destino;
- Meio de transporte;
- Chave de autenticação para utilização da API.

Os dados utilizados como ponto de origem e ponto de destino, com exceção ao ponto de partida que dá início ao roteiro, estão presentes em uma lista interna da aplicação que contém os pontos turísticos disponíveis no sistema para geração de rotas.

4.2.5.2 *Places API*

A *Places API* foi utilizada para a obtenção de detalhes a respeito de locais do mapa (isto é, os pontos turísticos adicionados no roteiro). Através do *Place ID* obtido na chamada da *directions API*, a *Places API* fornece as informações referentes aos horários de funcionamento do local de interesse. Este dado foi utilizado para validar se o ponto analisado é um possível candidato a próximo local de visitação, sendo descartado caso esteja fechado no intervalo atual do roteiro que está sendo construído. De forma análoga à *directions API*, ao chamar a *Places API* foi obtida uma resposta em formato JSON, lida e processada pela aplicação durante a criação da rota que será retornada como resultado final para o usuário. Essa chamada permite que apenas os dados de interesse sejam retornados, indicando-os nos parâmetros da requisição. Para o aplicativo VivaCuritiba, as informações desejadas da *Places API* são latitude e longitude, que serão utilizadas na chamada da *One Call API*, além dos horários de funcionamento. Os campos utilizados como entrada para a utilização desta API são:

- *Place ID*;
- Dados de interesse (horários de funcionamento, latitude e longitude);
- Chave de autenticação para utilização da API.

4.2.6 *One Call API*

Criada pela empresa britânica OpenWeather (OPENWEATHER, 2022), a *One Call API* fornece dados relacionados ao tempo de um determinado ponto de interesse, como temperatura atual, previsão minuto a minuto, alertas emitidos por autoridades nacionais, entre outros.

Este serviço foi utilizado para evitar que atrações turísticas ao ar livre sejam escolhidas como próximo ponto de visitação em caso de mau tempo.

Para a utilização do serviço, foram fornecidas à API as informações de latitude e longitude do ponto em questão, para limitação da localização geográfica de interesse, além de

excluir da requisição outras configurações de resposta que não fossem relacionadas à previsão do tempo de hora em hora, a qual é a mais adequada para o objetivo proposto visto que a decisão tomada pelo algoritmo é baseada em intervalos de tempo proporcionais a uma visita turística. Também é fornecida uma chave pessoal para a utilização da API, para fins de autenticação.

Foi utilizada a funcionalidade relacionada à previsão do tempo para os oito dias posteriores à data em que a API está sendo acionada, sendo esse o limite máximo de intervalo de tempo disponível na versão gratuita do serviço. O roteiro a ser construído pelo aplicativo VivaCuritiba é composto por intervalos com indicação de horário de início e fim, fornecendo uma sugestão de tempo de permanência, considerando o tempo total indicado pelas datas e horários inseridos pelo usuário, o total de atrações turísticas e o tempo de deslocamento entre elas. Foi considerado que um intervalo de tempo do roteiro não estará com tempo propício para atividades ao ar livre quando a resposta da API para esse intervalo possuir um *weather id* de valor menor a 800, conforme a documentação da ferramenta (OPENWEATHER, 2022). Nessa documentação são indicados os seguintes grupos de condições meteorológicas associados ao valor do *weather id*:

- *Group 2xx: Thunderstorm*: indica previsão de tempestades (*weather id* entre 200 e 232);
- *Group 3xx: Drizzle*: indica previsão de garoa (*weather id* entre 300 e 321);
- *Group 5xx: Rain*: indica previsão de chuva (*weather id* entre 500 e 531);
- *Group 6xx: Snow*: indica previsão de neve (*weather id* entre 600 e 622);
- *Group 7xx: Atmosphere*: indica previsão de condições atmosféricas adversas como neblina, névoa, fumaça, tornado, etc. (*weather id* entre 701 e 781);
- *Group 800: Clear*: indica previsão de tempo com céu claro (*weather id* de 800);
- *Group 80x: Clouds*: indica previsão de nuvens *weather id* entre 801 e 804).

4.2.7 Serviços Web

Serviços Web, do inglês *Web Services*, podem ser definidos como uma solução para integrar sistemas e permitir a comunicação entre diferentes aplicações que são construídas em plataformas diferentes através de uma rede (POLO, 2018). Essa interação é possível através da tradução da linguagem de cada aplicação para uma linguagem universal capaz de ser lida por diferentes tipos de sistema, como por exemplo o *JavaScript Object Notation* (JSON) e o *Extensible Markup Language* (XML).

Para a implementação deste projeto, foi utilizado o padrão de arquitetura REST (do inglês Transferência Representacional de Estado), considerada mais leve e compatível com a necessidade de aplicações desenvolvidas para dispositivos móveis. Padrões de arquitetura são responsáveis por definir um conjunto de restrições a serem usadas na criação de serviços web. No caso do REST, a utilização de operações bem definidas, cujas mensagens escritas em uma sintaxe universal possuem toda a informação necessária para seu processamento contribuem para a escalabilidade e bom desempenho demonstrado por aplicações que seguem esse padrão.

Para o teste de resposta das APIs RESTful utilizadas neste projeto, foi utilizado o aplicativo Postman (POSTMAN, 2022).

O próximo capítulo apresenta o funcionamento do sistema de recomendação e do aplicativo VivaCuritiba.

5 APLICATIVO VIVACURITIBA E SISTEMA DE RECOMENDAÇÃO

Neste capítulo está descrito o processo de desenvolvimento do sistema proposto. É descrito o funcionamento do sistema de recomendação hospedado na AWS e o processo de conexão desse sistema com o aplicativo VivaCuritiba. Também é apresentada a interface gráfica do aplicativo. Ao final do capítulo, demonstra-se um exemplo de utilização do aplicativo.

5.1 Sistema de Recomendação

Nesta seção é apresentado o sistema de recomendação construído para este projeto. Como já descrito no capítulo anterior, o sistema é composto por um *script* escrito na linguagem de programação Python que é executado em um serviço AWS Lambda e acessado através de uma API RESTful criada com o serviço AWS API Gateway. Para o armazenamento e leitura dos dados que alimentam o sistema, é utilizado um serviço AWS S3.

Através de um *endpoint* configurado para esta API, o aplicativo VivaCuritiba envia os dados dos usuários e obtém uma resposta contendo os dados de pontos turísticos a serem recomendados.

5.1.1 Recebendo Escolhas do Usuário

Para enviar ao sistema de recomendação as escolhas feitas pelo usuário dentre a lista de opções disponíveis, o aplicativo VivaCuritiba encapsula a lista de atrações selecionadas em um objeto JSON juntamente ao identificador único do dispositivo[1], para a localização do usuário no sistema. É feita uma requisição HTTP POST para o *endpoint* /default/recommendationSystem configurado na AWS API Gateway através da *Uniform Resource Locator* (URL) criada para a utilização do serviço, como é possível observar no Código 1.

[1] Combinação alfanumérica única que identifica um dispositivo móvel, obtida através da biblioteca *React Native Device Info* (REACT, 2022)

Código 1 – Exemplo de requisição para o sistema de recomendação

```
POST /default/recommendationSystem HTTP/1.1
Host: 5s2nwgnkj2.execute-api.us-east-2.amazonaws.com
Content-Type: application/json
Content-Length: 268

{
    "user_id": "0c1999fec8e0573b",
    "atracoes": [
        {
            "atracao_id": 1,
            "atracao_name": "Parque Barigui"
        },
        {
            "atracao_id": 7,
            "atracao_name": "Parque São Lourenço"
        }
    ]
}
```

Fonte: Autoria própria (2022).

A seguir é detalhado o processamento dos dados recebidos pela requisição.

5.1.2 Processamento dos Dados de Entrada

As informações recebidas pela API são utilizadas como entrada pelo *script* que executa o sistema de recomendação. Ele recupera o identificador de usuário contido no corpo da requisição e o procura dentre os usuários existentes no sistema, que estão armazenados em uma estrutura AWS S3. Caso o usuário não exista, ele é criado no sistema e suas preferências são salvas, atribuindo uma classificação numérica de valor 1 a cada atração turística selecionada por ele. Caso o usuário já exista no sistema, a classificação de cada ponto turístico enviado na solicitação é atualizada, acrescentando-se o valor 1 à nota já existente para aquela opção e salvando o valor atualizado na base de dados. O Código 2 apresenta o código-fonte implementado para esse processo.

Código 2 – Reprodução do processamento dos dados de entrada do sistema de recomendação

```
def lambda_handler(event, context):
    user_to_generate_recommendation = body["user_id"]

    object_key = 'users.csv'
    csv_obj = client.get_object(Bucket=bucket_name, Key=object_key)
    body = csv_obj['Body']
    csv_string = body.read().decode('utf-8')
    users = pd.read_csv(StringIO(csv_string))

    input_user = (users.loc[users['user_id'] ==
    user_to_generate_recommendation])
    input_user_ratings = (ratings.loc[ratings['userId'] ==
    user_to_generate_recommendation])

    if input_user.empty:
        new_user = pd.DataFrame([user_to_generate_recommendation],
        columns=(['user_id']))
        users = users.append(new_user)
        csv_buffer2 = StringIO()
        users.to_csv(csv_buffer2, index=False)
        s3_resource = boto3.resource('s3')
        s3_resource.Object(bucket_name, 'users.csv')
        .put(Body=csv_buffer2.getvalue())

    for atracao in atracoes_escolhidas:
        atracao_id = atracao["atracao_id"]
        registered_rating = (input_user_ratings
        .loc[input_user_ratings['atracaoId'] == atracao_id])
        if not registered_rating.empty:
            previous_rating = int(float(registered_rating["rating"]))
            ratings.loc[(ratings['userId'] ==
            user_to_generate_recommendation) & (ratings['atracaoId'] ==
            atracao_id), 'rating'] = str(previous_rating+1)
        else:
            user_rating = pd.DataFrame([[user_to_generate_recommendation,
            atracao_id, previous_rating+1]], columns=list(['userId',
            'atracaoId', 'rating']))
            ratings = ratings.append(user_rating)

        csv_buffer = StringIO()
        ratings.to_csv(csv_buffer, index=False)
        s3_resource = boto3.resource('s3')
        s3_resource.Object(bucket_name, 'ratings.csv')
        .put(Body=csv_buffer.getvalue())
```

Fonte: Autoria própria (2022).

5.1.3 Obtendo as Recomendações

A resposta obtida através da chamada da API descrita na subseção 5.1.1 também é em formato JSON, contendo um vetor que representa a lista de atrações turísticas recomendadas a partir da lista enviada. Ela é o resultado do processamento de funções matemáticas utilizadas no sistema de recomendação descrito no Capítulo 2.

Inicialmente, tem-se a matriz usuário x item, na qual as linhas representam os usuários, as colunas representam os pontos turísticos e o valor contido em cada intersecção linha x coluna é a classificação dada pelo dado usuário a um dado ponto turístico, sendo que pontos ainda não avaliados são preenchidos com valor 0. A partir dela é obtida a matriz de similaridade, utilizando o módulo *metrics.pairwise* da biblioteca Python *scikit-learn*. Esse módulo contém uma função para o cálculo de similaridade por cosseno, que é aplicada à matriz usuário x item. Tendo essas informações, o sistema busca na matriz de similaridade, dentre todos os outros usuários, aqueles que sejam mais similares ao usuário em questão, ou seja, que possuam preferências mais parecidas. Na prática, ele busca na matriz os valores mais altos da linha deste usuário, que não seja ele mesmo. A quantidade *k* de usuários mais similares que deseja-se analisar é definida no código, sendo neste projeto definido como 1. Em seguida, obtém-se a média da classificação de cada um desses usuários mais semelhantes para cada ponto turístico, descarta-se os pontos já visitados pelo usuário a quem será dada a recomendação, e retorna-se os *n* pontos de nota mais alta, cuja quantidade é definida no código, sendo neste projeto definido como 2. Optou-se por não oferecer uma quantidade muito grande de sugestões para não comprometer o tempo de permanência em cada ponto, já que o intervalo de tempo para a execução do roteiro mantém-se o mesmo. Ambos os valores de *k* e *n* são valores fixos definidos na codificação do algoritmo de recomendação.

Por fim, a lista das recomendações obtidas pelo sistema é retornada em formato JSON, como exibido no Código 3. O Código 4 apresenta o trecho de código que realiza esse processamento.

Código 3 – Trecho da resposta de requisição HTTP ao sistema de recomendação

```
    {
    "body-json": [
        {
            "atracaoId": 2.0,
            "nome": "Jardim Bot nico"
        },
        {
            "atracaoId": 3.0,
            "nome": "Museu Oscar Niemeyer (Museu do Olho)"
        }
    ],
    "params": {
        "header": {
            "Accept": "*/*",
            "Accept-Encoding": "gzip, deflate, br",
            "Content-Type": "application/json",
            "Host": "5s2nwgnkj2.execute-api.us-east-2.amazonaws.com",
            "Postman-Token": "e9c13073-f33e-4d77-938e-255d66718bc4",
            "User-Agent": "PostmanRuntime/7.28.4",
            "X-Amzn-Trace-Id": "Root=1-63698e00-6c5e11b0202cee3f3ec05fe4",
            "X-Forwarded-For": "177.220.172.184",
            "X-Forwarded-Port": "443",
            "X-Forwarded-Proto": "https"
        }
    },
    "stage-variables": {},
    "context": {
        "api-id": "5s2nwgnkj2",
        "http-method": "POST",
        "stage": "default",
        "source-ip": "177.220.172.184",
        "request-id": "189c2a87-4f68-4565-944a-1a0fae16e8ba",
        "resource-id": "s1ru27",
        "resource-path": "/recommendationSystem"
    }
}
```

Fonte: Autoria própria (2022).

Código 4 – Trecho da implementação do sistema de recomendação

```
    user_item_m = ratings.pivot('userId','atracaoId','rating').fillna(0)

    X_user = cosine_similarity(user_item_m)

    def atracao_recommender(user_item_m, X_user, user, k=1, top_n=2):
        user_ix = user_item_m.index.get_loc(user)
        user_similarities = X_user[user_ix]
        most_similar_users = user_item_m.index[user_similarities
        .argpartition(-k)[-k:]]
        rec_atracoes = user_item_m.loc[most_similar_users].mean(0)
        .sort_values(ascending=False)
        m_atracoes_visitadas = user_item_m.loc[user].astype(float).gt(0)
        atracoes_visitadas = m_atracoes_visitadas.index[m_atracoes_visitadas]
        .tolist()
        rec_atracoes = rec_atracoes.drop(atracoes_visitadas).head(top_n)
        return rec_atracoes.index.to_frame().reset_index(drop=True
        ).merge(atracoes)

    rec = atracao_recommender(user_item_m, X_user,
    user_to_generate_recommendation)

    return json.loads(rec.to_json(orient='records'))
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

Fonte: Adaptado de Nixon (2020).

5.2 Interface Gráfica do Aplicativo VivaCuritiba

A seguir são apresentadas as telas que compõe a interface gráfica do aplicativo móvel VivaCuritiba. A interface gráfica foi desenvolvida considerando o fluxo de utilização das funcionalidades disponíveis, de forma que seu uso se dê de maneira simples e fluida. A interface é composta por elementos originados de componentes *React Native*, conforme descrito na subseção 4.2.2.1.

5.2.1 Tela Inicial

A tela inicial do aplicativo VivaCuritiba é composta por botões que encaminham o usuário para três funcionalidades do aplicativo, sendo elas:

- Novo Roteiro: ao ser acionado, direciona o usuário para a tela na qual ele poderá inserir os dados de entrada para geração do roteiro turístico;

- Roteiros Salvos: ao clicar neste botão, são apresentados ao usuário roteiros que já tenham sido criados e salvos no sistema;
- Explorar: abre uma tela na qual é exibida uma lista com todos os pontos turísticos disponíveis no sistema.

A tela inicial apresenta também o logotipo do aplicativo, conforme exibido na Figura 11.

Figura 11 – Tela inicial do aplicativo VivaCuritiba.

Fonte: Autoria própria (2022).

5.2.2 Tela de Criação de um Novo Roteiro

Ao clicar no botão “Novo Roteiro” da tela inicial, o usuário é direcionado a uma tela na qual poderá inserir os dados de entrada que guiarão a criação de um novo roteiro.

A Figura 12 apresenta a configuração visual desta tela.

Figura 12 – Tela de criação de um roteiro.

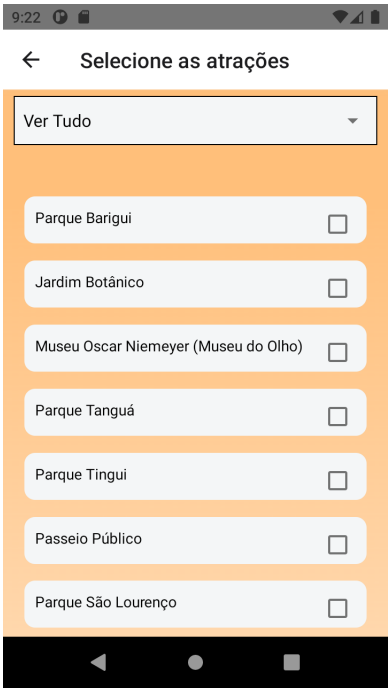
Fonte: Autoria própria (2022).

A seguir são descritos os componentes da tela de criação de um novo roteiro:

- Inserção do ponto de partida: o usuário fornece essa informação através da caixa de texto de um componente *Google Places Autocomplete*;
- seleção de atrações a serem visitadas: ao clicar no botão “Selecionar Atrações”, o usuário é direcionado a uma tela contendo a lista de atrações turísticas disponíveis no sistema. Cada item possui um *checkbox* para indicar os pontos que irão compor o roteiro (ver Figura 13);
- seleção das datas e horários: o botão “Selecionar Datas e Horários” leva o usuário a uma tela em que é definida a agenda, inserindo datas, horários de início e de fim para cada intervalo de tempo em que se deseja executar a rota (ver Figura 14a). Como os locais disponíveis no sistema não são comumente destinados a atividades noturnas, foi estabelecido um limite para os horários possíveis, devendo ser entre 6 e 22 horas. Caso haja uma tentativa de inserir um horário fora desses limites, uma mensagem de erro é exibida (ver Figura 14b) e a ação não é executada. É importante salientar que datas que ultrapassem 8 dias no futuro não terão os aspectos meteorológicos considerados;
- seleção do meio de transporte: são exibidos botões na tela para representar cada um dos meios de transporte disponíveis: “Carro”, “Ônibus”, “Bicicleta” e “A Pé”. Apenas um deles pode ser selecionado e este fica indicado na cor laranja (ver Figura 15). Caso deseje trocar, o usuário pode remover a seleção através de outro clique;

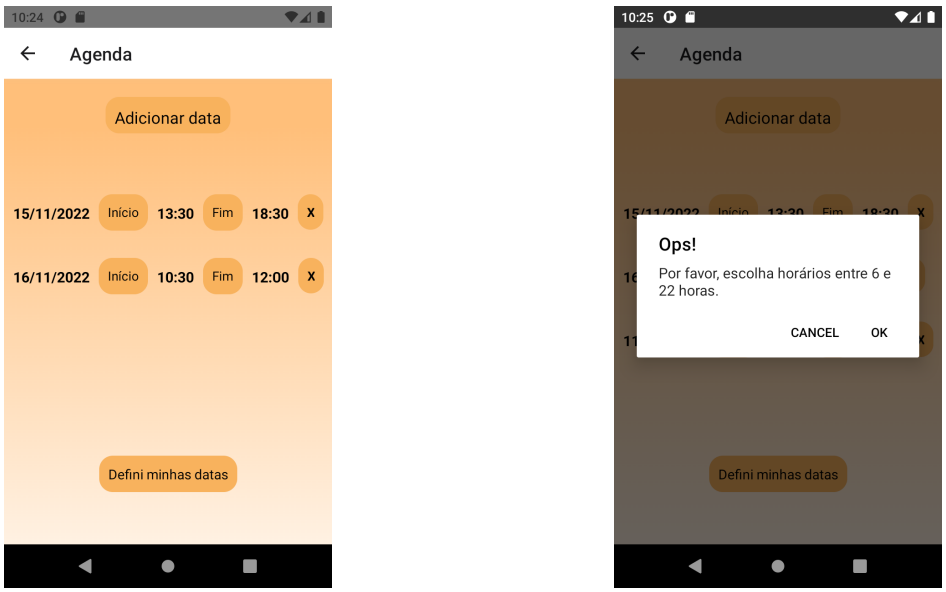
- por fim, o usuário confirma as informações selecionadas através do botão "Criar Roteiro".

Figura 13 – Tela de seleção de atrações.



Fonte: Autoria própria (2022).

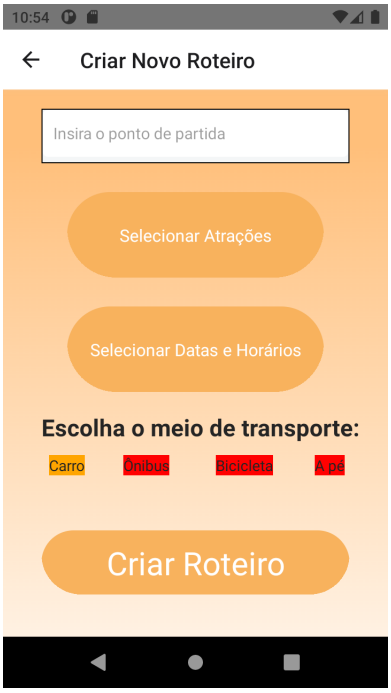
Figura 14 – Tela de agendamento



(a) Datas e horários. **(b) Erro no horário inserido.**

Fonte: Autoria própria (2022).

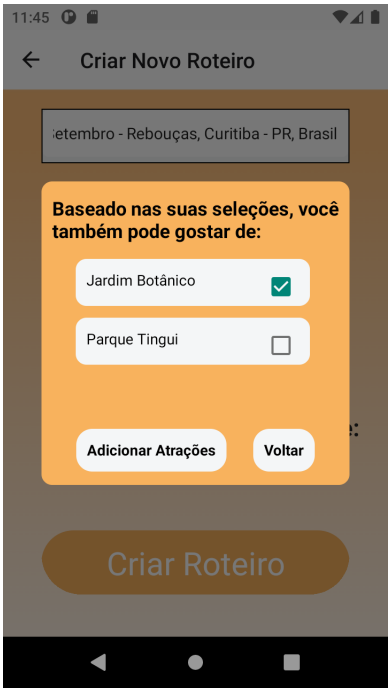
Figura 15 – Tela de indicação do meio de transporte.



Fonte: Autoria própria (2022).

Após finalizar a inserção dos dados e clicar em "Criar Roteiro", é enviada uma requisição ao sistema de recomendação com os dados de identificação do usuário e as atrações escolhidas. A resposta desta chamada é apresentada ao usuário em seguida, indicando os locais sugeridos que podem ser ou não adicionados ao roteiro, conforme indicado na Figura 16.

Figura 16 – Tela de sugestão de atrações.



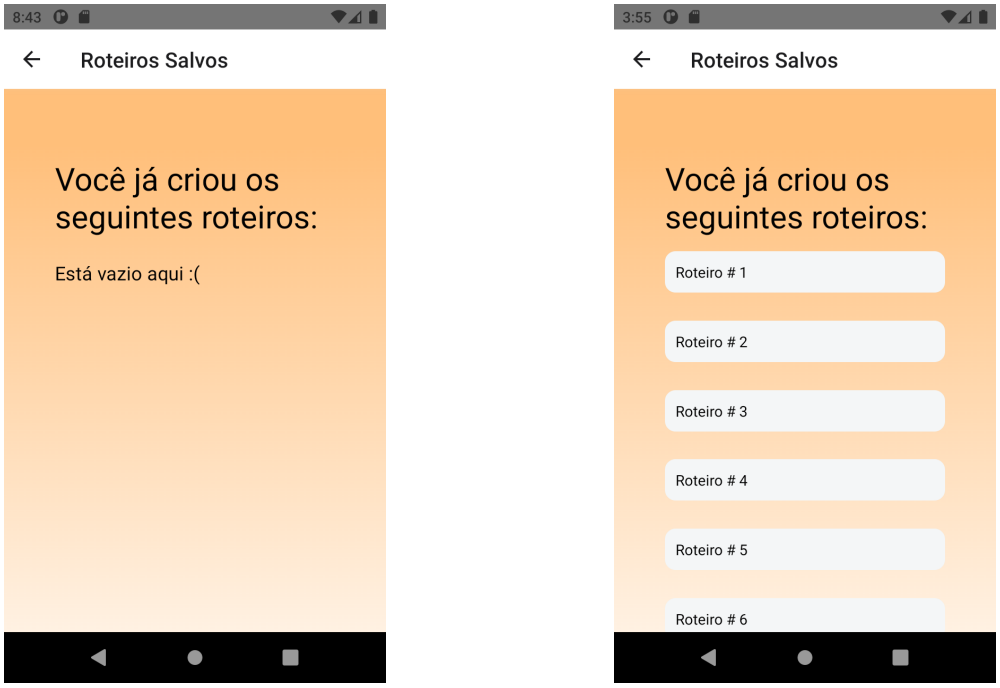
Fonte: Autoria própria (2022).

Ao sair da tela de sugestão, o processo de criação do roteiro é acionado e seus resultados são mostrados ao usuário e salvos no dispositivo.

5.2.3 Tela de Roteiros Salvos

Na tela de roteiros salvos apresentada na Figura 19 são exibidos os roteiros já criados pelo usuário (ver Figura 17b). Caso ainda não exista um roteiro, é exibida uma mensagem, conforme ilustra a Figura 17a.

Figura 17 – Tela de Roteiros Salvos



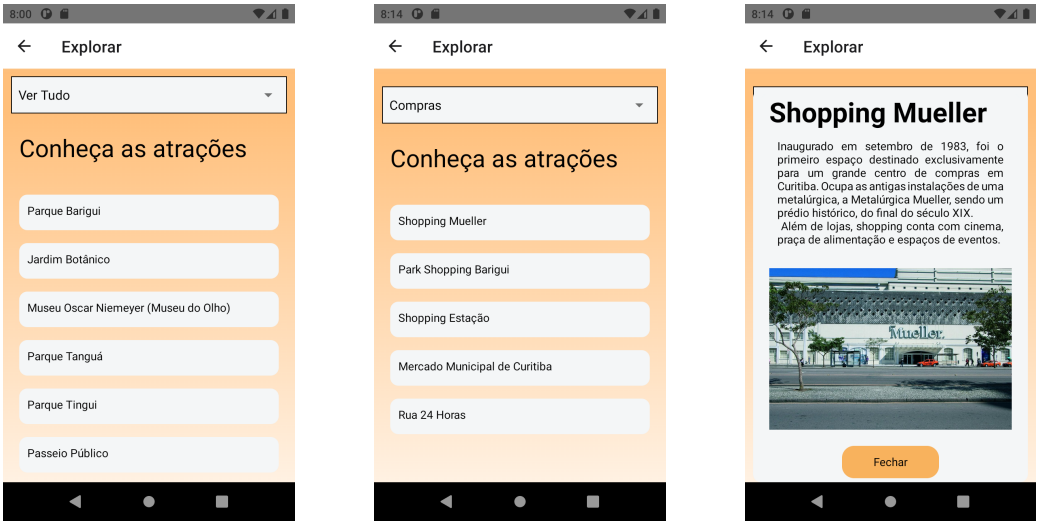
(a) Sem roteiros no sistema. **(b) Roteiros previamente criados.**

Fonte: Autoria própria (2022).

5.2.4 Tela para Explorar Atrações

Ao abrir a tela dedicada para que o usuário conheça os pontos turísticos disponíveis no aplicativo, exibe-se uma lista contendo o nome de cada atração. Cada item que compõe essa lista é clicável, e essa ação abre uma caixa de informações contendo uma breve descrição e uma imagem do local. Existe também um menu para filtrar as atrações de acordo com as categorias criadas no sistema, sendo elas: parques, compras, restaurantes e museus. A Figura 18a apresenta a visão inicial da lista, sem nenhum filtro aplicado. A Figura 18b apresenta o resultado de um filtro de categoria aplicado à lista e, por fim, a Figura 18c exibe a tela de detalhes de uma atração turística.

Figura 18 – Tela Explorar Atrações



(a) Ponto inicial e meio de transporte. **(b) Atrações selecionadas.** **(c) Detalhes sobre a atração selecionada.**

Fonte: Autoria própria (2022).

5.3 Aplicativo VivaCuritiba

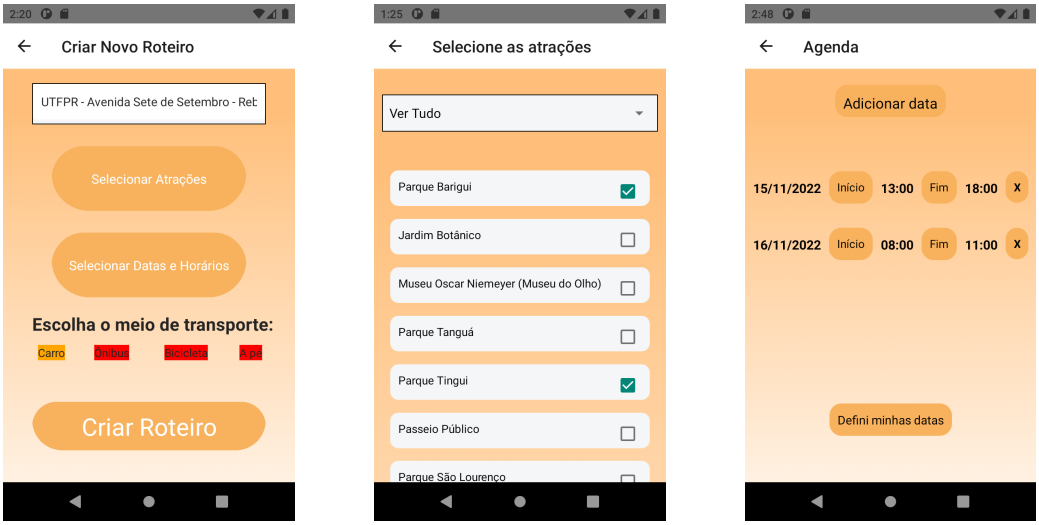
Nesta seção é demonstrada a utilização do aplicativo VivaCuritiba, descrevendo seu fluxo de funcionamento e apresentando o roteiro obtido ao final do processo.

5.3.1 Inserindo Dados

Primeiramente, são definidos os seguintes dados de entrada para o roteiro, conforme apresentado na Figura 19.

- Ponto de origem: UTFPR Campus Curitiba (sede centro) (Figura 19a);
- atrações turísticas desejadas: Parque Barigui, Shopping Estação, Rua 24 Horas e Parque Tingui (Figura 19b);
- meio de transporte: carro (Figura 19a);
- datas e Horários: dia 15/11/2022 das 13 às 18 horas e dia 16/11/2022 das 8 às 11 horas (Figura 19c).

Figura 19 – Utilizando o aplicativo VivaCuritiba



(a) Ponto inicial e meio de transporte. **(b) Atrações selecionadas.** **(c) Datas e horários.**

Fonte: Autoria própria (2022).

5.3.2 Obtendo Recomendações

Ao informar ao sistema que a inserção foi concluída, obtém-se as sugestões resultantes do sistema de recomendação. Dentre elas, o ponto turístico Passeio Público é selecionado para ser adicionado ao roteiro, conforme pode ser visto na Figura 20.

Figura 20 – Sugestões fornecidas pelo sistema de recomendação.

Fonte: Autoria própria (2022).

Dá-se, então, início ao processo de criação do roteiro, que é descrito a seguir.

5.3.3 Fluxo de Execução e Validações

Primeiramente, o ponto inicial é indicado como o local inserido pelo usuário. Em seguida, é calculada a distância entre cada local escolhido através da *Google Directions API* (ver Código 5), cuja resposta também fornece o *Place ID* que será utilizado para a validação dos horários de funcionamento.

Código 5 – Trecho de código para chamada da *Google Directions API*

```
1     const calculateDistanceBetweenTwoPoints = async (origin: string,
2     destination: string) => {
3
4     let neededInfo = {
5         distanceInMeters: -1,
6         destinationPlaceId: "",
7         durationInMinutes: 1
8     }
9
10    const url = 'https://maps.googleapis.com/maps/api/directions/json?
11    origin=' + origin + '&destination=' + destination + '&mode=' +
12    transportTranslate() + '&key=*chave pessoal para utilização
13    da Google API*';
14
15    try {
16        const response = await fetch(url,
17            {
18                method: 'POST'
19            }
20        );
21        const json = await response.json();
22        neededInfo.distanceInMeters =
23        getNumericDistanceInMeters(json.routes[0].legs[0].distance.text);
24        neededInfo.destinationPlaceId = json.geocoded_waypoints[1].place_id;
25        neededInfo.durationInMinutes =
26        getPathDurationInMinutes(json.routes[0].legs[0].duration.text);
27      } catch (error) {
28        console.error(error);
29      }
30      return neededInfo;
31  }
```

Fonte: Autoria própria (2022).

É feita, então, a validação dos horários de funcionamento, que são verificados pela resposta de uma chamada realizada para a *Google Places API* (ver Código 6) com o dado de *Place ID* obtido anteriormente.

Código 6 – Trecho de código para chamada da *Google Places API*

```
    const getPlaceOpeningHours = async (placeId) => {
    let placeInfo = {
        openingHours: {},
        lat: "",
        long: ""
    }
    const url = 'https://maps.googleapis.com/maps/api/place/details/json?
    place_id=' + placeId + '&fields=name,opening_hours,geometry&language=pt
    &key=*chave pessoal para utilização da Google API*';

    try {
        const response = await fetch(url,
            {
                method: 'GET'
            }
        );
        const json = await response.json();
        placeInfo.openingHours = json.result.opening_hours;
        placeInfo.lat = json.result.geometry.location.lat;
        placeInfo.long = json.result.geometry.location.lng;
      } catch (error) {
        console.error(error);
      }
     return placeInfo;
}
```

Fonte: Autoria própria (2022).

Da chamada para a *Google Places API* também são obtidos os dados de latitude e longitude, que são parâmetros de entrada para a requisição que será feita à One Call API (ver Código 7). Essa validação é feita apenas em casos em que o local que está sendo considerado como próximo ponto de visitação for ao ar livre e o intervalo de tempo analisado esteja dentro de um intervalo de 8 dias após a data atual.

Código 7 – Trecho de código para chamada da *One Call API*

```
const openWeatherBaseUrl =
"https://api.openweathermap.org/data/2.5/onecall?";
const getHourlyWeatherForecast = async (lat, long) => {
const url = openWeatherBaseUrl + "lat=" + lat + "&lon=" + long +
"&exclude=current,minutely,daily,alerts&units=metric&appid=" + myAPIKey;
let hourly: any[] = [];
let hourlyConditions: any[] = [];
try {
    const response = await fetch(url);
    const json = await response.json();
    hourly = json.hourly;
  } catch (error) {
    console.error(error);
  }
  for (let i = 0; i < hourly.length; i++) {
    let weatherCode = hourly[i].weather[0].id;
    let weatherConditionGood = weatherCode < 800 ? false : true;
    let obj = {
        data: convertUnixTimestampToDate(hourly[i].dt),
        isWeatherGood: weatherConditionGood
    }
    hourlyConditions.push(obj);
  }
  return hourlyConditions;
}

const convertUnixTimestampToDate = (timestamp) => {
    return new Date(timestamp*1000);
}
```

Fonte: Autoria própria (2022).

Caso seja verificado que alguma dessas condições não é atendida, o local é descartado como opção de próximo ponto de visitação. Esse processo é aplicado a todos os pontos ainda não adicionados ao roteiro. Para a definição de qual será o próximo local a ser adicionado, é comparada a distância do ponto atual a cada candidato a próximo. Caso seja verificado que a distância sendo analisada é menor do que a distância ao candidato que ocupa a posição temporária de próximo ponto, o próximo ponto é atualizado e recebe o candidato cuja distância é menor. Esse fluxo pode ser verificado no Código 8.

Código 8 – Trecho de código para definição da próxima atração do roteiro.

```
    if (placeIsOpen && weatherConditionsOk) {
        if (atracaoArrayIndex == 0) {
            shortestPath = possibleShortestPath;  // inicializa candidato a
                                                  // próximo com o primeiro ponto
            nextNode = newAtracaoArray[atracaoArrayIndex];
            pathDuration = placesInfo.durationInMinutes;
        } else if (possibleShortestPath < shortestPath) {
                shortestPath = possibleShortestPath;
                nextNode = newAtracaoArray[atracaoArrayIndex];
                pathDuration = placesInfo.durationInMinutes;
        }
    }
```

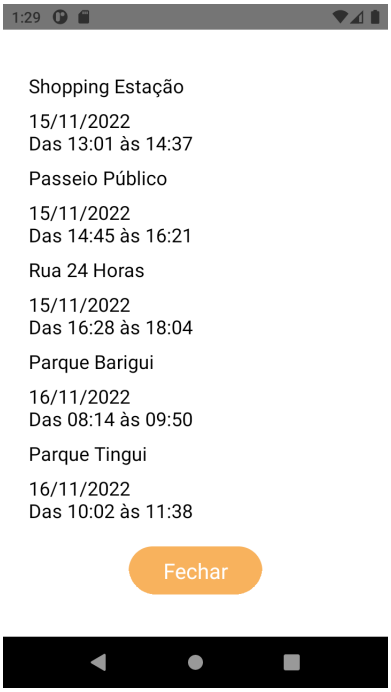
Fonte: Autoria própria (2022).

Para a definição do intervalo de tempo de cada visitação, inicialmente é calculada a média de tempo para cada atração dividindo-se o tempo total definido pelo usuário pelo número total de atrações selecionadas. A cada definição de ponto que será visitado, é acrescido o tempo gasto no deslocamento para verificar a data de início e fim de cada visitação.

Ao fim da definição de ordem de visitação para os pontos que não apresentaram problemas com horário e previsão do tempo, as atrações restantes, caso existam, são incluídas no roteiro seguindo apenas o critério de menor distância.

Por fim, o resultado do roteiro construído é apresentado ao usuário, conforme pode ser visto na Figura 21, e salvo para consulta posterior.

Figura 21 – Roteiro construído



Fonte: Autoria própria (2022).

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho de conclusão de curso foi desenvolvido e apresentado um sistema de geração de roteiros turísticos para a cidade de Curitiba. Esse sistema foi implementado para a utilização em dispositivos móveis com sistema operacional Android.

Através de informações simples fornecidas pelo usuário, como seleção das atrações a serem visitadas, datas e horários desejados para a realização do roteiro turístico e meio de transporte a ser utilizado para o deslocamento entre as atrações, o sistema é capaz de produzir um roteiro turístico otimizado. Essa otimização considera o menor caminho entre as atrações selecionadas, de modo que estas possam ser visitadas percorrendo a menor distância possível, condições climáticas para evitar que visitações em atrações ao ar livre sejam feitas em condições de mau tempo, além dos horários de funcionamento de cada atração para evitar que a visita seja agendada em um horário em que esta esteja fechada.

O fato de dispositivos móveis serem uma constante presente no dia-a-dia das pessoas também possibilitou a concepção de um sistema de recomendação colaborativa que se baseia em informações fornecidas por outros usuários do sistema para gerar sugestões personalizadas a um dado usuário, a partir da interação deste com o aplicativo na qual são feitas escolhas de atrações turísticas dentre um universo de opções existentes.

A partir da definição do projeto de *software*, no qual foram especificados o escopo, casos de uso e requisitos funcionais e não funcionais, foi possível realizar uma análise entre as possíveis abordagens para a obtenção do resultado esperado. Deste modo, foram estudadas diferentes tecnologias e modelos de implementação, tendo sido escolhidas as opções mais adequadas ao alcance do objetivo proposto.

Como resultado desse processo de pesquisa, desenvolveu-se um aplicativo móvel utilizando tecnologias que possibilitam uma implementação multi-plataforma, porém com foco no sistema operacional Android devido à sua popularidade e também ao fato de o seu ambiente de desenvolvimento ser mais acessível. Este aplicativo utiliza os recursos tecnológicos dos dispositivos móveis para prover a interface entre o usuário e o sistema de recomendação colaborativa, o qual foi desenvolvido e hospedado na plataforma de nuvem AWS.

A seguir são descritos os trabalhos futuros propostos para o sistema implementado.

6.1 Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- fazer as adaptações necessárias para que o sistema possa ser executado em dispositivos móveis com o sistema operacional iOS da Apple Inc.;
- aumentar a robustez do sistema acionando as versões pagas das tecnologias utilizadas, como os serviços AWS, *Google Maps* e *One Call API*;

- introduzir um sistema de autenticação para que os usuários possam acessar seus perfis pessoais em diferentes dispositivos móveis;
- adicionar mensagens e/ou símbolos para informar ao usuário quando o sistema estiver aguardando a resposta de solicitações feitas a serviços remotos;
- expandir o leque de opções disponíveis para pontos de visitação, adquirindo esses dados *online* de modo que sejam aplicáveis a qualquer cidade onde o usuário esteja e não apenas Curitiba;
- incluir o disparo de mensagens de notificação quando o horário da próxima visita turística estiver se aproximando;
- incluir uma opção no aplicativo para que o usuário possa solicitar que os roteiros sejam recalculados, de modo a atualizar os dados sobre a previsão do tempo;
- aprimorar a interface gráfica do aplicativo, buscando uma melhor usabilidade.

REFERÊNCIAS

AMAZON. **Amazon API Gateway**. 2022. Disponível em: https://aws.amazon.com/pt/api-gateway/. Acesso em: 06 nov. 2022.

AMAZON. **Amazon S3**. 2022. Disponível em: https://aws.amazon.com/pt/s3/. Acesso em: 06 nov. 2022.

AMAZON. **AWS Lambda**. 2022. Disponível em: https://aws.amazon.com/pt/lambda/. Acesso em: 06 nov. 2022.

AMAZON. **O que é AWS?** 2022. Disponível em: https://aws.amazon.com/pt/what-is-aws. Acesso em: 07 nov. 2022.

ANDROID. **Compreendendo o Android**. 2022. Disponível em: https://www.android.com/everyone/facts. Acesso em: 04 nov. 2022.

ANDROID. **Interaction in Android**. 2022. Disponível em: https://source.android.com/docs/core/interaction. Acesso em: 04 nov. 2022.

BLACK, P. E. **shortest path, in Dictionary of Algorithms and Data Structures [online]**. 2020. Disponível em: https://www.nist.gov/dads/HTML/shortestpath.html. Acesso em: 4 nov. 2022.

COLLEGE, C. **Memory-based algorithms**. 2022. Disponível em: https://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/memorybased.html. Acesso em: 06 nov. 2022.

COLLEGE, C. **Model-based recommendation systems**. 2022. Disponível em: https://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html. Acesso em: 06 nov. 2022.

CORMEN, T. H. *et al.* **Algoritmos**: Teoria e prática. [*S.l.*]: Editora Elsevier, 2009. 470-471 p.

CURITIBA, P. de. **Plano Municipal de Turismo de Curitiba**. 2015. Disponível em: https://mid-turismo.curitiba.pr.gov.br/2015/11/pdf/00000817.pdf. Acesso em: 27 ago. 2022.

CURITIBA, P. de. **Instituto Municipal de Turismo - Curitiba Turismo**: Dados e estatísticas. 2022. Disponível em: https://turismo.curitiba.pr.gov.br/conteudo/dados-e-estatisticas/1724. Acesso em: 27 ago. 2022.

GOOGLE. **Google Maps**. 2022. Disponível em: https://www.google.com.br/maps. Acesso em: 03 set. 2022.

GOOGLE. **Sobre o Google Maps**. 2022. Disponível em: https://www.google.com/maps/about. Acesso em: 04 nov. 2022.

GRACHET, M. **Understanding the React Native bridge concept by @mfrachet**. 2022. Disponível em: https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8. Acesso em: 01 set. 2022.

IBM. **Machine Learning**. 2022. Disponível em: https://www.ibm.com/br-pt/cloud/learn/machine-learning. Acesso em: 06 nov. 2022.

INSTITUTE, P. **Python® – the language of today and tomorrow**. 2022. Disponível em: https://pythoninstitute.org/about-python. Acesso em: 07 nov. 2022.

ISINKAYE, F.; FOLAJIMI, Y.; OJOKOH, B. Recommendation systems: Principles, methods and evaluation. **Egyptian Informatics Journal**, v. 16, n. 3, p. 261–273, 2015. ISSN 1110-8665. Disponível em: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

LEVINAS, C. A. **An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals**. set. 2014. 72 p. Dissertação (Mestrado) — Master in Artificial Intelligence - Universitat Politècnica de Catalunya, set. 2014.

MATHIESON, A.; WALL, G. **Tourism**: Economic, physical and social impacts. [*S.l.*]: Longman, 1982.

MEDEIROS, I. R. G. **Estudo sobre Sistemas de Recomendação Colaborativos**. 2013. 30 p.

MOZILLA. **JavaScript**. 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Glossary/JavaScript. Acesso em: 07 nov. 2022.

NARDI, E.; AUGUSTINHO, J. **DESENVOLVIMENTO DE UM APLICATIVO ANDROID PARA GERAÇÃO DE ITINERÁRIO TURÍSTICO EM CURITIBA**. 2018. 2 p.

NIXON, A. E. **Building a memory based collaborative filtering recommender**. 2020. Disponível em: https://towardsdatascience.com/how-does-collaborative-filtering-work-da56ea94e331. Acesso em: 07 nov. 2022.

OPENWEATHER. **Weather Conditions**. 2022. Disponível em: https://openweathermap.org/weather-conditions#Weather-Condition-Codes-2. Acesso em: 10 nov. 2022.

PLATFORMS, I. M. **Component State**. 2022. Disponível em: https://reactjs.org/docs/faq-state.html. Acesso em: 04 nov. 2022.

PLATFORMS, I. M. **Core Components and APIs**. 2022. Disponível em: https://reactnative.dev/docs/components-and-apis. Acesso em: 12 nov. 2022.

PLATFORMS, I. M. **React Fundamentals**. 2022. Disponível em: https://reactnative.dev/docs/intro-react. Acesso em: 04 nov. 2022.

PLATFORMS, I. M. **React Native**. 2022. Disponível em: https://reactnative.dev. Acesso em: 04 nov. 2022.

PLATFORMS, I. M. **Rendering Elements**. 2022. Disponível em: https://reactjs.org/docs/rendering-elements.html. Acesso em: 04 nov. 2022.

PLATFORMS, I. M. **Setting up the development environment**. 2022. Disponível em: https://reactnative.dev/docs/environment-setup. Acesso em: 04 nov. 2022.

POLO, G. **O que é um web service?** 2018. Disponível em: https://gabrielpolo.medium.com/o-que-%C3%A9-um-webservice-c5104d847a85. Acesso em: 07 nov. 2022.

POSTMAN. **About Postman**. 2022. Disponível em: https://www.postman.com/company/about-postman/. Acesso em: 13 nov. 2022.

PROJECT, A. O. S. **Arquitetura da plataforma**. 2020. Disponível em: https://developer.android.com/guide/platform. Acesso em: 17 set. 2022.

REACT, A. **React Native Get Unique ID of Device**. 2022. Disponível em: https://aboutreact.com/react-native-get-unique-id-of-device/. Acesso em: 07 nov. 2022.

RICCI, F.; ROKACH, L.; SHAPIRA, B. **Recommender Systems Handbook**. [*S.l.*]: Springet, 2011.

ROADTRIPPERS. **Roadtrippers Media Center**. 2021. Disponível em: https://roadtrippers.com/media-center/. Acesso em: 03 set. 2022.

ROADTRIPPERS. **About - Roadtrippers**. 2022. Disponível em: https://roadtrippers.com/about. Acesso em: 04 nov. 2022.

SAFI, F.; YASKEVICH, M.; PONTES, G. **Google Maps Search Component for React Native**. 2015. Disponível em: https://github.com/FaridSafi/react-native-google-places-autocomplete. Acesso em: 01 out. 2022.

STATCOUNTER. **Mobile Operating System Market Share Worldwide**. 2022. Disponível em: https://gs.statcounter.com/os-market-share/mobile/worldwide. Acesso em: 12 set. 2022.