==========

Programação Para Dispositivos Moveis

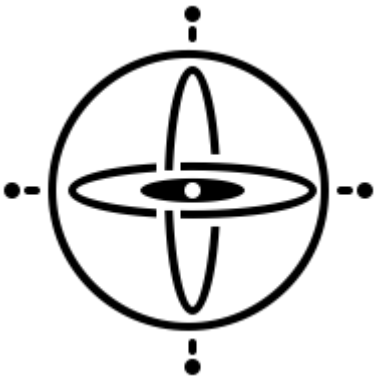
==========

Nessa Prova irá ser abordado os temas Referentes aos Trabalhos apresentados pelos Grupos, o conteúdo desse arquivo é um **COMPILADÃO** dos mesmos.

==========

Acelerômetro

==========



O acelerômetro do celular capta movimentos através de um sensor com massas sensíveis à aceleração.

Quando o dispositivo é acelerado, as massas se movem em relação ao sensor, gerando sinais elétricos que serão capturados pelo celular

Os dados coletados permitem que o aparelho celular realize algumas ações quando esses valores de coordenada chegam a certo padrão ou constante.

Os acelerômetros são compostos por:

- **Massa Sísmica:** É uma massa interna suspensa por molas dentro do acelerômetro. Ela se move em resposta à aceleração aplicada ao dispositivo.
- **Molas de Suspensão:** Fornecem a força de restauração para a massa sísmica, permitindo que ela se mova livremente e a mantenha centralizada quando em repouso.
- **Capacitores Variáveis:** Consistem em uma placa fixa e outra ligada à massa sísmica. Com o deslocamento da massa sísmica, a distância entre as placas muda.
- **Piezoelementos:** Com a aceleração, a massa sísmica pressiona os piezoelementos, gerando uma carga elétrica proporcional à força mecânica aplicada.
- **Circuitos Eletrônicos:** Processam os sinais gerados pelos outros componentes, amplificando, filtrando e convertendo os sinais analógicos em um formato digital.

==========

Persistência com SQLite (CRUD)

==========

Aplicação Cliente/Servidor: São arquiteturas de aplicação distribuídas, possuindo uma rede de fornecedores de recursos (servidores) e requerentes dos recursos oferecidos (clientes), utilizam bancos de dados para armazenar suas informações.

SQLite: formado por uma biblioteca em linguagem C, funciona como um servidor próprio e independente, não sendo necessário instalá-lo ou configurá-lo, pois gerencia os dados diretamente do seu sistema de arquivos. Não possui licença, é leve e recomendado para aplicativos desktop/mobile simples, sites leves sem muitos recursos e para sistemas com poucos usuários.

❖ SQLite *vs* MySQL

SQLite:

- Opera de forma independente;
- Consultas mais rápidas;
- Estável, multiplataforma e compatível com versões anteriores;
- Código Open Source;
- Não necessita de instalação ou configuração, guarda todo o banco de dados em um só arquivo;

MySQL:

- Suporta mais tipos de dados;
- Suporta uma maior quantidade de dados;
- Segurança melhor;
- Possui integração com a nuvem e roda em diversos sistemas operacionais;

ImageCropper: Dependência/Biblioteca para Android oferecem funcionalidades de recorte de uma imagem da galeria ou tirar uma nova com a câmera do aparelho.

CircleImageView: Dependência/Biblioteca para Android que estende a classe ImageView para exibir imagens em formato circular, adicionando um efeito de máscara à imagem original.

ScalableJava: Envolve projetar o sistema de forma que ele possa lidar com contratempos como o aumento na demanda/carga de usuários, dados e transações sem comprometer o desempenho ou a estabilidade.

==========

Câmera e Galeria de Imagens

==========

Picasso: Dependência/biblioteca de download de imagens que simplifica o processo de carregamento de imagens de URLs externas e as exibe em seu aplicativo. Reduz a quantidade de linhas de código necessárias para realizar tarefas.

CameraX: Dependência/biblioteca do **Jetpack** criada para facilitar o desenvolvimento de apps de câmera, dando suporte à visualização, análise (acesso ao buffer) e captura de imagens e vídeos.

Permissões:

- **CAMERA** (Acessar a Câmera do Dispositivo)
- **WRITE_EXTERNAL_STORAGE** (Escrever no Armazenamento Interno do Dispositivo)
- **READ_EXTERNAL_STORAGE** (Ler o Armazenamento Interno do Dispositivo)

Feature:

- **CAMERA** (Indica que o Aplicativo vai usar a Câmera, e define ela como Obrigatória para o funcionamento do mesmo)
- **Atributos MAIN**:
- **REQUEST_CAMERA_PERMISSION** (Indica a Permissão de uso da Câmera)
- **REQUEST_IMAGE_CAPTURE** (Indica a Captura da Imagem)
- **REQUEST_IMAGE_SELECT** (Indica a Imagem Selecionada)
- **REQUEST_TAKE_PHOTO** (Indica a Foto Tirada)
- **caminhoDaFoto** (Indica o Caminho da Foto Tirada)
- **imageView** (Variável da Interface usada para Mostrar a Imagem)
- **imagemCapturada** (Mapa de Pixels, variável que irá guardar a imagem)

Provider: Declarar um provedor para que outras aplicações tenham acesso aos dados da aplicação, ou o sistema não saberá da existência deles e eles não serão executados.

GPS e Maps

Foi necessário realizar um cadastro na plataforma **Google Maps** para conseguir uma **key.**

No **AndroidManifest.xml**, foi feita associação do projeto com o serviço do **Maps,** pegando a versão do **Android** e definindo a **API Key.**

AndroidManifest.xml

```
<!--        Faz associação do projeto com o serviço do Maps, pegando sua versão.-->
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
<!--        Faz associação do projeto com o serviço do Maps, define a API Key.-->
        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyCPI8EwTJunEKHnLaReLT86Ed1iciC6ITE" />
```

No método **onCreate**, foi **Setado** um **ContentView** para trazer o **Fragmento** do mapa.

PolyActivity.java

```
public class PolyActivity extends AppCompatActivity
        implements
        OnMapReadyCallback,
        GoogleMap.OnPolylineClickListener,
        GoogleMap.OnPolygonClickListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Obtêm o SupportMapFragment do activity_main e
        // aparece uma notificação quando o mapa estiver pronto para ser usado.
        setContentView(R.layout.activity_main);

        //Obtêm o SupportMapFragment e
        //aparece uma notificação quando o mapa estiver pronto para ser usado.
        SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync( callback: this);
    }
```

Em seguida, o método que adiciona as **Polilinhas** ao Mapa.

PolyActivity.java

```
58      @Override
59      public void onMapReady(GoogleMap googleMap) {
60
61          // Método que adiciona os Polilinhas ao mapa.
62          // As Polilinhas é utilizada para mostrar a rota entre os pontos.
63          Polyline polyline1 = googleMap.addPolyline(new PolylineOptions()
64                  .clickable(true)
65                  .add(
66                          new LatLng( latitude: -23.081490957878117,  longitude: -52.46503461579375),
67                          new LatLng( latitude: -23.082167049338047,  longitude: -52.46528137901965),
68                          new LatLng( latitude: -23.079793311763375,  longitude: -52.46069480163793),
69                          new LatLng( latitude: -23.080637202470054,  longitude: -52.45575953704676),
70                          new LatLng( latitude: -23.080538502075726,  longitude: -52.455732714956994),
71                          new LatLng( latitude: -23.08050395692059,  longitude: -52.455646884269726),
72                          new LatLng( latitude: -23.080513826965813,  longitude: -52.4555556891645),
73                          new LatLng( latitude: -23.080533567054108,  longitude: -52.455496680566995),
74                          new LatLng( latitude: -23.08057304722198,  longitude: -52.455416214297685),
75                          new LatLng( latitude: -23.080637202470054,  longitude: -52.455378663371995),
76                          new LatLng( latitude: -23.08075070783397,  longitude: -52.45536257011814),
77                          new LatLng( latitude: -23.081880821239416,  longitude: -52.448050868342904),
78                          new LatLng( latitude: -23.078406555870387,  longitude: -52.44222511037551),
79                          new LatLng( latitude: -23.07969954579936,  longitude: -52.44120587091821),
80                          new LatLng( latitude: -23.080854343083296,  longitude: -52.44112004023094),
81                          new LatLng( latitude: -23.080745772818155,  longitude: -52.437515151313995),
82                          new LatLng( latitude: -23.07986733747083,  longitude: -52.43712891322127),
83                          new LatLng( latitude: -23.07899876654099,  longitude: -52.43783701639127)));
84          // Armazena um objeto de dados com a polilinha.
85          polyline1.setTag("A");
86          //Estiliza o polígono.
87          stylePolyline(polyline1);
```

PolyActivity.java

```
100         // Adiciona polígonos para indicar áreas no mapa.
101         Polygon polygon1 = googleMap.addPolygon(new PolygonOptions()
102                 .clickable(true)
103                 .add(
104                         new LatLng( latitude: -23.081579787695777,  longitude: -52.46417630866293),
105                         new LatLng( latitude: -23.08067174770082,  longitude: -52.46471275045838),
106                         new LatLng( latitude: -23.081263948393097,  longitude: -52.46585537148269),
107                         new LatLng( latitude: -23.08217198438849,  longitude: -52.465329658523146)));
108         //Armazena um objeto de dados com o polígono.
109         polygon1.setTag("origem");
110         //Estiliza o polígono.
111         stylePolygon(polygon1);
```

```
// Posicione a câmera do mapa no centro da trajeto,
// e defina o fator de zoom para que a maior parte da Austrália apareça na tela.
googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng( latitude: -23.081259013406576,  longitude: -52.45183814736756),

//Definir ouvintes para eventos de clique.
googleMap.setOnPolylineClickListener(this);
googleMap.setOnPolygonClickListener(this);
```