Opis aplikacji i platformy

http://developer.android.com/guide/components/fundamentals.html

Technologia

android sdk

java dl androida

informacje o wykorzystywanych urządzeniach i ich wersjach androida

http://developer.android.com/about/dashboards/index.html

porównanie wersji podstawowej aplikacji do nowych:

<http://developer.android.com/about/versions/android-3.0-highlights.html>

<http://developer.android.com/about/versions/android-4.0-highlights.html>

<http://developer.android.com/about/versions/jelly-bean.html>

narzędzia

testowanie:

emulator

rzczywiste urządzenie

1. co to są intent'y

2. zabezpieczenia w androidzie – uprawnienia aplikacji

3. sposób instalowania aplikacji

3. laczenie się z internetem

Aktualizacja na podstawie zmiany rozmiaru

Aktualizacja na podstawie zmiany daty ostatniej modyfikacji

http://developer.android.com/training/building-connectivity.html

Wykorzystanie sharedPreferences

Przygotowanie filmików i grafik na potrzeby aplikacji

Wielowątkowość:

Opisać możliwe rozwiązania I wybrane rozwiązanie:

Android-Multithreaded-Programming-1.pdf

Have inner class implement Runnable

# Rozdział 2

1. **Losowanie portu serwera WWW**
   1. **Zrobione - opisać**
2. **pobieranie i instalacja oprogramowania do obsługi technologii flash**
   1. **zrobione - opisać**
3. **mechanizm aktualizacji e-doświadczeń**
   1. **zrobione - opisać**
4. **Dodanie zrealizowanych e-doświadczen**
   1. **Lista dodanych eD.**
   2. **Ze scorm wyciągnąć katalog content, podmienić plik z intro na pusty filmik. I stworzyć zipy.**
   3. **W trakcie – opisać?**
5. **Poprawić scieżki do podręczników.**
   1. **opisac**
6. **Uzupełnienie opisów oraz przykładowych ćwiczeń**
   1. **Zrobione - opisać**
7. **Lista ed dla których wstawiono opisy.**
   1. **Zrobione - opisać**
8. **pomoc aplikacji**
   1. **Zrobione - opisać**
9. **pionowe przewijanie zdjęć w oknach popup wraz informacjami o realizatorach projektu**
10. **Poprawienie grafiki aplikacji wyświetlanej podczas pracy z uruchomionym e-doświadczeniem.**
    1. **Poprawione - opisać**
11. **Powiększanie oraz pomniejszanie opisów pojawiających się w aplikacji przy pomocy gestów.**
    1. **Ewentualnie poprawić płynność**
12. **Analiza pojawiania się i przyczyny błędów związanych z Html.fromHtml() występujących na Androidzie 4.0 i wcześniejszych wersjach systemu.**
    1. **Nie udaje się uzyskać**
13. **Analiza błędów zwiazane z obsługą tapnięć w komponencie webview.**
    1. **Pomimo usilnych prób nie udało się wywołać błędu związanego z Html.fromHtml()**
    2. **Nie udaje się uzyskać**
14. **Analiza sposobów modyfikacji grafiki aplikacji dedykowanej dla tabletów o wysokiej rozdzielczości(powyżej 1280x800).**
    1. **Nie da się bez grafik o lepszej rozdzielczości**
15. **Analiza sensowności umieszczania w aplikacji e-doświadczeń wymagających dużej mocy obliczeniowej (wahadło matematyczne, ruch ciał niebieskich).**

# Rozdział 3

Podsumowanie i prezentacja wyników

Przeprowadzone testy. Rodzaje i wyniki.

Możliwości testowania przy pomocy emulatora.

# Zakończenie

Podsumowanie wykonanych zadań i wnioski.

Dyskusja nad wycofaniem flasha z aplikacji mobilnych.

# Dodatek A

Dodatkowe informacje

Bibliografia

Lista linków i publikacji

# slowo wstepne

Manifest zawiera podstawowe, najważniejszy zbiór informacji o aplikacji, który jest potrzebny platformie Android, aby tą aplikację uruchomić. Mianowicie:

**Services**

**Quickview**

### Implementing a user interface

The user interface for an activity is provided by a hierarchy of views—objects derived from the [**View**](http://developer.android.com/reference/android/view/View.html) class. Each view controls a particular rectangular space within the activity's window and can respond to user interaction. For example, a view might be a button that initiates an action when the user touches it.

Android provides a number of ready-made views that you can use to design and organize your layout. "Widgets" are views that provide a visual (and interactive) elements for the screen, such as a button, text field, checkbox, or just an image. "Layouts" are views derived from [**ViewGroup**](http://developer.android.com/reference/android/view/ViewGroup.html) that provide a unique layout model for its child views, such as a linear layout, a grid layout, or relative layout. You can also subclass the [**View**](http://developer.android.com/reference/android/view/View.html) and [**ViewGroup**](http://developer.android.com/reference/android/view/ViewGroup.html)classes (or existing subclasses) to create your own widgets and layouts and apply them to your activity layout.

The most common way to define a layout using views is with an XML layout file saved in your application resources. This way, you can maintain the design of your user interface separately from the source code that defines the activity's behavior. You can set the layout as the UI for your activity with [**setContentView()**](http://developer.android.com/reference/android/app/Activity.html#setContentView(int)), passing the resource ID for the layout. However, you can also create new [**View**](http://developer.android.com/reference/android/view/View.html)s in your activity code and build a view hierarchy by inserting new [**View**](http://developer.android.com/reference/android/view/View.html)s into a [**ViewGroup**](http://developer.android.com/reference/android/view/ViewGroup.html), then use that layout by passing the root [**ViewGroup**](http://developer.android.com/reference/android/view/ViewGroup.html) to[**setContentView()**](http://developer.android.com/reference/android/app/Activity.html#setContentView(android.view.View)).

### Declaring the activity in the manifest

You must declare your activity in the manifest file in order for it to be accessible to the system. To declare your activity, open your manifest file and add an [**<activity>**](http://developer.android.com/guide/topics/manifest/activity-element.html) element as a child of the [**<application>**](http://developer.android.com/guide/topics/manifest/application-element.html) element. For example:

<manifest ... >

<application ... >

<activity android:name=".ExampleActivity" />

...

</application ... >

...

</manifest >

There are several other attributes that you can include in this element, to define properties such as the label for the activity, an icon for the activity, or a theme to style the activity's UI. The [**android:name**](http://developer.android.com/guide/topics/manifest/activity-element.html#nm) attribute is the only required attribute—it specifies the class name of the activity. Once you publish your application, you should not change this name, because if you do, you might break some functionality, such as application shortcuts (read the blog post, [Things That Cannot Change](http://android-developers.blogspot.com/2011/06/things-that-cannot-change.html)).

See the [**<activity>**](http://developer.android.com/guide/topics/manifest/activity-element.html) element reference for more information about declaring your activity in the manifest.

#### Using intent filters

An [**<activity>**](http://developer.android.com/guide/topics/manifest/activity-element.html) element can also specify various intent filters—using the [**<intent-filter>**](http://developer.android.com/guide/topics/manifest/intent-filter-element.html) element—in order to declare how other application components may activate it.

When you create a new application using the Android SDK tools, the stub activity that's created for you automatically includes an intent filter that declares the activity responds to the "main" action and should be placed in the "launcher" category. The intent filter looks like this:

<activity android:name=".ExampleActivity" android:icon="@drawable/app\_icon">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

The [**<action>**](http://developer.android.com/guide/topics/manifest/action-element.html) element specifies that this is the "main" entry point to the application. The [**<category>**](http://developer.android.com/guide/topics/manifest/category-element.html) element specifies that this activity should be listed in the system's application launcher (to allow users to launch this activity).

If you intend for your application to be self-contained and not allow other applications to activate its activities, then you don't need any other intent filters. Only one activity should have the "main" action and "launcher" category, as in the previous example. Activities that you don't want to make available to other applications should have no intent filters and you can start them yourself using explicit intents (as discussed in the following section).

However, if you want your activity to respond to implicit intents that are delivered from other applications (and your own), then you must define additional intent filters for your activity. For each type of intent to which you want to respond, you must include an [**<intent-filter>**](http://developer.android.com/guide/topics/manifest/intent-filter-element.html) that includes an [**<action>**](http://developer.android.com/guide/topics/manifest/action-element.html) element and, optionally, a[**<category>**](http://developer.android.com/guide/topics/manifest/category-element.html) element and/or a [**<data>**](http://developer.android.com/guide/topics/manifest/data-element.html) element. These elements specify the type of intent to which your activity can respond.

You can start another activity by calling [**startActivity()**](http://developer.android.com/reference/android/app/Activity.html#startActivity(android.content.Intent)), passing it an [**Intent**](http://developer.android.com/reference/android/content/Intent.html) that describes the activity you want to start. The intent specifies either the exact activity you want to start or describes the type of action you want to perform (and the system selects the appropriate activity for you, which can even be from a different application). An intent can also carry small amounts of data to be used by the activity that is started.

When working within your own application, you'll often need to simply launch a known activity. You can do so by creating an intent that explicitly defines the activity you want to start, using the class name. For example, here's how one activity starts another activity named **SignInActivity**:

Intent intent = new Intent(this, SignInActivity.class);

startActivity(intent);

However, your application might also want to perform some action, such as send an email, text message, or status update, using data from your activity. In this case, your application might not have its own activities to perform such actions, so you can instead leverage the activities provided by other applications on the device, which can perform the actions for you. This is where intents are really valuable—you can create an intent that describes an action you want to perform and the system launches the appropriate activity from another application. If there are multiple activities that can handle the intent, then the user can select which one to use. For example, if you want to allow the user to send an email message, you can create the following intent:

Intent intent = new Intent(Intent.ACTION\_SEND);

intent.putExtra(Intent.EXTRA\_EMAIL, recipientArray);

startActivity(intent);

The [**EXTRA\_EMAIL**](http://developer.android.com/reference/android/content/Intent.html#EXTRA_EMAIL) extra added to the intent is a string array of email addresses to which the email should be sent. When an email application responds to this intent, it reads the string array provided in the extra and places them in the "to" field of the email composition form. In this situation, the email application's activity starts and when the user is done, your activity resumes.

Intents can be divided into two groups:

* *Explicit intents* designate the target component by its name (the [component name field](http://developer.android.com/guide/components/intents-filters.html#cname), mentioned earlier, has a value set). Since component names would generally not be known to developers of other applications, explicit intents are typically used for application-internal messages — such as an activity starting a subordinate service or launching a sister activity.
* *Implicit intents* do not name a target (the field for the component name is blank). Implicit intents are often used to activate components in other applications.

Android delivers an explicit intent to an instance of the designated target class. Nothing in the Intent object other than the component name matters for determining which component should get the intent.

A different strategy is needed for implicit intents. In the absence of a designated target, the Android system must find the best component (or components) to handle the intent — a single activity or service to perform the requested action or the set of broadcast receivers to respond to the broadcast announcement. It does so by comparing the contents of the Intent object to *intent filters*, structures associated with components that can potentially receive intents. Filters advertise the capabilities of a component and delimit the intents it can handle. They open the component to the possibility of receiving implicit intents of the advertised type. If a component does not have any intent filters, it can receive only explicit intents. A component with filters can receive both explicit and implicit intents.

Only three aspects of an Intent object are consulted when the object is tested against an intent filter:

action   
data (both URI and data type)   
category

The extras and flags play no part in resolving which component receives an intent.

### Intent filters

To inform the system which implicit intents they can handle, activities, services, and broadcast receivers can have one or more intent filters. Each filter describes a capability of the component, a set of intents that the component is willing to receive. It, in effect, filters in intents of a desired type, while filtering out unwanted intents — but only unwanted implicit intents (those that don't name a target class). An explicit intent is always delivered to its target, no matter what it contains; the filter is not consulted. But an implicit intent is delivered to a component only if it can pass through one of the component's filters.

A component has separate filters for each job it can do, each face it can present to the user. For example, the NoteEditor activity of the sample Note Pad application has two filters — one for starting up with a specific note that the user can view or edit, and another for starting with a new, blank note that the user can fill in and save. (All of Note Pad's filters are described in the [Note Pad Example](http://developer.android.com/guide/components/intents-filters.html#npex) section, later.)

## Filters and security

An intent filter cannot be relied on for security. While it opens a component to receiving only certain kinds of implicit intents, it does nothing to prevent explicit intents from targeting the component. Even though a filter restricts the intents a component will be asked to handle to certain actions and data sources, someone could always put together an explicit intent with a different action and data source, and name the component as the target.

An intent filter is an instance of the [**IntentFilter**](http://developer.android.com/reference/android/content/IntentFilter.html) class. However, since the Android system must know about the capabilities of a component before it can launch that component, intent filters are generally not set up in Java code, but in the application's manifest file (AndroidManifest.xml) as [**<intent-filter>**](http://developer.android.com/guide/topics/manifest/intent-filter-element.html) elements. (The one exception would be filters for broadcast receivers that are registered dynamically by calling[**Context.registerReceiver()**](http://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter, java.lang.String, android.os.Handler)); they are directly created as IntentFilter objects.)

A filter has fields that parallel the action, data, and category fields of an Intent object. An implicit intent is tested against the filter in all three areas. To be delivered to the component that owns the filter, it must pass all three tests. If it fails even one of them, the Android system won't deliver it to the component — at least not on the basis of that filter. However, since a component can have multiple intent filters, an intent that does not pass through one of a component's filters might make it through on another.

### Using intent matching

Intents are matched against intent filters not only to discover a target component to activate, but also to discover something about the set of components on the device. For example, the Android system populates the application launcher, the top-level screen that shows the applications that are available for the user to launch, by finding all the activities with intent filters that specify the "**android.intent.action.MAIN**" action and "**android.intent.category.LAUNCHER**" category (as illustrated in the previous section). It then displays the icons and labels of those activities in the launcher. Similarly, it discovers the home screen by looking for the activity with "**android.intent.category.HOME**" in its filter.

Your application can use intent matching is a similar way. The [**PackageManager**](http://developer.android.com/reference/android/content/pm/PackageManager.html) has a set of **query...()**methods that return all components that can accept a particular intent, and a similar series of **resolve...()**methods that determine the best component to respond to an intent. For example, [**queryIntentActivities()**](http://developer.android.com/reference/android/content/pm/PackageManager.html#queryIntentActivities(android.content.Intent, int))returns a list of all activities that can perform the intent passed as an argument, and [**queryIntentServices()**](http://developer.android.com/reference/android/content/pm/PackageManager.html#queryIntentServices(android.content.Intent, int))returns a similar list of services. Neither method activates the components; they just list the ones that can respond. There's a similar method, [**queryBroadcastReceivers()**](http://developer.android.com/reference/android/content/pm/PackageManager.html#queryBroadcastReceivers(android.content.Intent, int)), for broadcast receivers.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

1. **Losowanie portu serwera WWW**

**Aplikacja uzyskuje dostęp do zasobów sieciowych poprzez wysokopoziomowe mechanizmy komunikacji tj.: dostarczycieli** URLs i URLConnections. Zdarzają się jednak sytuacje, w których aplikacja wymagane jest korzystanie z komunikacji niskopoziomowej.

Dzieje się tak w przypadku aplikacji typu klient – serwer. Serwer udostępnia pewne usługi, takie jak: przetwarzanie zapytań bazodanowych. Klient zajmuje się przetwarzaniem danych pochodzących z udostępnionych przez serwer usług. Komunikacja pomiedzy klientem a serwerem musi być niezawodna. Oznacza to, że każdorazowo zapewniony jest niezmienny sposób przesyłania danych. Zunifikowane protokoły komunikacyjne zapewniają o wierzytelności tej komunikacji. Jednym z nich jest protokół TCP zapewniający wiarygodną komunikację między odbiorca a nadawcą i szeroko używany przez aplikacje mobilne do wzajemnej komunikacji. Klient i serwer komunikują się poprzez ustanowiony kanał komunikacyjny po uruchomieniu nasłuchu na nim. Nasłuch odbywa się za pomocą portów (gniazd) na stałe związanych z danym połączeniem. Gnizdo jest zakończeniem dwukierunkowej komunikacji, każdy uczestnik komunikacji dysponuje własnym portem, który po związaniu z kanałem staje się zablokowany dla innych „użytkowników”. **[http://docs.oracle.com/javase/tutorial/networking/sockets/index.html]**

[http://tools.ietf.org/html/rfc791] [http://www.w3.org/Protocols/]

**W aplikacji zostały użyte komponenty wymagające uruchomienie lokalnego serwera WWW np.: komponent webView. Ze względu na zastosowane rozwiązania konieczne jest uruchomienie lokalnego serwera WWW – komponent webView.**

**W przypadku niefortunnego wylosowania zajętego już portu dokonywane jest następne losowanie zabezpieczające prawidłowe działanie aplikacji.**

**Ponieważ sysytem operacyjny może wykorzystywać w/w serwer do działań drugoplanowych istotne jest wybranie portu, który jeszcze nie jest wykorzystywany. Wybór portu serwera WWW odbywa się poprzez wylosowanie liczby z zakresu 0 – (216-1).** 65535 jest liczbą często występująca w informatyce, ponieważ jest to mkasymalna liczba, która, może być reprezentowana przez 16 bitową liczbę bez znaku binarnego. W protokołach internetowych, 65535 jest to także liczba portów TCP i UDP, dostępnych dla adresie IP.

|  |
| --- |
| **try** {  //**TODO** gniazdo  portRandomization();  //Losowanie portu  // Start serwera WWW    nanoHTTPD = **new** NanoHTTPD(*WWW\_SERVER\_PORT*, NanoHTTPDserverRoot);  }  //jeśli zajety port  **catch**(BindException bEx){  portRandomization();  }  //jeżli pojawią się ewentualne błędy podczas ustanawiania kanału komunikacji  **catch** (IOException ioe) {  tworzenia socket  /\*myDialog mDialog = myDialog.myDialog();  mDialog.setTitle(getString(R.string.msg\_title\_error));  mDialog.setMessage(getString(R.string.msg\_internal\_error001));\*/  } |

|  |
| --- |
| /\*  \* "A valid port value is between 0 and 65535.  \*/  **private** **void** portRandomization(){  Random rdn = **new** Random();  **int** tmp = rdn.nextInt();  **if** (tmp > *MAX\_PORT* || tmp < 0){  *WWW\_SERVER\_PORT* = Math.*abs*(tmp % *MAX\_PORT*);  }  **else**{  *WWW\_SERVER\_PORT* = tmp;  }  /\*  String text = String.*valueOf*(*WWW\_SERVER\_PORT*);  Log.*i*(*TAG*,text);  \*/  } |

**Port oznaczony nr 0 jest zastrzeżony i zaleca się nieuzywanie go do komunikacji TPC lub UDP. Czasami jednak można wykorzystać go do dynamicznego zaalokowania portu. Programista zamiast wskazywac konkretny port ajko docelowy do danej komunikacji może podać jako parametr połączenia port 0 (np. Unix). Wówczas system operacyjny dynamicznie zaalokuje pierwszy nieobsadzony port. Istotną kwestią jest to, aby system przewidywał obsługę portu 0 w opisany powyżej sposób.[ http://compnetworking.about.com/od/tcpip/p/port-numbers-0.htm]**

**[http://javasourcecode.org/html/open-source/jdk/jdk-6u23/java.net/InetSocketAddress.java.html]**

**Aby skorzystać z dynamiczniej alokacji wolnego portu przez system, w stystemie Android należy użyć klasy ServerSocket przestrzeni** java.lang.Object**. Podczas konstrukcji nowego obiektu tej klasy, z parametrem port = 0, system dynamicznie przypisze port do gniazda. [**Constructs a new **ServerSocket** instance bound to the given **port**. The backlog is set to 50. If **port == 0**, a port will be assigned by the OS.**]**

<http://developer.android.com/reference/java/net/ServerSocket.html>

**Biblioteka** NanoHTTPD uż**ywana w aplikacji nie przewidywała obsługi dynamicznego alokowania portu przez system operacyjny.**

|  |
| --- |
| /\*\*  \* Starts a HTTP server to given port.<p> Throws an IOException if the  \* socket is already in use  \*/      **public** NanoHTTPD(**int** port, File wwwroot) **throws** IOException {  myTcpPort = port;  **this**.myRootDir = wwwroot;  myServerSocket = **new** ServerSocket(myTcpPort);  …  } |

**Zmodyfikowano parametrowy konstruktor klasy** NanoHTTPD, aby stale korzystał z aktualnych danych przechowywanych przez obiekt klasy ServerSocket. W**ykorzystując port 0 zyskujemy na prostocie i przejrzystości kodu, nie ma potrzeby zabezpieczania się przed błędami wynikającymi z próby podpięcia się do zajętego portu -** BindException**.**

|  |
| --- |
| **public** NanoHTTPD(**int** port, File wwwroot) **throws** IOException {  **this**.myRootDir = wwwroot;  myServerSocket = **new** ServerSocket(port);  myTcpPort = myServerSocket.getLocalPort();  …  } |

1. **pobieranie i instalacja oprogramowania do obsługi technologii flash**
2. **mechanizm aktualizacji e-doświadczeń**
3. **Dodanie zrealizowanych e-doświadczen- zrobione**
4. **Uzupełnienie opisów, przykładowych ćwiczeń oraz pomocy aplikacji**

**W mobilnej aplikacji** projektu "e-Doświadczenia w fizyce" uzupełnione zostały:

1. opisy zjawisk fizycznych przybliżanych przez kolejne e-doświadczenia,
2. **przykładowe ćwiczenia.**

**Aplikacja umożliwa użytkownikowi pracę w języku angielskim lub poskim, dlatego też opracowane do doświadczeń opisy i ćwiczenia napisane zostały w dwóch wersjach językowych.**

**Do następujących e-doświadczeń dodano w języku polskim brakujące treści:**

1. **Właściwości cieczy**
2. **Drgania mechaniczne**
3. **Pole elektryczne**
4. **Obwody prądu stałego**
5. **Laboratorium dźwięku**
6. **Kalorymetria**
7. **Kondensatory**
8. **Pole magnetyczne**
9. **Cewki i indukcja**
10. **Optyka geometryczna**
11. **Układy RLC**
12. **Korpuskuralna natura światła i materii**
13. **Interferencja i dyfrakcja**
14. **Fizyka atomowa**

**W angielskiej wersji językowej uzupełniono treści związane ze wszystkimi 23 e-doświadczeniami. W dodatku B zostały zebrane opisy i ćwiczenia napisane w języku polskim, zaś dodatku C – w języku angielskim. Pozostałe teksty zostały umieszczone w dodatku D. Zaliczają się do nich:**

* **przetłumaczone na język angielski:**
  + **komunikaty informujące użytkownika o stanie aplikacji,**
  + **informacje dot. realizatorów projektu,**
  + **treści plansz wspomagających poruszanie się po aplikacji mobilnej,**
* **napisane w języku polskim:**
  + **treści plansz wspomagających poruszanie się po aplikacji mobilnej.**

**Ponieważ autor aplikacji umieścił w niej tekst pomocy dla ekranu tytułowego pozostałe plansze wspomagające korzystanie z aplikacji zostały utrzymane w podobnej konwencji.**

1. **Poprawienie grafiki aplikacji wyświetlanej podczas pracy z uruchomionym e-doświadczeniem**

**Zaprezentowana poniżej migawka (Rys. xxx) przedstawia wygląd fragmentu ekranu aplikacji przed wprowadzeniem poprawki. Wzór znajdujący się na bitmapie wyświetlanej obok uruchomionego e-doświadczenia jest przesunięty względem odpowiadającego mu motywu kolorystycznego e-doświadczenia. Naprawienie uszkodzonej grafiki rastowej sprowadziło się do doklejenia, dopasowanego kolorystycznie paska o szerokości 3 px, do jej dolnej krawędzi. Następnie powiększone tło skadrowano do poprzednich wymiarów. Kolejna migawka (Rys. vvv) pokazuje omawiany fragment po dokładnym dopasowaniu do uruchamianego w aplikacji e-doświadczenia. Motywy kolorytyczne obydwu komponentów stanowaią kontinuum w ramach aplikacjji mobilnej.**

1. **Powiększanie oraz pomniejszanie opisów pojawiających się w aplikacji przy pomocy gestów - zrobione - ewentualnie poprawić płynność**
2. **Analiza pojawiania się i przyczyny błędów związanych z Html.fromHtml() występujących na Androidzie 4.0 i wcześniejszych wersjach systemu. nie udaje się uzyskać**
3. **Analiza błędów zwiazane z obsługą tapnięć w komponencie webview - nie udaje się uzyskać**
4. **Analiza sposobów modyfikacji grafiki aplikacji dedykowanej dla tabletów o wysokiej rozdzielczości(powyżej 1280x800). - nie da się bez grafik o dużo lepszej rozdzielczości niż te zawarte w apliakcji.**
5. **pionowe przewijanie zdjęć w oknach popup wraz informacjami o realizatorach projektu - niewykonane**