# ****Dodatek A****

## ****Główna aktywność aplikacji - TitlePage****

|  |
| --- |
| **public** **class** TitlePage **extends** Activity[[1]](#footnote-1) {  //dodane pola klasy TitlePage  **private** SimpleOnGestureListener myGestureScanner;  DisplayMetrics mDisplayMetrics;  **long** eventTime;  **public** **static** Boolean *updateDone* = **false**;  **public** **static** **final** String *ED\_REMOTE\_REPOSITORY*  = "http://e-doswiadczenia.mif.pg.gda.pl/files/ed-android-repo/";  **public** **static** **final** String *PREFS\_UPDATE\_SUFFIX* = "\_update";  **public** **static** **final** String *PREFS\_DATE\_MODF\_SUFFIX* = "\_date";  //zmodyfikowane lub dodane metody klasy TitlePage    /\*\*  \* Called when the activity is first created.  \*\*/  @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  // Sprawdzanie rozdzielczości - musi być co najmniej 1280x752 px  /\* metody getHeight() i getWidth() są wycofywane z użytku, aby zachować  \* funkcjonalność pobranie danych dot. wymiarów ekranu wykonywane jest poprzez  \* obiekt DisplayMetrics.  \*/  mDisplayMetrics = **new** DisplayMetrics();  getWindowManager().getDefaultDisplay().getMetrics(mDisplayMetrics);    **if** (mDisplayMetrics.widthPixels < MIN\_SCREEN\_WIDTH || mDisplayMetrics.heightPixels < MIN\_SCREEN\_HEIGHT) {  Toast.*makeText*(getApplicationContext(), getString(R.string.*msg\_screen\_not\_supported*), Toast.*LENGTH\_LONG*).show();  finish();  }  ...  // Start serwera WWW i losowanie portu  nanoHTTPD = **new** NanoHTTPD(*WWW\_SERVER\_PORT*, NanoHTTPDserverRoot);  }    ...  //uruchomienie, w nowym wątku, sprawdzania czy na zdalnym serwerze znajduje się nowsza wersja edoświadczenia  **new** CheckForEDUpdates().execute(EdFileNames.*edName*);    /\*zmiana implementacji komponentu do obsługi gestów z OnGestureListener na  \* SimpleOnGestureListener  \* Ze względu na wycofywane z użytku konstruktora parametrowego  \* GestureDetector(this) klasa TitlePage nie jest implementacją OnGestureListener.  \* Zamiast tego zaimplementowano interfejs SimpleOnGestureListener.  \*/  myGestureScanner = **new** GestureDetector.SimpleOnGestureListener(){  /\* Przeniesiono kod wcześniej zaimplementowany na potrzeby klasy TitlePage.  \* Ciała metod pominięto.  \* autor Paweł Syty  \*/  **public** **boolean** onDown(MotionEvent e) {}  **public** **boolean** onFling(MotionEvent e1, MotionEvent e2, **float** velocityX, **float** velocityY) {}  **public** **void** onLongPress(MotionEvent e) {}  **public** **boolean** onScroll(MotionEvent e1, MotionEvent e2, **float** distanceX, **float** distanceY) {}  **public** **void** onShowPress(MotionEvent e) {}  // Obsluga tapniecia na ekranie  **public** **boolean** onSingleTapConfirmed(MotionEvent e) {  ...  /\* metody getHeight() i getWidth() są wycofywane z użytku, aby zachować  \* funkcjonalność pobranie danych dot. wymiarów ekranu wykonywane jest poprzez  \* obiekt DisplayMetrics.  \*/  getWindowManager().getDefaultDisplay().getMetrics(mDisplayMetrics);  Float xRel = e.getX() / mDisplayMetrics.widthPixels;  Float yRel = e.getY() / mDisplayMetrics.heightPixels;  ...  }  };  ...  /\* metoda zwrotna uruchamiana, gdy aplikacja dostanie informację o zdarzeniu  \* będącym dotknięciem ekranu.  \*/  @Override  **public** **boolean** onTouchEvent(MotionEvent me) {  /\* Ze względu na niezwykle wysoką czułość na dotknięcie wprowadzono ograniczenie  \* częstotliwości reakcji metody. Bez ograniczenia obiekt myGestureScanner  \* rejestrował 3 dotknięcia ekranu zamiast jednego. Po wprowadzeniu ograniczenia  \* aplikacja reaguje na 1 dotknięcie z zarejestrowanych w ciągu pół sekundy.  \*/  **if** (me.getEventTime() > eventTime + 500){  eventTime = me.getEventTime();  myGestureScanner.onSingleTapConfirmed(me);  **return** **true**;  }  **else**  **return** **true**;  }  ...  /\* Przeniesiono kod wcześniej zaimplementowany (w klasie ListED) na potrzeby klasy CheckForEDUpdates. Ciała metod pominięto.  \* autor Paweł Syty  \*/  /\*\*  \* Sprawdza, czy plik istnieje na zdalnym serwerze.  \*/  **private** **static** **boolean** fileExistsOnServer(String URLName) {}  /\*\*  \* Test polaczenia z siecią.  \*/  **private** **boolean** isInternetOn() {}  /\*\*  \* Klasa CheckForEDUpdates jest wewnętrzną klasą klasy TitlePage.  \*/  **public** **class** CheckForEDUpdates **extends** AsyncTask<String, Void, Void> {  /\*\*  \* Sprawdza (w osobnym wątku) czy jest uaktualnienie danego e-doświadczenia.  \* Bada, czy na serwerze jest plik ZIP o innej dacie modyfikacji niż pobrany.  \* Jeżeli tak, to oznacza, że jest uaktualnienie.  \*/  @Override  **protected** Void doInBackground(String... filenames) {  **for** (String edName : filenames) {  **long** lastModification= 0;  **try** {  **if**(!isInternetOn()){  Toast.*makeText*(getApplicationContext(), R.string.*msg\_no\_network*, Toast.*LENGTH\_LONG*).show();  **return** **null**;  }  **final** String fileURL = *ED\_REMOTE\_REPOSITORY* + File.*separator* + edName + ".zip";  // Sprawdzenie, czy plik ZIP istnieje na serwerze  **if** (!*fileExistsOnServer*(fileURL)) { **break**; }  **else**{  // Otwarcie połączenia  HttpURLConnection conn =  (HttpURLConnection) **new** URL(fileURL).openConnection();  **if**(conn.getDoInput()){  conn.setRequestMethod("HEAD");  // Pobierz datę ostatniej zmiany zdalnego pliku  lastModification = conn.getLastModified();  }  conn.disconnect();  //Zapisuje i porównuje ustawienia w Preferencjach  SharedPreferences edSavedData = getPreferences(*MODE\_PRIVATE*);  **long** edModificationDate =  edSavedData.getLong(edName + *PREFS\_DATE\_MODF\_SUFFIX*, 0);    // Zwróci zero, jak klucza nie ma  **if** (edModificationDate==0 || edModificationDate < lastModification){  // Zapisuje informację o aktualizacji do preferencji  SharedPreferences.Editor edLocalDataEditor = edSavedData.edit();  edLocalDataEditor.putBoolean(edName + *PREFS\_UPDATE\_SUFFIX*, **true**);  edLocalDataEditor.apply();  }  **else** **if** (edModificationDate > lastModification) {  // Zapisuje informację o braku aktualizacji do preferencji  SharedPreferences.Editor edLocalDataEditor = edSavedData.edit();  edLocalDataEditor.putBoolean(edName + *PREFS\_UPDATE\_SUFFIX*, **false**);  edLocalDataEditor.apply();  }  }  }  **catch** (IOException e) {  **return** **null**;  }  }  **return** **null**;  }  } |

## Aktywność ListED

|  |
| --- |
| //Uruchomienie pobierania aktualizacji,  //Uruchomienie pobierania Adobe Flash Playera,  //Ustanowienie odniesień do zasobów,  //Udostępnienie pomocy aplikacji  /\*\*  \* Wyświetlenie listy e-doświadczeń. Dziedziczy po StronaTytulowa, aby uwspólnić  \* ActionBar.  \*/  **public** **class** ListED **extends** Activity {  ...    @Override  **public** **void** onCreate(**final** Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*list\_ed*);    //pobieranie instalacji flasha, jeżeli zachodzi taka potrzeba  **if**(!*isFlashAvailable*(**this**)){  askForDownloadingFlash();  }  ...  //pobieranie Adobe Flash Playera  **private** **void** askForDownloadingFlash() {  ...  **public** **void** onClick(DialogInterface dialog,**int** which) {    Downloading flash = **new** Downloading(getApplicationContext());  flash.downloadFlash();  }  ...  }  ...  /\*\*  \* Pobiera (w osobnym wątku) dane e-doświadczenie, wyświetlając progress dialog.  \*/  **private** **class** DownloadED **extends** AsyncTask<String, Integer, Integer> {  ...  @Override  **protected** Integer doInBackground(String... edRemoteZipFileName) {  **try** {  ...  // plik do pobrania  URL url = **new** URL(TitlePage.*ED\_REMOTE\_REPOSITORY* + fileURL);    ...  // Wczytanie preferencji  SharedPreferences edLocalData = getSharedPreferences("TitlePage", *MODE\_PRIVATE*);  **long** edSavedModifiedDate = edLocalData.getLong(ED.*edSubDir*+ TitlePage.*PREFS\_DATE\_MODF\_SUFFIX*,0);  **long** tmp = ucon.getLastModified();  **if** (edSavedModifiedDate == 0 || edSavedModifiedDate != tmp) {  // Stworzenie nowego klucza bądź uaktualnienie starego  SharedPreferences.Editor edLocalDataEditor = edLocalData.edit(); edLocalDataEditor.putLong(ED.*edSubDir*+TitlePage.*PREFS\_DATE\_MODF\_SUFFIX*,tmp);  edLocalDataEditor.apply();  }  // Plik lokalny  File file = **new** File(*ED\_BASE\_DIR* + fileName);    ...  }  ...  @Override  **protected** **void** onPostExecute(Integer result) {  // Wyczyszczenie flagi "update" w preferencjach  SharedPreferences edLocalData =  getSharedPreferences("TitlePage",*MODE\_PRIVATE*);  SharedPreferences.Editor edLocalDataEditor =  edLocalData.edit();  edLocalDataEditor  .putBoolean(ED.*edSubDir* + TitlePage.*PREFS\_UPDATE\_SUFFIX*, **false**);  ...  }  ...  **private** **void** showHelp() {  AlertDialog.Builder builder;  AlertDialog alertDialog;    LayoutInflater inflater =  (LayoutInflater)ListED.**this**.getSystemService(*LAYOUT\_INFLATER\_SERVICE*);  View layout = inflater.inflate(R.layout.*help\_popup*,  (ViewGroup) findViewById(R.id.*pomoc\_popup*));  String title = getString(R.string.*txt\_title\_help*);  String body = getString(R.string.*txt\_help\_list\_ed*);  TextView text = (TextView) layout.findViewById(R.id.*text\_help\_popup*);  text.setText(Html.*fromHtml*(body));  builder = **new** AlertDialog.Builder(ListED.**this**)  .setTitle(title).setNeutralButton(getString(R.string.*btn\_close*),  **new** DialogInterface.OnClickListener() {  **public** **void** onClick(DialogInterface dialog, **int** which) {}  }).setIcon(R.drawable.*ic\_menu\_help*);  builder.setView(layout);  alertDialog = builder.create();  alertDialog.show();  } |

## Aktywność DetailsED

|  |
| --- |
| // Skalowanie tekstu  // Pobieranie podręcznika  /\*\*  \* Wyświetlenie szczegółów e-doświadczenia.  \* Dziedziczy po StronaTytulowa, aby uwspólnić ActionBar.  \*/  **public** **class** DetailsED **extends** Activity {  ...    ScaleGestureDetector myScaleGestureDetector;  MyScaleListener myListener;  // Zmienne do obsługi gestów:  **private** **static** **float** *ORG\_TEXT* = 18f;  **private** **float** newTextSize;  **private** **float** myPrevScaleFactor = 1.0f;  **private** **static** **float** *MIN\_TEXT* = 15f;  **private** **static** **float** *MAX\_TEXT* = 50f;  @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  ...    //obsługa pinch zoom  myListener = **new** MyScaleListener();  myScaleGestureDetector =  **new** ScaleGestureDetector(getApplicationContext(), myListener);    View.OnTouchListener onTouchTekst = **new** OnTouchListener() {    @Override  **public** **boolean** onTouch(View v, MotionEvent event) {    newTextSize = info.getTextSize();  myScaleGestureDetector.onTouchEvent(event);  **if**(newTextSize < *MIN\_TEXT*){ newTextSize = *MIN\_TEXT*; }  **else** **if**(newTextSize > *MAX\_TEXT*){ newTextSize = *MAX\_TEXT*; }  info.setTextSize(TypedValue.*COMPLEX\_UNIT\_SP*, newTextSize);    **return** **true**;  }  };    // Obsługa gestów  info.setOnTouchListener(onTouchTekst);    ...  }  });  //implementacja interfejsu ScaleGestureDetector  **private** **class** MyScaleListener **extends** ScaleGestureDetector.SimpleOnScaleGestureListener{    @Override  **public** **boolean** onScale(ScaleGestureDetector myDetector) {    //powiększanie tekstu  **if**(myPrevScaleFactor < myDetector.getScaleFactor()){  myPrevScaleFactor \*= myDetector.getScaleFactor();  newTextSize \*=myPrevScaleFactor;  **return** **true**;  }  //pomniejszanie tekstu  **else** **if** (myDetector.getPreviousSpan() > myDetector.getCurrentSpan()  && myDetector.getScaleFactor() > 0.4){  myPrevScaleFactor \*= myDetector.getScaleFactor();  newTextSize = myPrevScaleFactor \**ORG\_TEXT*;  **return** **true**;  }  **else**  **return** **true**;  }  }  //pomoc aplikacji  **private** **void** showHelp() {  AlertDialog.Builder builder;  AlertDialog alertDialog;    LayoutInflater inflater =  (LayoutInflater)ListED.**this**.getSystemService(*LAYOUT\_INFLATER\_SERVICE*);  View layout = inflater.inflate(R.layout.*help\_popup*,  (ViewGroup) findViewById(R.id.*pomoc\_popup*));  String title = getString(R.string.*txt\_title\_help*);  String body = getString(R.string.*txt\_help\_details\_ed*);  TextView text = (TextView) layout.findViewById(R.id.*text\_help\_popup*);  text.setText(Html.*fromHtml*(body));  builder = **new** AlertDialog.Builder(ListED.**this**)  .setTitle(title).setNeutralButton(getString(R.string.*btn\_close*),  **new** DialogInterface.OnClickListener() {  **public** **void** onClick(DialogInterface dialog, **int** which) {}  }).setIcon(R.drawable.*ic\_menu\_help*);  builder.setView(layout);  alertDialog = builder.create();  alertDialog.show();  }  } |

## Downloading - klasa obsługująca pobieranie Adobe Flash Playera

|  |
| --- |
| /\*\*  \* Klasa dziedzicząca po klasie ListED pobierająca instalator Adobe Flash Playera  \* **@author** maja  \*  \*/  **public** **class** Downloading **extends** ListED{  **private** Object tmp;  **private** DownloadManager mang;  **private** DownloadManager.Request rtmp;  **private** String link = "http://download.macromedia.com/pub/flashplayer/" +  "installers/archive/android/11.1.115.27/install\_flash\_player\_ics.apk";  **private** String externalStorageState;  **private** Context mContext;  **private** systemRespond rec;    File externalPath;  BroadcastReceiver mExternalStorageReceiver;  **boolean** mExternalStorageAvailable = **false**;  **boolean** mExternalStorageWriteable = **false**;  **private** **final** String flashPackageName = "com.adobe.flashplayer";    **public** Downloading(Context c){ mContext = c;}    //sprawdzenie czy karta SD jest dostępna i czy aplikacja ma uprawnienia do zapisu.  **void** checkExternalStorageState() {  externalStorageState = Environment.*getExternalStorageState*();  **if** (Environment.*MEDIA\_MOUNTED*.equals(externalStorageState)) {  mExternalStorageAvailable = mExternalStorageWriteable = **true**;  externalPath = Environment.*getExternalStorageDirectory*();    **if**(!externalPath.exists()){  externalPath.mkdirs();  }  }  **else** **if** (Environment.*MEDIA\_MOUNTED\_READ\_ONLY*.equals(externalStorageState)) {  mExternalStorageAvailable = **true**;  mExternalStorageWriteable = **false**;  //przekazanie informacji użytkownikowi  Toast.*makeText*(getApplicationContext(),  getString(R.string.*msg\_no\_premission*), Toast.*LENGTH\_LONG*).show();  }  **else** {  mExternalStorageAvailable = mExternalStorageWriteable = **false**;  //przekazanie informacji użytkownikowi  Toast.*makeText*(getApplicationContext(),  getString(R.string.*msg\_no\_sd*), Toast.*LENGTH\_LONG*).show();  }  }  **void** startWatchingExternalStorage(){  mExternalStorageReceiver = **new** BroadcastReceiver() {  @Override  **public** **void** onReceive(Context context, Intent intent) {  checkExternalStorageState();  }  };  IntentFilter filter = **new** IntentFilter();    filter.addAction(Intent.*ACTION\_MEDIA\_MOUNTED*);  filter.addAction(Intent.*ACTION\_MEDIA\_REMOVED*);  mContext.registerReceiver(mExternalStorageReceiver, filter);  checkExternalStorageState();  }    **void** stopWatchingExternalStorage() {  mContext.unregisterReceiver(mExternalStorageReceiver);  }    //bezparametrowa funkcja sprawdzająca czy istnieje potrzeba instalowania Adobe Flash Playera  **boolean** checkIfFlashExixts(){    **boolean** status = **false**;  List<PackageInfo> mPackages;  PackageManager mPm = mContext.getPackageManager();  **int** iter = 0;  mPackages = mPm.getInstalledPackages(0);  **while** (!status && iter < mPackages.size() ) {  status = mPackages.get(iter).packageName.equals(flashPackageName);  iter++;  }  **return** status;  }    /\*  \* bezparametrowa funkcja przygotowująca obiekt z danymi o pobieranym pliku,  \* uruchamiająca systemową usługę pobierania  \* oraz rozpoczynająca pobieranie pliku  \*/  **private** **long** prepareDownload() {    tmp = mContext.getSystemService(Context.*DOWNLOAD\_SERVICE*);  mang = (DownloadManager) tmp;  rtmp = **new** DownloadManager.Request(Uri.*parse*(link));    startWatchingExternalStorage();  **if**(mExternalStorageAvailable){  rtmp.setDestinationInExternalPublicDir("DIRECTORY\_DOWNLOADS",  s"install\_flash\_player\_ics.apk");  }  rtmp.allowScanningByMediaScanner();  rtmp.setMimeType("apk");  stopWatchingExternalStorage();  **return** mang.enqueue(rtmp);  }  /\*  \* Funkcja nasłuchująca, jako parametr pobiera nr porządkowy ściągnięcia pliku;  \* Odbiera informacje o zdarzeniach systemowych  \* i filtrująca je pod kątem zakończenia pobierania plików  \*/  **private** **void** checkDownload(**long** id) {  rec = **new** systemRespond(id);  mContext.registerReceiver(rec, **new** IntentFilter(  DownloadManager.*ACTION\_DOWNLOAD\_COMPLETE*));  }    /\*  \* funkcja instalująca pobrany plik, jako parametr pobiera nr porządkowy  \* ściągnięcia pliku  \* rejestruje nowy "zamiar",  \* przekazuje dane pobranego pliku i rozpoczyna jego instalację  \*/  **private** **void** install(**long** id) {  Intent install = **new** Intent(Intent.*ACTION\_INSTALL\_PACKAGE*);    install.setDataAndType(mang.getUriForDownloadedFile(id),  "application/vnd.android.package-archive");  install.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*);  mContext.grantUriPermission("pl.gda.pg.mif.edoswiadczenia",  mang.getUriForDownloadedFile(id), Intent.*FLAG\_GRANT\_READ\_URI\_PERMISSION*);  mContext.startActivity(install);  afterInstall();  }  **public** **void** downloadFlash() {  checkDownload(prepareDownload());  }    //funkcja sprzątająca  **void** afterInstall() {  mContext.unregisterReceiver(rec);  }    /\*  \* klasa systemRespond jest klasą wewnętrzną klasy Downloading co pozwala na  \* zapewnienie dostępu do danych przechowywanych w obiekcie Downloading  \*/  **class** systemRespond **extends** BroadcastReceiver {  **private** **long** idDown;    **public** systemRespond() {  }  /\*  \* konstruktor parametrowy pobierający nr porządkowy ściągnięcia pliku  \*/  **public** systemRespond(**long** id) {  idDown = id;  }  /\*  \* android.content  \* .BroadcastReceiver#onReceive(android.content.Context, android.content.Intent)  \* Metoda dokonująca porównania czy odebrana informacja o zdarzeniu systemowym  \* jest pożądana. Jeżeli test przebiegnie pomyślnie zostaje wywołana funkcja  \* instalująca pobrany plik.  \*/  @Override  **public** **void** onReceive(Context context, Intent intent) {  **long** id = intent.getExtras().getLong(  DownloadManager.*EXTRA\_DOWNLOAD\_ID*);  **if** (idDown == id) {  install(id);  }  }    }  } |

Dodanie pola przechowującego informacje o dacie ostatniej modyfikacji plików pobranych z zewnętrznego serwera.

|  |
| --- |
| /\*\*  \* Klasa zawierająca statyczne pola określające właściwości uruchamianego e-doświadczenia  \*\*/  **public** **class** ED {  ...  **public** **static long** *edLastModification*;  } |

**Nazwy wszystkich e-doświadczeń**

|  |
| --- |
| /\*\*  \* Klasa zawierająca niezmienalną tablicę łańcuchów znakowych. Tablica przygotowana na potrzeby sprawdzania i pobierania aktualizacji e-doświadczeń otraz tworzenia ścieżek do podręczników.  \*\*/  **public** **class** EdFileNames {  **public** **static** **final** String[] *edName* = {  "rzuty",  "wahadlo\_matematyczne",  "bryla\_sztywna",  "rownia\_pochyla",  "zderzenia\_sprezyste\_i\_niesprezyste",  "ruch\_cial\_niebieskich",  "wlasciwosci\_gazow",  "mechanika\_cieczy",  "kalorymetria",  "drgania\_mechaniczne",  "laboratorium\_dzwieku",  "pole\_elektryczne",  "kondensatory",  "obwody\_pradu\_stalego",  "pole\_magnetyczne",  "cewki\_i\_indukcja",  "uklady\_RLC",  "optyka\_geometryczna",  "lawa\_optyczna",  "interferencja\_i\_dyfrakcja",  "eksperymenty\_myslowe\_einsteina",  "korpuskularna\_natura\_swiatla",  "fizyka\_atomowa"};  } |

## Dodanie powiązania do ikon:

|  |
| --- |
| <Button  ...  android:drawableLeft=*"@drawable/ic\_pole\_elektryczne"*/>  <Button  ...  android:drawableLeft=*"@drawable/ic\_wkrotce"*/>  <!--android:drawableLeft="@drawable/ic\_prad\_staly"/>-->  <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_dzwiek"/>-->  <Button  ...  android:drawableLeft="@drawable/ic\_kalorymetria"/>    <Button  ...  android:drawableLeft="@drawable/ic\_kondensatory"/>    <Button  ...  android:drawableLeft="@drawable/ic\_pole\_magnetyczne"/>    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_cewki"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_optyka\_geom"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_rlc"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_korpuskularna"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!--android:drawableLeft="@drawable/ic\_interferencja"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_wkrotce"/>  <!-- android:drawableLeft="@drawable/ic\_f\_atomowa"/>-->    <Button  ...  android:drawableLeft="@drawable/ic\_gazy"/> |

1. Usunięto „implements on gesture listener” [↑](#footnote-ref-1)