

COMPUTAÇÃO PARA DISPOSITIVOS MÓVEIS

CAPÍTULO 4 - COMO APRIMORAR NOSSO APLICATIVO COM AS APIS DA GOOGLE?

Fernando Skackauskas Dias

INICIAR

Introdução

É notório o grande avanço dos aplicativos nos dispositivos móveis, as novas funcionalidades, a usabilidade e capacidade de comunicação e interação com outros dispositivos e componentes. As tecnologias de geolocalização e mapeamento estão levando à criação de aplicativos que atendem às necessidades de diversas empresas e usuários, que, há poucos anos, eram inimagináveis. Nesse contexto, os serviços em nuvem também estão cada vez mais presentes no dia a dia. O Android tem inserido, dentre as suas possibilidades, a interação com os serviços *cloud*, como a execução de *backup* e serviços de mensagens por meio de APIs. E como a API trabalha? Por que as APIs são tão importantes para os desenvolvedores?

Uma API (*Application Programming Interface*, ou, em português, Interface de Programação de Aplicativos) trata do conjunto de aplicações que permitem a

construção de uma interface inteligente. Ou seja, a API é o motor dos aplicativos, pois ela é quem faz a ligação de todos os recursos necessários para que um *software* funcione de forma correta para o usuário. A empresa Google oferece várias APIs, sendo que um dos principais destaques é o Google *Maps*. Este é um serviço gratuito, que pode ser utilizado em qualquer aplicativo de dispositivo móvel. Mas como utilizar o Google *Maps* em aplicativos?

Neste capítulo, vamos entender como o Android utiliza a geolocalização e o mapeamento, por meio do Google *Maps*. Estudaremos o Android e a utilização dos serviços em nuvens, como o Android *Data Backup*, o Firebase e o *Cloud*, do Service Manager, e as funcionalidades de sensores e toque. Por fim será demonstrada a construção de um *software* para aplicativo, utilizando as funcionalidades do Android.

Vamos estudar com atenção!

4.1 Geolocalização e mapas

A geolocalização é um conjunto integrado de *hardware*, *software* e processos, que fornecem a localização baseada em coordenadas geográficas. A geolocalização utiliza, de forma geral, o Sistema de Posicionamento Global (GPS). A posição geográfica pode ser exibida em mapas virtuais e, dependendo do nível de detalhamento das coordenadas, pode-se chegar até a um determinado endereço completo.

Além do GPS, a geolocalização também pode ser identificada por meio de um endereço IP, endereço de controle de acesso à mídia (MAC), sistemas de radiofrequência (RF), dados EXIF (*Exchangeable Image File Format*) e outros sistemas de posicionamento sem fio. Conforme explicado por Felipe e Dias (2014, p. 2), “a latitude e longitude são disponíveis através da página via JavaScript, que por sua vez, pode ser enviada a um servidor *web* e fazer coisas como encontrar locais ao seu redor ou mostrar sua posição em um mapa”. Uma das grandes utilidades da geolocalização é a capacidade de exibir, em tempo real, o posicionamento geográfico em mapas digitais. Esta possibilidade tem sido utilizada por vários aplicativos para controle de tráfego, auxílio para a locomoção em vias públicas e diversas funcionalidades adicionais.

VOCÊ QUER LER?

Um ramo da área de automação que tem crescido consideravelmente é a automação residencial. Com a evolução tecnológica da eletrônica e da Realidade Virtual, a automação residencial tem se tornado mais acessível, possibilitando conforto para uma maior quantidade de pessoas. Fernandes e Barbosa (2017) desenvolveram um trabalho que tem como objetivo, apresentar o desenvolvimento de um sistema de Automação Residencial e Realidade Virtual utilizando o dispositivo Myo, e também uma versão *mobile* em Android, unindo as áreas da saúde, engenharia e tecnologia. Leia mais em <http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf (http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf)>.

Com as possibilidades que se abrem a partir da geolocalização, surgem muitos questionamentos para motivar os trabalhos: como é possível utilizar as funcionalidades de geolocalização em dispositivos móveis? Quais as ferramentas de dispositivos móveis que auxiliam no desenvolvimento de aplicativos de geolocalização e mapas? Como o sistema operacional Android fornece os componentes para a geolocalização e mapeamento?

Para começar a responder essas questões, vamos fazer observações sobre o Android. Deitel, Deitel e Wald (2016, p. 3) apontam que “uma vantagem de desenvolver aplicativos Android é a sua fraqueza (ou grau de abertura) da plataforma. O sistema operacional é de código-fonte aberto e gratuito”.

A seguir serão mostrados e analisados os componentes, classes e bibliotecas disponíveis para a implantação de aplicativos em dispositivos móveis, com a utilização da geolocalização e mapas interativos.

4.1.1 Location API

Os aplicativos baseados em localização e mapas oferecem uma experiência atraente em dispositivos móveis. É possível criar esses recursos no seu aplicativo usando as classes do `android.location` e a API do Google *Maps* para Android. O componente central da estrutura de localização é o `LocationManager` do sistema, que fornece APIs para determinar a localização e o porte do dispositivo subjacente (se disponível).

Como com outros serviços do sistema, não se instancia um `LocationManager` diretamente. Em vez disso, é solicitada uma instância do sistema chamando

getService(). O método retorna um identificador para uma nova instância LocationManager. Depois que o aplicativo obter um LocationManager, será possível realizar:

- consulta à lista de todas as instâncias do LocationProvider, para determinar o último local do usuário conhecido;
- registrar ou cancelar o registro de atualizações periódicas da localização atual do usuário de um provedor de localização;
- registrar ou cancelar o registro, para que um dado seja acionado se o dispositivo estiver dentro de uma determinada proximidade (especificada por raio em metros) de uma determinada latitude/longitude.

O API do Google *Maps*, disponível para o sistema operacional Android, possibilita incluir mapas aos aplicativos residentes no dispositivo. O API tem a capacidade de interagir diretamente com os servidores do Google *Maps*, permitindo fazer o *download* das informações de geolocalização, exibição de mapas virtuais e acesso aos mapas por meio de toque. É possível, também, incluir marcadores e manipular as visualizações específicas de uma determinada área do mapa.

VOCÊ SABIA?

Os deficientes visuais encontram enormes desafios no seu dia a dia, como a limitação para perceber o entorno dos objetos e para aquisição de produtos. Para promover a autonomia dos deficientes visuais, foi desenvolvida uma solução na área de Engenharia Elétrica, para aparelhos Android, que orienta o usuário por meio da tecnologia *Near Field Communication*. Leia mais no artigo escrito por Lamas e Souza (2016): <<https://periodicos.ufsc.br/index.php/extensio/article/view/1807-0221.2016v13n24p37> (<https://periodicos.ufsc.br/index.php/extensio/article/view/1807-0221.2016v13n24p37>)>.

A API do Google *Maps* disponibiliza a classe principal MapView. Esta classe tem como função exibir, em um determinado mapa, as coordenadas e dados oriundos

do Google *Maps*. É possível fazer o *zoom* no mapa virtual pelo toque, inclusive interagindo com mapas adicionais, oferecendo os elementos de interface para o usuário. O aplicativo também pode usar a classe *MapView* para controlar o mapa de forma programática e desenhar várias sobreposições na parte superior do mapa.

Conforme demonstrado por Murukannaiah e Singh (2015), utilizando-se o *MapView*, um lugar deriva seu significado do espaço físico, das atividades ou do contexto social de um usuário. Dessa maneira, o local pode facilitar uma melhor experiência do usuário em comparação com a representação tradicional de localização, que são coordenadas espaciais.

VOCÊ O CONHECE?

Quando a Google comprou o Android, Nick Sears, cofundador, esperava o desenvolvimento de um sistema operacional avançado, como, por exemplo, câmeras digitais. No entanto, houve uma grande reviravolta com as tecnologias que vieram a seguir. Quer ler mais sobre o cofundador do Android? Leia no artigo escrito por Moliane e Almeida (2016): <<https://docplayer.com.br/54762841-Art-e-dalvik-em-dispositivos-android.html> (https://docplayer.com.br/54762841-Art-e-dalvik-em-dispositivos-android.html)>.

As APIs do Google *Maps* para Android não estão incluídas na plataforma Android, mas estão disponíveis em qualquer dispositivo com o Google *Play Store*, executando o Android 2.2 ou superior, por meio dos serviços do Google *Play*. Portanto, para integrar o Google *Maps* ao aplicativo, é necessário instalar as bibliotecas do Google *Play Services* para o Android SDK.

4.1.2 Google Maps API

É possível para os desenvolvedores de aplicativos para dispositivos móveis incorporarem as diversas funcionalidades disponíveis pelo Google *Maps* nas páginas *web* ou obter informações do Google *Maps*. Isto torna o resultado do aplicativo mais personalizado. Neste sentido, a Google oferece um conjunto de interfaces de programação de aplicativos (APIs) desenvolvidas, que permitem a comunicação com o Google *Services* e sua integração a outros serviços. Exemplos

disso incluem: pesquisa, Gmail, tradutor ou Google *Maps*. Aplicativos de terceiros podem usar essas APIs para aproveitar ou ampliar a funcionalidade dos serviços existentes.

As APIs fornecem funcionalidades como análise, aprendizado de máquina como um serviço (a API de previsão) ou acesso a dados do usuário (quando a permissão para ler os dados é fornecida). Outro exemplo importante é um mapa do Google incorporado em um *site*, que pode ser obtido usando a API de mapas estáticos, a API do Google *Places* ou a API do Google *Earth*.

VOCÊ QUER LER?

Veja como foi desenvolvida uma aplicação em Android, utilizando um sistema de localização geográfica, para determinar pontos turísticos na cidade de Curitiba, a fim de facilitar o trânsito dos turistas (EGGEA, 2013). Leia mais em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/2484/1/CT_TECJAVMOV_I_2012_10.pdf (http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/2484/1/CT_TECJAVMOV_I_2012_10.pdf)>.

O Google Apps *Script* é uma plataforma em JavaScript baseada em nuvem, que permite aos desenvolvedores escrever *scripts* que podem manipular APIs de serviços, como Calendário, Documentos, Drive, Gmail e Planilhas, e criar facilmente complementos para esses serviços com aplicativos baseados no navegador Chrome. As funcionalidades mais comuns são:

- o registro do usuário é, geralmente, feito por meio do *login* do Google+, que permite que os usuários façam *login* em serviços de terceiros com a conta do Google+, usando a API do Google+. Isso está disponível no Android, iOS ou em JavaScript;
- os aplicativos do Google Drive são vários aplicativos da *web* (geralmente de terceiros) que funcionam usando a Google Drive API;
- a pesquisa personalizada permite que os desenvolvedores da *web* forneçam uma pesquisa em seu próprio *site* incorporando uma caixa de pesquisa personalizada e usando a API de pesquisa personalizada;

- os aplicativos do Google App *Engine*, são aplicativos da *web* que são executados no Google App *Engine*, que é uma plataforma de computação em nuvem de plataforma como serviço (PaaS), que permite que os desenvolvedores da *web* executem seus *websites* nos *data centers* do Google;
- os *gadgets* são mini-aplicativos criados em HTML, JavaScript, Flash e Silverlight, que podem ser incorporados em páginas da *web* e outros aplicativos. Eles podem ser executados em vários *sites* e produtos.

O Google Loader é uma biblioteca JavaScript que permite que os desenvolvedores da *web* carreguem facilmente outras APIs JavaScript fornecidas pela Google e por outros desenvolvedores de bibliotecas populares. O Google *Loader* fornece um método JavaScript para carregar uma API específica (também chamada de módulo), no qual, configurações adicionais podem ser especificadas, como versão da API, idioma, localização, pacotes selecionados, retorno de chamada de carga e outros parâmetros específicos de uma API específica.

4.2 Serviços em nuvem

Os serviços em nuvem indicam inúmeros recursos de TI, que estão disponíveis na internet. Esta expressão também engloba a oferta de serviços para a seleção, implantação e gerenciamento constante de vários recursos. Os serviços de computação em nuvem abrangem muitos recursos que um provedor de serviços oferece aos clientes pela internet, como o armazenamento e processamento de dados em um servidor online. A nuvem “permite que uma grande diversidade de dispositivos (*tablets*, celulares, *notebooks* e *desktops*), possa acessar e executar esses recursos, sendo necessário somente o acesso à internet e um mecanismo padronizado, que por sua vez pode ser um navegador que necessita poucos recursos computacionais” (VANDRESEN; MAGALHÃES, 2013, p. 1).

As características dos serviços em nuvem incluem autoprovisionamento e elasticidade; ou seja, os clientes podem provisionar serviços sob demanda e desligá-los, quando não forem mais necessários. Além disso, os clientes geralmente assinam serviços em nuvem, por exemplo, em um acordo de cobrança mensal, para não ser necessário pagar por licenças de *software*. Em muitas

transações, essa abordagem torna uma tecnologia baseada em nuvem uma despesa operacional, em vez de uma despesa de capital (VANDRESEN; MAGALHÃES, 2013).

A tecnologia em nuvem oferece para os desenvolvedores, organizações e usuários o acesso a diversos *softwares*, armazenamento e demais funcionalidades de infraestrutura de TI sem nenhum custo de manutenção e atualização.

O uso de serviços em nuvem tornou-se estreitamente associado a ofertas comuns de nuvem, como *software* como serviço (SaaS), plataforma como serviço (PaaS), e infraestrutura como serviço (IaaS), baseados em nuvem (VANDRESEN; MAGALHÃES, 2013).

Neste sentido, como o sistema operacional Android utiliza os serviços em nuvem? Quais são as funcionalidades? Como são incorporados aos aplicativos? Estas questões serão exploradas a seguir.

4.2.1 Android data backup

O *backup* automático para aplicativos faz a cópia automática de aplicativos, que é executado no Android 6.0 (nível 23 da API) ou posterior. O Android preserva os dados do aplicativo fazendo o *upload* para o Google *Drive* do usuário, no qual ele é protegido pelas credenciais da Conta do Google. A quantidade de dados é limitada a 25 MB por usuário do seu aplicativo e não há custos para armazenar dados de *backup*. O aplicativo pode personalizar o processo de *backup* ou desativar os *backups*.

Por padrão, o *backup* automático inclui arquivos na maioria dos diretórios atribuídos ao aplicativo pelo sistema:

- arquivos de preferências compartilhados;
- arquivos salvos no armazenamento interno do aplicativo, acessado por `getFilesDir()` ou `getDir()`;
- arquivos no diretório retornado por `getDatabasePath()`, que também inclui arquivos criados com a classe `SQLiteOpenHelper`;
- arquivos no armazenamento externo, no diretório retornado por `getExternalFilesDir()`.

O *backup* exclui arquivos em diretórios retornados por `getCacheDir()`,

`getCodeCacheDir()`, ou `getNoBackupFilesDir()`. Os arquivos salvos nesses locais são necessários apenas temporariamente ou são intencionalmente excluídos das operações de *backup*.

4.2.2 *Firebase*

O Firebase é um API da Google que tem como objetivo enviar mensagens de notificações e dados de forma confiável. O Firebase é um Firmware *Cloud Messaging* (FCM), no qual é possível, por exemplo, executar uma notificação no aplicativo do usuário informando que *e-mail*, ou outros dados, estão disponíveis.

Portanto, é possível enviar mensagens de notificação para fazer a interação entre usuários. O pré-requisito para configurar o Firebase é, pelo menos, ter uma conta do Google para entrar no sistema.

4.2.3 *Cloud to Device Message e Google Cloud Messaging*

O Android *Cloud* para mensagens de dispositivo (comumente chamado de mensagens em nuvem para dispositivos) ou C2DM, é um serviço de notificação móvel extinto que foi desenvolvido pela Google e substituído pelo serviço Google *Cloud Messaging*. Ele permitiu que os desenvolvedores enviassem dados de servidores para aplicativos Android e extensões do Chrome. O C2DM foi lançado originalmente em 2010, e estava disponível a partir da versão 2.2 do Android.

Em 27 de junho de 2012, a Google revelou o serviço do Google *Cloud Messaging* para substituir o C2DM, citando melhorias na autenticação e, na entrega, novos pontos da API e parâmetros de mensagens, a remoção de limites de taxa da API e de tamanhos máximos de mensagens.

O Google *Cloud Messaging* é um serviço gratuito que permite aos desenvolvedores enviar mensagens entre servidores e aplicativos clientes. Isso inclui mensagens *downstream* de servidores para aplicativos clientes e mensagens *upstream* de aplicativos clientes para servidores. Por exemplo, uma mensagem poderia informar um aplicativo cliente de que há novos dados a serem buscados no servidor, como no caso de uma notificação de "novo *e-mail*". Como mensagens instantâneas, uma mensagem do GCM pode transferir até 4 kbytes para o aplicativo cliente. O serviço GCM lida com todos os aspectos de enfileiramento de mensagens e entrega para o aplicativo cliente de destino.

VOCÊ SABIA?

Existe uma ciência chamada de “ciência do contexto”, que leva em consideração todo o entorno e contexto, no qual o usuário está inserido. Baseando-se na ciência do contexto, foi desenvolvido o aplicativo Accessibility, que tem como função disponibilizar informações que auxiliam a locomoção de usuários portadores de necessidades especiais (SANTOS, *et al.*, 2015). Você por se aprofundar neste assunto lendo o artigo “Dispositivos móveis e usuários cegos: recomendações de acessibilidade em discussão” (MACHADO; MACHADO; CONFORTO, 2014): < http://www.tise.cl/volumen10/TISE2014/tise2014_submission_231.pdf (http://www.tise.cl/volumen10/TISE2014/tise2014_submission_231.pdf)>.

A suspensão oficial do serviço C2DM se deu em agosto de 2012, pouco depois de anunciar o serviço do Google *Cloud Messaging*. A Google, então, publicou documentação para orientar os desenvolvedores de aplicativos a migrar do C2DM para o novo serviço.

A migração para o serviço exigiu mudanças no SDK e no código, bem como a liberação de uma atualização de aplicativo para o repositório de publicação (como o Google *Play*) para *download* e atualização. O C2DM e o serviço do Google *Cloud Messaging* não eram interoperáveis entre si; não era possível enviar solicitações de dados usando um serviço para ser recebido e processado, no aplicativo cliente, usando o outro. A migração também exigia que alterações fossem feitas no servidor de terceiros, operado pelo desenvolvedor (dependendo da complexidade e do caso de uso em relação aos dados enviados).

4.3 Sensores e toque

Um sensor de toque é um tipo de equipamento que captura e grava o toque físico em um dispositivo. Ele permite que um dispositivo ou objeto detecte o toque, geralmente por um usuário ou operador humano. Um sensor de toque também pode ser chamado de detector de toque.

Ao contrário de um botão ou outro controle mais manual, os sensores de toque são mais sensíveis e geralmente respondem de maneira diferente a vários tipos de toque, como tocar ou arrastar. Os sensores de toque são usados em dispositivos

de tecnologia de consumo, como smartphones e *tablets*. Normalmente, os sensores de toque são usados como um meio de receber informações do usuário. Para que se possam extrair ao máximo as funcionalidades de sensores e toques, é necessária uma estrutura com câmera, sensores de movimento e presença posicionados estrategicamente no ambiente (SOUZA, 2012).

Cada traço físico que um sensor de toque grava é enviado para uma unidade de processamento/*software*, que o processa de acordo. Por exemplo, ao navegar por um *smartphone* ou usar um aplicativo, o sensor de toque captura os toques humanos ou a pressão aplicada na tela. Com o sistema operacional Android utiliza estes recursos? Como é possível implantar nos aplicativos? Quais são os seus componentes? Estas questões serão vistas a seguir.

4.3.1 Câmera API

A estrutura do Android inclui suporte para vários recursos de câmeras disponíveis nos dispositivos, permitindo que se capturem fotos e vídeos em aplicativos. Será realizada uma abordagem sobre a captura de imagens e vídeos, descrevendo as aplicações de câmera personalizadas para os usuários.

Antes de ativar o aplicativo para usar câmeras em dispositivos Android, deve-se considerar algumas perguntas sobre como seu aplicativo pretende usar esse recurso de *hardware*.

- Requisito da câmera: o uso de câmera é um requisito fundamental para determinados aplicativos, sendo uma condição necessária para a instalação. Nesse caso, deve-se declarar o requisito de câmera.
- Imagem rápida ou câmera personalizada: como o aplicativo usará a câmera?
- Armazenamento: as imagens ou vídeos de seu aplicativo devem ser visíveis apenas para o aplicativo ou compartilhados, para que outros aplicativos, como o Gallery ou outras mídias e aplicativos sociais, possam usá-los? É desejável que as fotos e os vídeos estejam disponíveis, mesmo se o aplicativo for desinstalado?

O *framework* Android suporta a captura de imagens e vídeos pela `android.hardware.camera2`. Aqui estão as classes relevantes:

- `Android.hardware.camera2`. Este pacote é a principal API para controlar câmeras de dispositivos. Pode ser usado para tirar fotos ou gravar vídeos, quando estiver sendo criado um aplicativo de câmera;
- `SurfaceView`. Esta classe é usada para apresentar uma prévia da câmera ao vivo para o usuário;
- `MediaRecorder`. Esta classe é usada para gravar vídeos da câmera.

A Câmera API 2.0 é uma nova API que permite um controle muito melhor por aplicativos e aumenta o desempenho da câmera. A Câmera API 2.0 é rápida o suficiente para aproveitar a resolução total do sensor. Isso significa que uma fotografia computacional como essa deve ser muito mais barata em dispositivos Android. A nova API também permite um controle muito mais refinado do sensor, lente e *flash* por quadro individual.

4.3.2 Acesso a sensores disponíveis

A maioria dos dispositivos com Android possui sensores integrados que medem movimento, orientação e várias condições ambientais. Esses sensores são capazes de fornecer dados brutos com alta precisão e são úteis, se desejamos monitorar o movimento ou o posicionamento tridimensional do dispositivo, ou desejamos monitorar alterações no ambiente próximo a um dispositivo. Tem-se como exemplo um aplicativo que é capaz de rastrear o sensor de gravidade de um dispositivo móvel para que possa ler e inferir gestos e movimentos do usuário, como a inclinação, oscilação ou rotação do aparelho. De forma semelhante, um determinado aplicativo pode ter a capacidade de utilizar os sensores de temperatura e umidade para informar para o usuário, como também o dispositivo tem a capacidade de usar sensores de campo geomagnético para expor uma bússola.

A plataforma Android suporta três grandes categorias de sensores (DEITEL; DEITEL; WALD, 2016, p. 200):

- sensores de movimento: medem forças de aceleração e forças de rotação ao longo de três eixos. Esta categoria inclui acelerômetros, sensores de gravidade, giroscópios e sensores vetoriais rotacionais;

- sensores ambientais: medem vários parâmetros ambientais, como temperatura e pressão do ar ambiente, iluminação e umidade. Esta categoria inclui barômetros, fotômetros e termômetros;
- sensores de posição: medem a posição física de um dispositivo. Esta categoria inclui sensores de orientação e magnetômetros.

Pode-se acessar sensores disponíveis no dispositivo e adquirir dados brutos do sensor usando a estrutura do sensor Android. A estrutura do sensor fornece várias classes e interfaces que ajudam a executar uma ampla variedade de tarefas relacionadas ao sensor. Por exemplo, pode-se usar a estrutura do sensor para fazer o seguinte:

- determinar quais sensores estão disponíveis em um dispositivo;
- determinar os recursos de um sensor individual, como o alcance máximo, o fabricante, os requisitos de energia e a resolução;
- adquirir dados brutos do sensor e definir a taxa mínima na qual você adquire os dados do sensor;
- registrar e cancelar o registro dos eventos do sensor, que monitoram as alterações do sensor.

A estrutura do sensor Android permite acessar vários tipos de sensores. Alguns desses sensores são baseados em *hardware* e alguns são baseados em *software*. Os sensores baseados em *hardware* são componentes físicos incorporados em um aparelho ou *tablet*. Eles derivam seus dados medindo diretamente propriedades ambientais específicas, como aceleração, força do campo geomagnético ou alteração angular. Os sensores baseados em *software* não são dispositivos físicos, embora imitem sensores baseados em *hardware*. Os sensores baseados em *software* obtêm seus dados de um ou mais dos sensores baseados em *hardware* e, às vezes, são chamados de sensores virtuais ou sensores sintéticos. O sensor de aceleração linear e o sensor de gravidade são exemplos de sensores baseados em *software*.

Alguns dispositivos com Android possuem todos os tipos de sensores. Por exemplo, a maioria dos aparelhos e *tablets* possui um acelerômetro e um magnetômetro, mas poucos aparelhos possuem barômetros ou termômetros.

Além disso, um dispositivo pode ter mais de um sensor de um determinado tipo. Por exemplo, um dispositivo pode ter dois sensores de gravidade, cada um com um intervalo diferente.

4.3.3 *Single e multitoque*

Seja uma interface de máquina humana (HMI) compatível com *smartphone*, *tablet* ou *touch*, alguns dispositivos com tela sensível ao toque são compatíveis, enquanto outros suportam multitoque. O princípio básico por trás dessas duas tecnologias é o mesmo: apoiar a entrada do usuário por meio do toque. No entanto, existem algumas diferenças fundamentais entre o toque único e o multitoque que discutiremos hoje.

Durante os primeiros anos da tecnologia *touchscreen*, os dispositivos eram limitados estritamente a funções de toque único. O toque único faz jus ao seu nome, suportando a entrada a partir de um único toque. Por exemplo, um toque no ícone com o dedo indicador para abrir o aplicativo em um *smartphone*, que é uma função de toque único, pois usa um único toque.

VOCÊ QUER VER?

Na palestra *Produtividade Ninja no Android Studio*, o desenvolvedor Ramon Rabello (2017) (Android *Software Engineer Senior* na Colab.re), dá dicas para usar o *Android Studio* de forma mais eficaz e ampliar sua produtividade. Para assistir acesse <<https://www.youtube.com/watch?v=3V8pkvNCAKM> (https://www.youtube.com/watch?v=3V8pkvNCAKM)>.

Isso não é necessariamente ruim, mas não oferece muito, em termos de versatilidade e expansão, em mais entradas de comando. Concedido, os usuários poderiam executar rápidos toques duplos para acionar um comando diferente. Ao contrário do toque único, o multitoque suporta o uso de dois ou mais comandos simultâneos baseados em toque. Um exemplo de função multitoque é o recurso *pinch-to-zoom*, que é encontrado em muitos *smartphones* e *tablets*. Se quiser aumentar o *zoom*, basta apertar a tela para dentro usando dois dedos. Se quiser diminuir o *zoom*, basta executar o movimento oposto, movendo para fora. Recursos como esse não são encontrados em dispositivos de toque único,

simplesmente porque não podem registrar dois pontos de contato simultâneos.

As funções de toque único e multitoque podem ser encontradas em dispositivos resistivos e capacitivos. Os dispositivos *touchscreen* resistivos identificam o ponto de contato do operador apertando as camadas superior e inferior juntas. Eles são menos sensíveis, mas podem ser usados com ou sem uma caneta. Dispositivos *touchscreen* capacitivos, por outro lado, identificam o ponto de toque com base na carga elétrica criada pelo operador. O corpo humano produz uma quantidade pequena, mas perceptível de eletricidade. E quando pressiona o dedo contra a superfície de um dispositivo capacitivo, ele capta essa carga para determinar exatamente onde tocou.

4.3.4 Reconhecimento de gestos

O reconhecimento de gestos é a interpretação matemática de um movimento humano por um dispositivo de computação.

O reconhecimento de gestos, acompanhado de reconhecimento facial, de voz, rastreamento ocular e reconhecimento de movimentos labiais são componentes do que os desenvolvedores chamam de interface de usuário perceptual (PUI). O objetivo do PUI é aumentar a eficiência e a facilidade de uso do design lógico subjacente de um programa armazenado, uma disciplina de *design* conhecida como usabilidade.

Na computação pessoal, os gestos são mais usados para comandos de entrada. Reconhecer os gestos como entrada, permite que os computadores sejam mais acessíveis para os deficientes físicos e torne a interação mais natural em um ambiente de jogos ou realidade virtual em 3D.

Gestos de mão e corpo podem ser amplificados por um controlador que contém acelerômetros e giroscópios para detectar inclinação, rotação e aceleração de movimento – ou o dispositivo de computação pode ser equipado com uma câmera para que o *software* no dispositivo possa reconhecer e interpretar gestos específicos. Uma onda feita com a mão, por exemplo, pode terminar o programa.

Além dos desafios técnicos da implementação do reconhecimento de gestos, também existem desafios sociais. Os gestos devem ser simples, intuitivos e universalmente aceitáveis. O estudo de gestos e outros tipos não verbais de comunicação é conhecido como cinesiologia.

4.4 Projeto de aplicativo

Agora, vamos entender como se desenvolve um projeto em Android com geolocalização. O objetivo do projeto é mostrar como recuperar a localização atual de um usuário e exibi-lo com um marcador em um mapa no dispositivo móvel. Segundo Lee, Schneider e Schell (2005, p. 100), ao se desenvolver aplicativos para dispositivos móveis, “cada objeto representa uma entidade em um cenário do dia a dia de aplicação”. Neste projeto, é importante identificar quais serão os objetos.

O Android tem duas maneiras básicas para determinar a localização de um usuário. A primeira é usar as APIs de localização incorporadas, que estão disponíveis desde que o Android foi introduzido pela primeira vez. Eles ainda funcionam, mas não tão bem quanto as APIs de localização mais recentes agrupadas conhecidas como Google *Play Services*. A API de serviços de localização do Google, parte do Google *Play Services*, fornece uma estrutura de alto nível mais poderosa que automatiza tarefas como a escolha do provedor de localização e o gerenciamento de energia. Os Serviços de Localização também fornecem novos recursos, como detecção de atividades, que não estão disponíveis na API do *framework*. Os desenvolvedores que usam a API de estrutura, bem como os desenvolvedores que agora adicionam reconhecimento de local aos aplicativos, devem considerar seriamente o uso da API de serviços de localização.

CASO

O desenvolvimento de aplicativos para dispositivos móveis, tem atingido as mais diversas áreas e atividades profissionais. Na área da saúde, o prontuário médico é um documento elaborado pelo profissional, e é uma ferramenta fundamental para seu trabalho. No prontuário são apresentados, de forma organizada e concisa, todos os dados relativos ao paciente, como seu histórico familiar, anamnese, descrição e evolução de sintomas e exames, além das indicações de tratamentos e prescrições. Neste sentido, foi proposto por um grupo de pesquisadores (SANTOS *et al.*, 2017), um modelo de Prontuário Eletrônico Único do Paciente. Esta é uma proposta para manipular a dispersão de dados dos prontuários sem ter que mobilizar grandes volumes de dados em atendimentos. Assim, esse modelo fornece níveis de informação, nos quais se destaca a síntese que concentra as informações relevantes para os atendimentos. Este modelo é disponível em dispositivos móveis e um *middleware* para negociar as transações entre eles.

A Google disponibiliza seus APIs por meio do Google *Play Services*, que é um pacote que oferece diversos recursos para serem implantados nos aplicativos desenvolvidos no sistema operacional Android. Eles são instalados na maioria dos dispositivos e executados no Android 2.3 e superiores.

A primeira tarefa a fazer é executar o *download* do SDK do Google *Play Services* usando o SDK *Manager*.

A seguir, deve ser criado um modelo de projeto disponível no Android *Studio*. A seguir, vamos acompanhar a demonstração de Deitel (2016) de como instalar o Android *Studio* de forma fácil. O modelo "Atividade do Google *Maps*" gera um código para localização que facilitará o desenvolvimento.

- Criar um novo projeto ("EuEstouAqui").
- Selecionar "Telefone e *tablet*" como forma.
- Selecionar a "Atividade do Google *Maps*" como o modelo da atividade e concluir a criação do projeto.

Na figura a seguir vemos a tela de criação inicial do projeto no Android *Studio*.

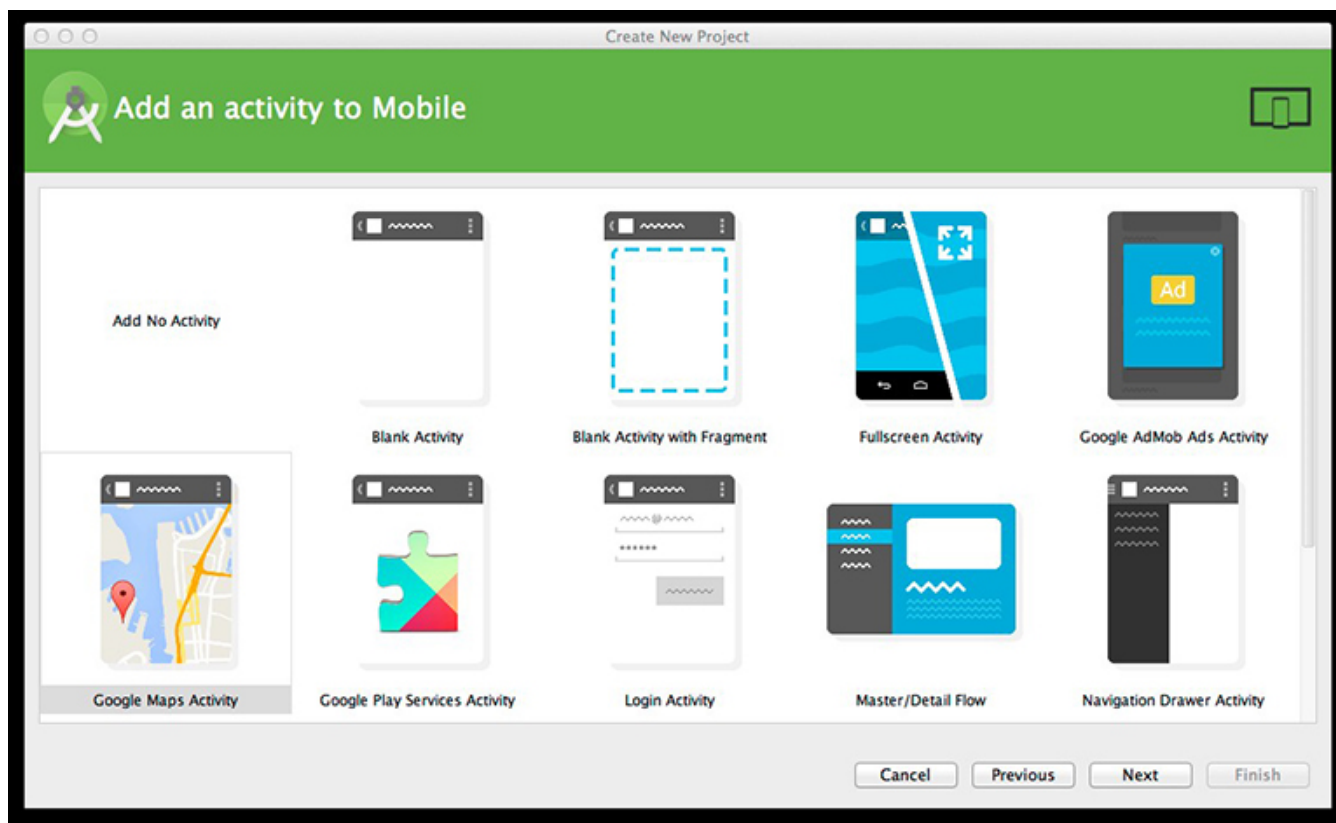


Figura 1 - O modelo de atividade do Google Maps da ferramenta Android Studio para criação de aplicativos. Fonte: Elaborada pelo autor, 2018, na ferramenta Android Studio.

#PraCegoVer: Print da tela do Android Studio contém no topo uma faixa verde escrito: Add na activity to Mobile, abaixo da faixa verde do lado esquerdo tem um pedaço em branco escrito Add No Activity; do lado à direita a tela de um celular em branco e no topo dessa tela representa o browser e embaixo dele está escrito Blank Activity; do lado desta tela à direita tem outra tela de celular e dentro dela tem no topo uma faixa representando o browser e embaixo o desenho de uma linha pontilhada azul claro que faz o contorno da borda da tela em branco e embaixo desta tela está escrito Blank Activity with Fragment. Do lado desta tela à direita tem outra tela de celular com faixas onduladas intercalando azul escuro com azul claro (cor de fundo da tela do celular) e no topo desta tela tem uma faixa representando o browser, do lado direito, tem um símbolo branco quadrado, onde cada extremidade desse quadrado é uma seta apontando de dentro para fora do quadrado, uma linha branca no meio da tela inclinada para a esquerda cruza a tela na diagonal e abaixo desta tela está escrito Fullscreen Activity; do lado desta tela à direita tem outra tela de celular preta e dentro dela tem uma quadrado azul com um quadrado amarelo no meio da tela escrito Ad, embaixo desta tela de celular está escrito Google AdMob Ad s Activity. Abaixo dessas telas tem-se mais 5 telas de

celular, à primeira tela à direita tem no topo uma linha representando o browser e embaixo o desenho de uma mapa e de um localizador em vermelho e abaixo dessa tela está escrito Google Maps Activity; do lado desta tela à direita tem outra tela de celular onde dentro dela no topo tem uma linha representando um browser e abaixo o símbolo do google play que é uma peça de quebra cabeça com o encaixe do lado esquerdo da peça e uma parte a ser encaixada no topo dela, está peça é colorida, com uma parte verde claro, outra azul claro, outra vermelho e a outra laranja, abaixo desta tela de celular está escrito Google Play Service Activity; do lado desta tela à direita tem outra tela de celular e dentro dela tem uma tela onde no topo tem uma barra representando o browser e embaixo tem dois campos um para digitar email e embaixo um para digitar a senha e embaixo um botão, embaixo da tela está escrito Login Activity; do lado desta tela à direita tem outra tela de celular mas virada na horizontal e dentro dela tem uma faixa cinza à esquerda com algumas linhas brancas onduladas (representando escritas) e do lado tem um quadro branco e embaixo dele algumas linhas brancas onduladas (representando escritas), embaixo desta tela está escrito Master / Detail Flow; do lado desta tela à direita tem outra tela de celular e dentro dela tem no topo uma barra representando um browser e embaixo do lado esquerdo tem uma faixa cinza algumas linhas brancas onduladas (representando escritas) e do lado direito tela em branco.

Em seguida, é necessário criar uma chave API do Google *Maps* para configurar o Google *Play Services*. Esta chave é obtida clicando-se no endereço:

```
<https://console.developers.google.com/flows  
/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&  
r=17:8A:66:F5:C7:FF:50:E3:77:FB:79:8A:79:3F:30:43:FC:D6:CE:61%3  
(https://console.developers.google.com/flows  
/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&  
r=17:8A:66:F5:C7:FF:50:E3:77:FB:79:8A:79:3F:30:43:FC:D6:CE:61%3)>.
```

É necessário acessar este *link* e se inscrever em uma chave da API do Google *Maps*. Não há custo para se inscrever e, quando receber uma chave desse *site*, você deve voltar a esse arquivo e substituir YOUR_KEY_HERE pela nova chave.

Para demonstrar o processo, fizemos capturas de tela, que podem ser usadas como referência.

1. Selecione 'Criar um novo projeto' e clique em 'Continuar', conforme figura a seguir:

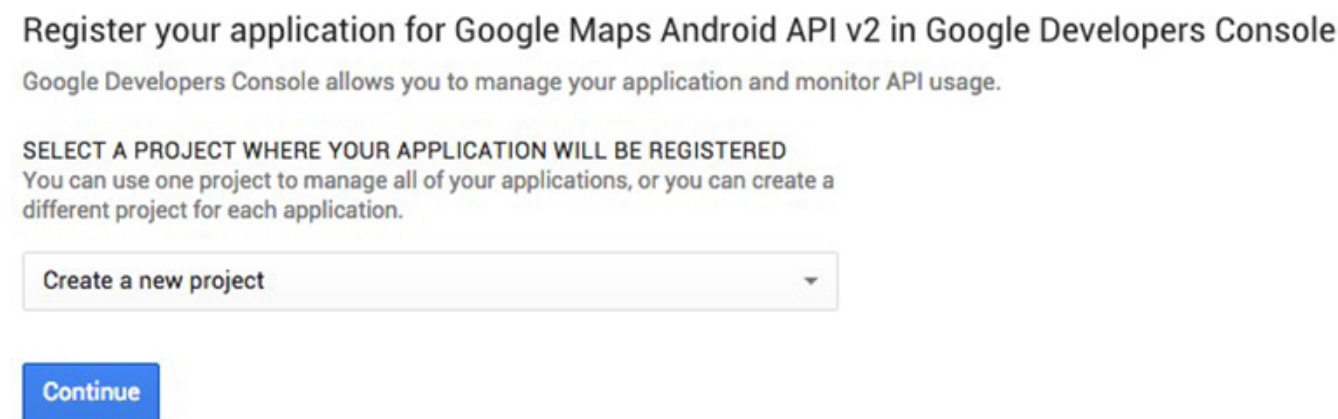


Figura 2 - Criação de novo projeto do Google Maps console developers Google do Google API – etapa 1.

Fonte: Elaborada pelo autor, 2018 na ferramenta Google API.

#PraCegoVer: Print da tela de criação de um novo projeto escrito:

Register your application for Google Maps Android API v2 in Google Developers Console

Google Developers Console allows you to manage your application and monitor API usage

SELECT A PROJETO WHERE YOUR APPLICATION WILL BE REGISTERED

You can use one project to manage all of your applications, or you can create a different project for each application

Opção selecionada: Create a new project e abaixo um botão azul escrito Continue.

2. A criação do novo projeto leva alguns segundos e ficará na tela que vemos na figura a seguir:





Activities (1) ↻ —	
Create Project: My Project	
Create Project: My Project	
Create Project: My Project	
Create Project: My Project	
See all activity	

Figura 3 - Criação de novo projeto do Google Maps console developers Google do Google API – etapa 2.

Fonte: Elaborada pelo autor, 2018 na ferramenta Google API.

#PraCegoVer: Print da tela chamada Activities (1), abaixo escrito Create Project: My Project e na frente o símbolo de carregando, abaixo escrito Create Project: My Project e na frente o símbolo de tick verde, abaixo escrito Create Project: My Project e na frente o símbolo de tick verde e abaixo escrito Create Project: My Project e na frente o símbolo de tick verde.

3. Clique em ‘Criar’, para, finalmente, criar a chave de API, conforme aparece na figura a seguir.

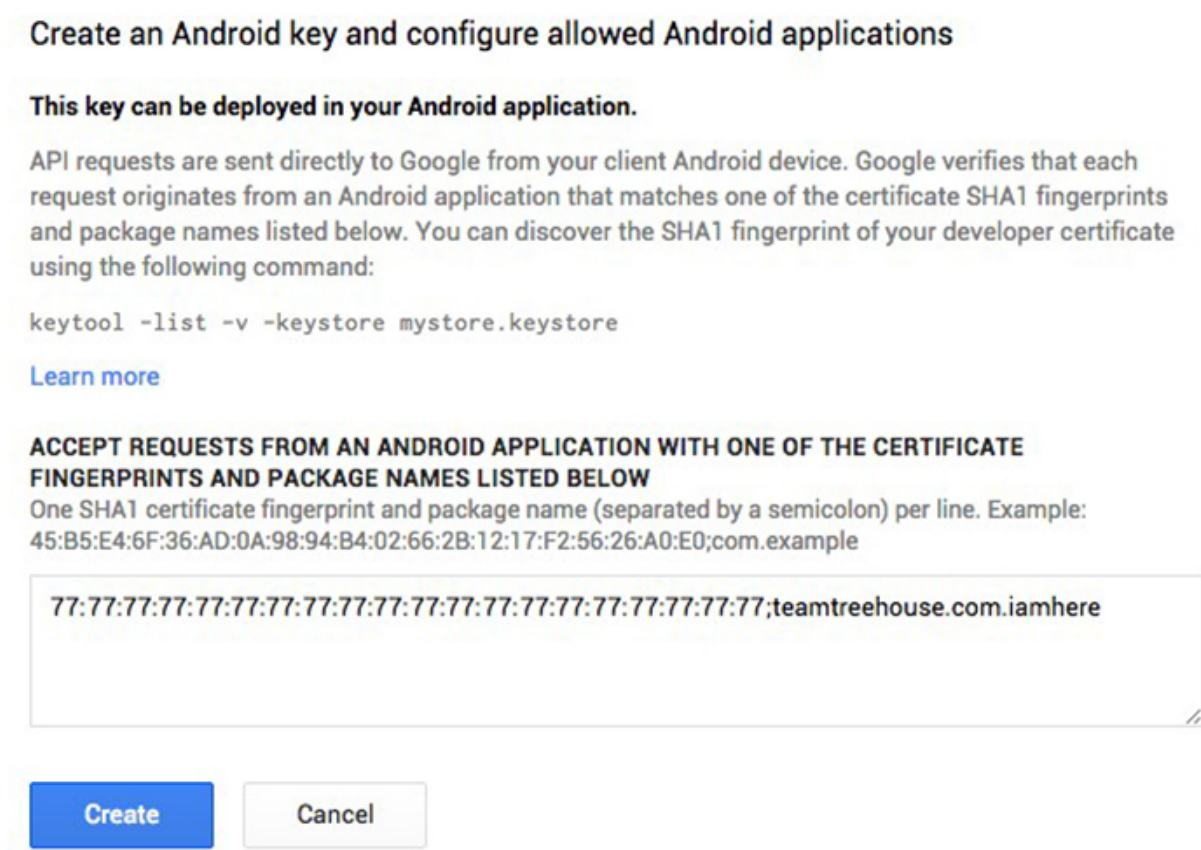


Figura 4 - Criação de novo projeto do Google Maps console developers Google do Google API – etapa 3.

Fonte: Elaborada pelo autor, 2018 na ferramenta Google API.

#PraCegoVer: Print da tela para criar, está escrito:

Create an Android key and configure allowed Android applications.

This key can be deployed in your Android Application.

API requests are sent directly from an Android application that matches one of the certificate SHA1 fingerprints and package names listed below. You can discover the SHA1 fingerprint of your developer certificate using the following command:

```
Keytool -list -v -keystore mystore.keystore
```

[Learn more](#) (escrito em azul)

ACCEPT REQUESTS FROM AN ANDROID APPLICATION WITH ONE OF THE CERTIFICATE FINGERPRINTS AND PACKAGE NAMES LISTED BELOW

One SHA1 certificate fingerprint and package name (separated by a semicolon) per line. Example:

```
45:B5:E4:6F:AD:0A:98:B4:02:66:2B:12:17:F2:56:26:E0;com.example
```

Desenho de um quadro branco e dentro dele escrito:

77:77: 77:77: 77:77: 77:77: 77:77: 77:77: 77:77: 77:77: 77:77:
77:77;teamtreehouse.com.iamhere

Abaixo dois botões um azul escrito Create e do lado um branco escrito Cancel.

4. Por fim, copiar e colar sua API KEY no aplicativo.

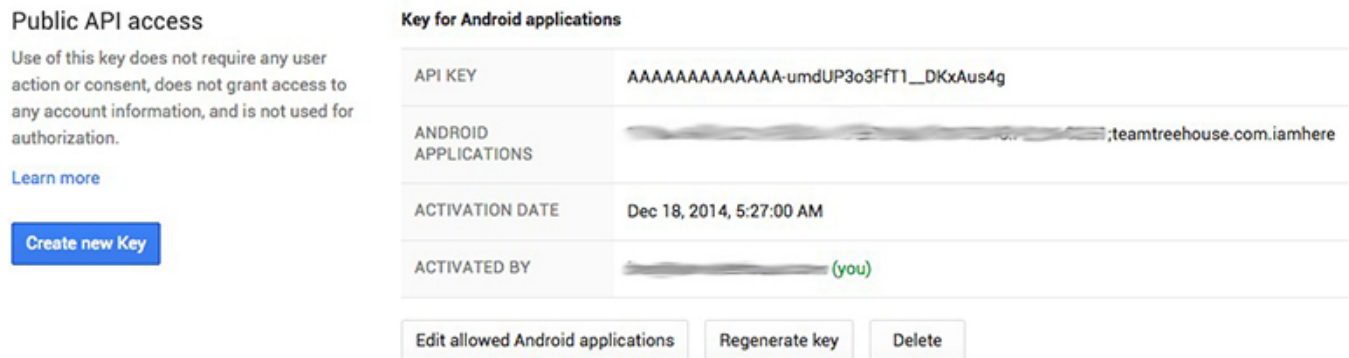


Figura 5 - Criação de novo projeto do Google Maps console developers Google do Google API – etapa 4.

Fonte: Elaborada pelo autor, 2018 na ferramenta Google API.

#PraCegoVer: Print da tela para criar, do lado esquerdo está escrito:

Public API access

Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.

Learn more(escrito em azul)

Botão azul escrito: Create new key

Do lado direito os campos, começando com a escrita: Key for Android applications, embaixo tem os campos:

API KEY preenchido com AAAAAAAAAAAAAA.umdUP3o3FFt1__DKxAus4g

ANDROID APPLICATIONS teamtreehouse.com.iamhere

ACTIVATION DATE Dec 18, 2014, 5:27:00 AM

ACTIVATED BY (you)

É necessário criar um arquivo “xml” denominado google_maps_api.xml. O arquivo (API) tem como função oferecer acesso seriado ao conteúdo de um documento XML por meio de orientação a evento. É necessário criar este arquivo e substituir YOUR_KEY_HERE pela nova chave.


```

<resources>
<string name="google_maps_key_instructions" templateMergeStrategy="replace"><!--

https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=17:8A:66:F5:C7:FF:50:E3:77:FB:79:8A:79:3F:30:43:FC:D6:CE:61%3B

string in this file.
--></string>

<string name="google_maps_key" templateMergeStrategy="preserve"> YOUR_KEY_HERE </string>
</resources>

```

Figura 6 - Criação do arquivo google_maps_api.xml para o desenvolvimento do aplicativo de geolocalização. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela, com o código xml:

```

<resources>
<string
                                name="google_maps_key_instructions"
templateMergeStrategy="replace"> https://console.developers.google.com/flows
/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=17:8A:66:F5:C7:FF:50:E3:77:FB:79:8A:79:3F:30:43:FC:D6:CE:61%3B
string in file:
--></string>
<string    name="google_maps_key"        templateMergeStrategy="preserve">
YOUR_KEY_HERE </string>
</resources>

```

Neste momento é necessário criar o arquivo build.gradle, que define o nome do programa. Quando utilizamos o Android *Studio*, o processo de *build* completo é feito a cada vez que você executa o *Gradle* para construir seus projetos ou módulos. O build.gradle é definido como na figura a seguir.


```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 21
    buildToolsVersion "21.0.2"

    defaultConfig {
        applicationId "EuEstouAqui"
        minSdkVersion 14
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.2'
    compile 'com.google.android.gms:play-services:6.1.71'
}
```

Figura 7 - Criação do arquivo build.gradle para desenvolvimento do aplicativo de geolocalização. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela, com o código:

```
apply plugin: 'com.android.application'
```

```
android{
    compileSdkVersion 21
    buildToolsVersion "21.0.2"
```

```
defaultConfig {
    applicationId "EuEstouAqui"
```

```
minSdk Version 14
targetSdk Version 21
versionCode 1
versionName "1.0"
}
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'),'proguard-rules.pro'
    }
}

dependencies {
    compile fileTree(dir: 'libs',include: ['*.jar'])
    compile 'com.android.support:appcompat-v7:21.0.2'
    compile 'com.google.android.gms:play-services:6.1.71'
}
```

Neste momento, é necessário iniciar o Gerenciador do SDK novamente e selecionar uma das imagens do emulador (AVD), que usa as APIs da Google. Depois de ter um dispositivo ou emulador apropriado para testar, executar o aplicativo e verificar se o mapa é carregado. Em caso afirmativo, a nova chave de API é válida e o projeto está pronto para ser implantado.

A seguir, para se obter o local atualizado, é necessário criar o arquivo `AndroidManifest.xml`, como vemos na figura:

```

<?xml version="1.0" encoding="utf-8"?>
  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="teamtreehouse.com.iamhere" >

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
      <meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="@string/google_maps_key" />

      <activity
        android:name=".MapsActivity"
        android:label="@string/title_activity_maps" >
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
    </application>
  </manifest>

```

Figura 8 - Criação do arquivo AndroidManifest.xml para desenvolvimento do aplicativo de geolocalização.

Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela, com o código:

```

<manifest xmlns:android=http://scheas.android.com/apk/res/android
package="teamtreehouse.com.iamhere">

```

```

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission

```

```
android:name="com.google.android.providers.gsf.permission.READ_GSERVICE" />
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

<application
android:allowBackup="true"
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@style/AppTheme"
<meta-data
android:name="com.google.android.gms.version"
android:value="@integer/google_play_services_version" />
<meta-data
android:name="com.google.android.maps.v2.API_KEY"
android:value="@string/google_maps_key" />

<activity
android:name=".MapsActivity"
android:value="@string/google_maps_key" />
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
<intent-filter>
</activity>
</application>

</manifest>
```

Na sequência, é necessário verificar se o Google *Play Services* está disponível. Está disponível, por padrão, na maioria dos dispositivos, mas se não for, deve-se solicitar ao usuário que o instale, antes de usar o aplicativo. No código Java, isso é realizado automaticamente. Podemos ver o arquivo `MapsActivity.java` na figura abaixo.

Package

EuEstouAqui;

```

import android.content.IntentSender;
import android.location.Location;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.util.Log;

import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.api.GoogleApiClient;
import com.google.android.gms.location.LocationListener;
import com.google.android.gms.location.LocationRequest;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements
    GoogleApiClient.ConnectionCallbacks,
    GoogleApiClient.OnConnectionFailedListener,
    LocationListener {

    public static final String TAG = MapsActivity.class.getSimpleName();

    private final static int CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;

    private GoogleMap mMap;

    private GoogleApiClient mGoogleApiClient;
    private LocationRequest mLocationRequest;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        setUpMapIfNeeded();

        mGoogleApiClient = new GoogleApiClient.Builder(this)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .addApi(LocationServices.API)
            .build();

        // Create the LocationRequest object
        mLocationRequest = LocationRequest.create()
            .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
            .setInterval(10 * 1000)
            .setFastestInterval(1 * 1000);
    }

    @Override
    protected void onResume() {
        super.onResume();
        setUpMapIfNeeded();
        mGoogleApiClient.connect();
    }

    @Override
    protected void onPause() {
        super.onPause();

        if (mGoogleApiClient.isConnected()) {
            LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient, this);
            mGoogleApiClient.disconnect();
        }
    }

    private void setUpMapIfNeeded() {
        if (mMap == null) {
            mMap = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map))
                .getMap();
        }
    }

```

```
do arquivo Maps
  if (mMap != null)
    setUpMap()
  }
}
```

```

        if (mMap != null) {
            setUpMap(Fon
        }
    }
}

print da tela, com
mMap.addMarker(new
}

```

```

mMap.addMarker(new MarkerOptions().position(new LatLng(0, 0)).title("Marker"));
}

```

```
private void handleNewLocation(Location location) {
    Log.d(TAG, location.toString());

    double currentLatitude = location.getLatitude();
    double currentLongitude = location.getLongitude();
}
```

```
double currentLatitude = location.getLatitude();
double currentLongitude = location.getLongitude();
```

```
content.IntentSender;
        LatLng newLatLng = new LatLng(currentLatitude, currentLongitude);
```

```
location: Location; new MarkerOptions()
    .position(latLng)
```

```
os.Bundle("I am here!");
```

```
support.v4.app.FragmentActivity;
```

```
util.Log;
```

```
@Override
public void onConnected(Bundle bundle) {
    Location location = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
```

```

    @Override
    public void onConnectionSucceeded() {
        Log.d(TAG, "onConnectionSucceeded");
        // ...
    }
}

```

```
gle.android.gms.common.api.GoogleApiClient;
```

```

    private static final LocationListener listener = new LocationListener() {
        public void onLocationChanged(Location location) {
            // Handle new location
        }
    };

```

```
gle.android.gms.location.LocationRequest;
```

```
import android.gms.location.LocationServices;
```

```
    private static final CameraUpdateFactory
```

```
gle.android.gms.maps.GoogleMap;
```

```
public class SupportMapFragment {
```

```
gle.android.gms.maps.model.LatLng;
```

```
gle.android.gms.maps.model.MarkerOptions;
```

$$\left. \begin{array}{l} \\ \end{array} \right\}$$

```
String TAG = MainActivity.class.getSimpleName();
```

3

```
int CONNECTION_FAILURE_RESOLUTION_REQUEST = 9000;
```

```
handleNewLocation(location);
```

```
ap mMap;
```

ApiClient mGoogleApiClient:

Request mLocationRequest:

```
onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);  
setContentView(R.layout.activity_maps);  
setUpMapIfNeeded();
```

```
mGoogleApiClient = new GoogleApiClientBuilder(this)  
.addConnectionCallbacks(this)  
.addOnConnectionFailedListener(this)  
.addApi(LocationServices.API)  
.build();
```

```
// Create the LocationRequest object  
mLocationRequest = LocationRequest.create()  
.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)  
.setInterval(10 * 1000)  
.setFasterInterval(1 * 1000);  
}
```

```
@Override  
protected void onResume() {  
super.onResume();  
setUpMapIfNeeded();  
mGoogleApiClient.connect();  
}
```

```
@Override  
protected void onPause() {  
super.onPause();  
}
```

```
if (mGoogleApiClient.isConnected()) {  
LocationServices.FusedLocationApi.removeLocationUpdates(mGoogleApiClient.  
This);  
mGoogleApiClient.disconnect();  
}  
}
```

```
Private void setUpMapIfNeeded() {
```

```
if (mMap == null) {  
    mMap = ((SupportMapFragment)  
getSupportFragmentManager().findFragmentById(R.id.map))  
.getMap();  
}
```

```
if (mMap != null) {  
    setUpMap();  
}  
}
```

```
private void setUpMap() {  
    mMap.addMarker(new MarkerOptions().position(new LatLng(0,  
0)).title("Marker"));  
}
```

```
Private void handleNewLocation(Location location) {  
    Log.d(TAG.location.toString());  
}
```

```
double currentLatitude = location.getLatitude();  
double currentLongitude = location.getLongitude();
```

```
LatLng latLng = new LatLng(currentLatitude, currentLongitude);
```

```
MarkerOptions options = new MarkerOptions()  
.positio(latLng)  
.title("I am here!");  
mMap.addMarker(options);  
mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));  
}
```

```
@Override  
Public void onConnected(Bundle bundle) {  
    Location location =  
    Location.Services.FusedLocation.Api.getLastLocation(mGoogleApiClient);  
    if (location == null) {
```



```
LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient,
mLocationRequest, this);

}
else {
handleNewLocation(location);
}
}

@Override
Public void onConnectionSuspended(int i) {

}

@Override
public void onConnectionFailed(Connection Result connectionResult) {
if (connectionResult.hasResolution()) {
try {
connectionResult.startResolutionForResult(this,
CONNECTION_FAILURE_RESOLUTION_REQUEST);
} catch (IntentSender.SendIntentException e) {
e.printStackTrace();
}
} else{
Log.i(TAG, "Location services connection failed with code " +
connectionResult.getErrorCode());
}
}

@Override
public void onLocationChanged(Location location) {
handleNewLocation(location);
}
}

mGoogleApiClient = new GoogleApiClient.Builder(this)
```

```
.addConnectionCallback(this)
.addOnConnectionFailedListener(this)
.addApi(LocationServices.API)
.build();
```

Para compreender melhor o código é necessário fazer algumas observações. É necessário usar o `GoogleApiClient`, que é o cliente mais recente do Google *Play Services*, projetado para facilitar a configuração e o uso desta e de outras funções do Google *Play Services*. “privado `GoogleApiClient mGoogleApiClient;`”.

É preciso de uma variável TAG, conforme a declaração: “public static final String TAG = `MapsActivity.class.getSimpleName ();`”.

Agora é necessário inicializar o cliente na parte inferior do `onCreate()` método `MapActivity`.

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(LocationServices.API)
    .build();
```

Agora, é necessário implementar os métodos no Android *Studio*, como demonstra a figura abaixo.

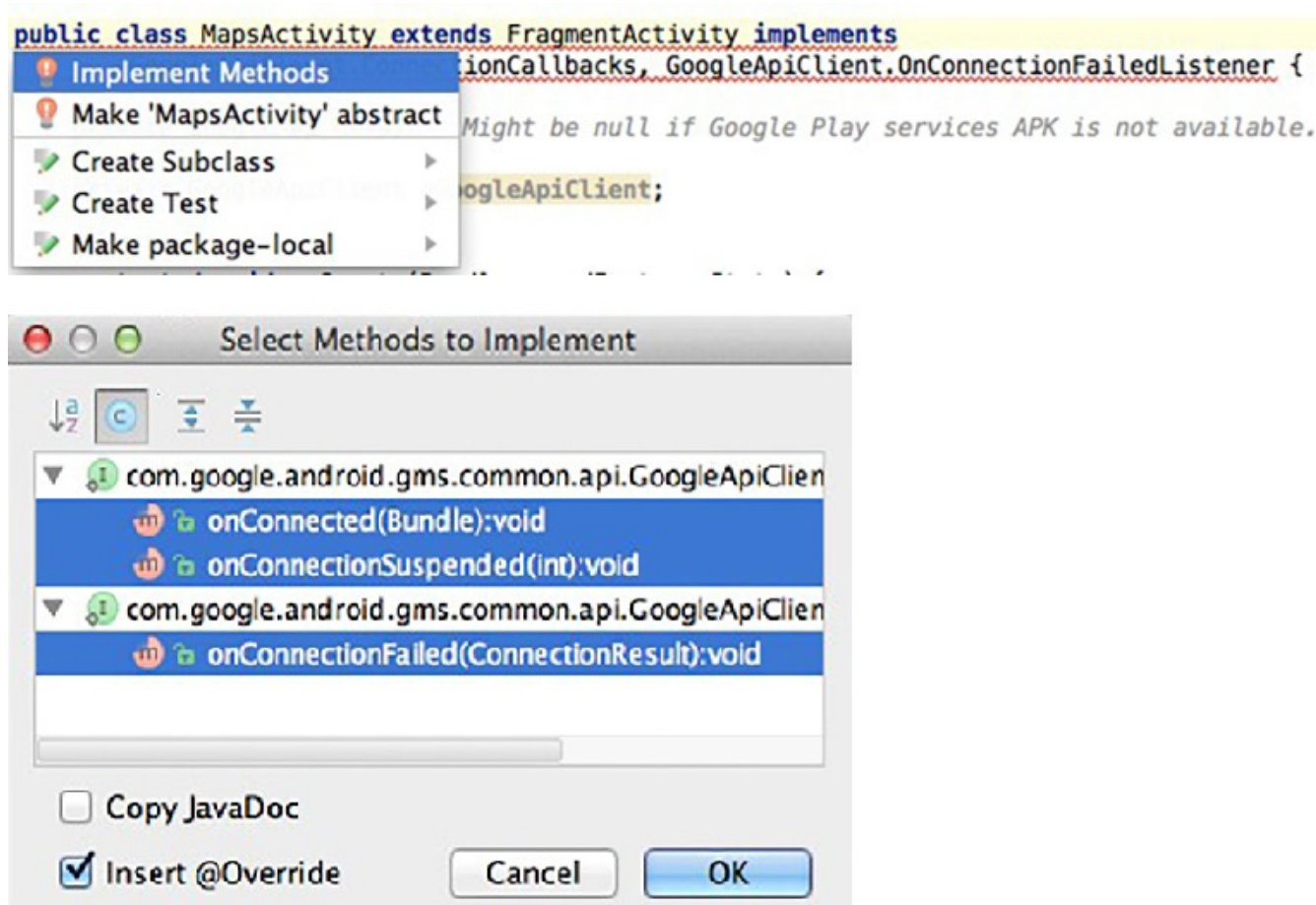


Figura 10 - Implementação dos métodos para desenvolvimento do aplicativo de geolocalização. Utilização da ferramenta Android Studio. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: print da tela do Android Studio na primeira linha está escrito `public class MapsActivity extends FragmentActivity implements`. Em baixo da palavra `public class` tem uma caixa com 5 opções: a primeira `Implement Methods` selecionada em azul, segunda `Make 'MapsActivity' abstract`, terceira `Create Subclass`, quarta `Create Test` e quinta `Make package-local`. Em baixo dessa caixa tem outra caixa maior com o título `Select Methods to Implement`, em baixo tem a barra com 4 botões, o segundo está selecionado, tem um bolinha azul clara e dentro dela a letra `c`; em baixo tem uma barra escrito `com.google.android.gms.api.GoogleApiClient` com os itens `onConnected(Bundle):void` e `onConnectionSuspended(int): void`, ambos selecionados na cor azul; em baixo tem uma barra escrito `com.google.android.gms.common.api.GoogleApiClient` com o item `onConnectionFailed(ConnectionResult):void` selecionado na cor azul; no final da caixa tem duas opções `CopyJavaDoc` (não selecionado) e `Insert @Override` (selecionado) e na frente dois botões: `Cancel` na cor branco acinzentado e `OK` na

cor azul para seleção.

Finalmente, se executar o aplicativo para verificar, veremos um novo marcador, que nos permite saber exatamente onde estamos, como mostra a figura a seguir.



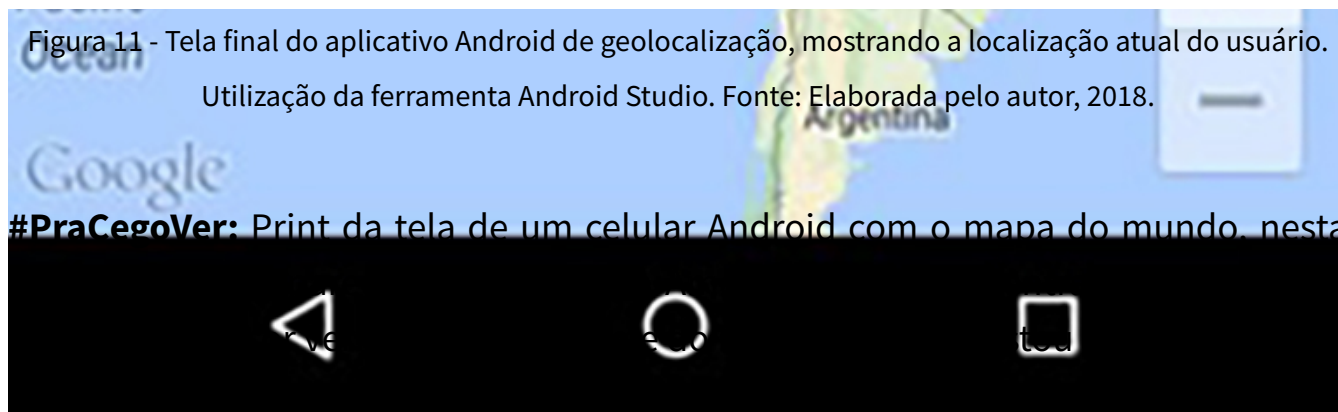


Figura 11 - Tela final do aplicativo Android de geolocalização, mostrando a localização atual do usuário.

Utilização da ferramenta Android Studio. Fonte: Elaborada pelo autor, 2018.

#PraCegoVer: Print da tela de um celular Android com o mapa do mundo, nesta

O desenvolvimento de aplicativos utilizando o *Android Studio* permite criar sistemas de forma bastante intuitiva e com alto grau de qualidade. As diversas funcionalidades que as APIs Google oferecem, interagindo com os recursos nativos do sistema operacional Android, tornam o desenvolvimento e implantação de aplicativos bastante eficiente. Além dos métodos e operações das APIs da Google, os serviços em nuvem podem ser integrados com os aplicativos, para serem desenvolvidas novas operações, como o armazenamento em nuvem, a comunicação por meio de mensagens, e a utilização de recursos avançados dos serviços Google e do sistema Android.

Síntese

Compreendemos neste capítulo como é possível aprimorar os aplicativos desenvolvidos para dispositivos móveis utilizando as APIs da Google. O Google oferece uma série de funcionalidades que incrementam e elevam a qualidade dos aplicativos, tornando-os mais eficientes e robustos, como geolocalização e serviços na nuvem, que estão cada vez mais presentes em diversos aplicativos. Outras funcionalidades oferecidas, com bastante requinte, são os sensores e toque, que permitem uma interação inteligente e intuitiva com a interface dos usuários de dispositivos móveis.

Neste capítulo, você teve a oportunidade de:

- utilizar os recursos de geolocalização e o Google *Maps*;
- conhecer os serviços em nuvens como *Android Data Backup*, *Firebase* e *Cloud Messaging*;
- programar os recursos de sensores, *touch* e câmera, bem como o reconhecimento de gestos;

- desenvolver um aplicativo de geolocalização utilizando o Android *Studio*.



◀ Clique para baixar o conteúdo deste tema.

Bibliografia

DEITEL, P. J.; DEITEL, H.; WALD, A. **Android 6 para Programadores**: uma abordagem baseada em aplicativos. 2. ed. Porto Alegre: Bookman, 2016. Disponível na Biblioteca Virtual Ânima: <https://Animabrasil.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_198689_1&content_id=_4122211_1&mode=reset (https://Animabrasil.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_198689_1&content_id=_4122211_1&mode=reset)>. Acesso em: 21/06/2018.

EGGEA, R, F. **Aplicação Android utilizando sistema de localização geográfica para determinação de pontos turísticos na cidade de Curitiba**. Monografia de especialização para o curso de Tecnologia Java e Desenvolvimento para Dispositivos Móveis. Universidade Tecnológica Federal do Paraná. 2013.

FELIPE, H. L.; DIAS, J. W. Aplicações baseadas em geolocalização. **XVI Semana de Informática de Paranavaí**. Universidade Paranaense (UNIPAR). Paranavaí-PR-Brasil, 2014. Disponível em: <<http://web.unipar.br/~seinpar/2014> (<http://web.unipar.br/~seinpar/2014>)>. Acesso em: 21/06/2018.

FERNANDES, F. G.; BARBOSA, J. L. M. Automação Residencial Aplicada para Pessoas com Limitação Motora. **Anais**. XIII Encontro Anual de Computação – EnAComp. 24 a 26 de maio de 2017 – Catalão (GO): UFG, 2017. Disponível em: <http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf (http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf)>. Acesso em: 21/06/2018.

MOLIANE, C. A.; ALMEIDA, D. E. C. **(ART) e Dalvik em dispositivos Android**. Iv Srst – Seminário de Redes e Sistemas de Telecomunicações. 2016. Disponível em: <<https://docplayer.com.br/54762841-Art-e-dalvik-em-dispositivos-android.html>

(<https://docplayer.com.br/54762841-Art-e-dalvik-em-dispositivos-android.html>)>.

Acesso em: 21/07/2018.

LAMAS, A. C.; SOUZA, R. I. Sistema de apoio à localização para deficientes visuais.

Extensio: Revista Eletrônica de Extensão, Florianópolis, v. 13, n. 24, p. 37-48, dez. 2016. ISSN 1807-0221. Disponível em: <<https://periodicos.ufsc.br/index.php/extensio/article/view/1807-0221.2016v13n24p37> (<https://periodicos.ufsc.br/index.php/extensio/article/view/1807-0221.2016v13n24p37>)>. Acesso em: 21/06/2018.

LEE, V.; SCHNEIDER, H.; SCHELL, R. **Aplicações Móveis**: arquitetura, projeto e desenvolvimento [recurso eletrônico, Biblioteca Virtual 3.0]. Pearson Education do Brasil: São Paulo, 2005.

MURUKANNAIAH, P.; SINGH, M. Platys: An Active Learning Framework for Place-Aware Application Development and Its Evaluation. **ACM Transactions on Software Engineering and Methodology** (TOSEM), Vol.24(3), pp.1-32, 13 May 2015. Disponível em: <<https://www.csc2.ncsu.edu/faculty/mpsingh/papers/mas/TOSEM-15-Platys.pdf> (<https://www.csc2.ncsu.edu/faculty/mpsingh/papers/mas/TOSEM-15-Platys.pdf>)>. Acesso em: 21/06/2018.

RABELLO, R. **Produtividade Ninja no Android Studio**. Canal iMasters, YouTube, publicado em 25 de out. de 2017. Disponível em: <<https://www.youtube.com/watch?v=3V8pkvNCAKM> (<https://www.youtube.com/watch?v=3V8pkvNCAKM>)>. Acesso em: 21/06/2018.

SANTOS, A. A.; *et al.* Modelo para Disponibilização do Prontuário Eletrônico Único do Paciente em Dispositivos Móveis. **Anais**. XIII Encontro Anual de Computação – EnAComp. 24 a 26 de maio de 2017 – Catalão (GO): UFG, 2017. Disponível em <http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf (http://www.enacomp.com.br/2017/docs/Anais_Enacomp2017.pdf)>. Acesso em: 21/06/2018.

SANTOS, E. S.; *et al.* Aplicativos móveis para pessoas com deficiência aplicando-se técnicas de ciência de contexto. **Revista Principia**, 01, Vol.1(27), pp.11-18. Dezembro 2015.

MACHADO, D. R.; MACHADO, R. P.; CONFORTO, D. Dispositivos móveis e usuários cegos: recomendações de acessibilidade em discussão. **Nuevas Ideas en Informática Educativa TISE**, 2014. Disponível em: <<http://www.tise.cl/volumen10>

/TISE2014/tise2014_submission_231.pdf (http://www.tise.cl/volumen10/TISE2014/tise2014_submission_231.pdf)>. Acesso em: 21/06/2018.

SOUZA, R. F. **Aplicativo Android para monitoramento remoto por câmeras IP com sensor de presença**. Trabalho de Conclusão de Curso apresentado ao curso Bacharelado em Ciência da Computação do Instituto Municipal de Ensino Superior de Assis – IMESA e à Fundação Educacional do Município de Assis, 2012. Disponível em: <<https://cepein.femanet.com.br/BDigital/arqTccs/0911270671.pdf>> (<https://cepein.femanet.com.br/BDigital/arqTccs/0911270671.pdf>)>. Acesso em: 21/06/2018.

TANENBAUM, A. S.; BOS, H. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Person Education do Brasil, 2016.

VANDRESEN, R. S.; MAGALHÃES, W. B. Conceitos e Aplicações da Computação em Nuvem. **XV Semana de Informática de Paranavaí**. Universidade Paranaense (UNIPAR) Paranavaí-PR-Brasil, 2013. Disponível em: <https://www.academia.edu/23654744/CONCEITOS_E_APLICA%C3%87%C3%95ES_DA_COMPUTA%C3%87%C3%83O_EM_NUVEM> (https://www.academia.edu/23654744/CONCEITOS_E_APLICA%C3%87%C3%95ES_DA_COMPUTA%C3%87%C3%83O_EM_NUVEM)>. Acesso em: 21/06/2018.

VOGELSTEIN, F. **The Android Explosion**. Wired, Vol.19(5), May 2011.