AFAndroid & AFWinPhone

Uživatelská příručka

Pavel Matyáš
5.4.2016

Úvod

AFAndroid a AFWinPhone jsou frameworky po řadě pro Android a Windows Phone aplikace umožňují postavit dynamicky v mobilní aplikaci některé prvky grafického uživatelského rozhraní. Popis UI je generován na serveru, který ho poskytuje například pomocí REST webových služeb. Oba frameworky tento popis získávají pomocí http dotazu na určité URL adresy a na základě těchto informací UI interpretují. Tato uživatelský příručka popisuje ovládání pouze těchto klientských frameworků, způsob generování UI na serveru je popsaný v uživatelské příručce pro Framework AFSwinx, ze kterého frameworky vycházejí.

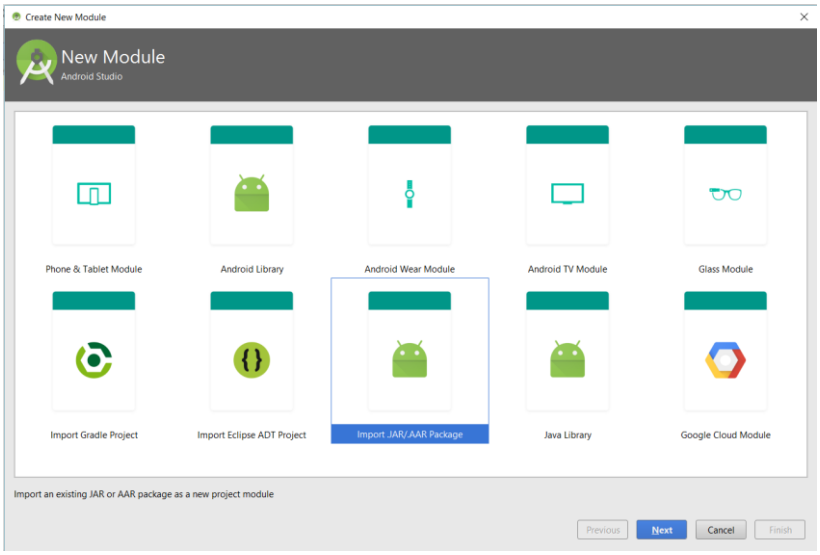
Použití frameworků

V této části si popíšeme, jak frameworky přidat do projektů a jak se frameworky používají a co vše je třeba k použití znát a mít připraveno.

Závislosti - Android

Pro použití frameworku na Android platformě je nutno přidat do aplikace AAR soubor, který framework obsahuje. Popíšeme zde způsob přidání ve vývojovém prostředí Android Studio.

1. Rozklikněte pravým tlačítlkem myši kořenový adresář projektu
2. Vyberte **New -> Module**
3. Otevře se dialog, ve kterém vyberte možnost **Include .JAR/.AAR Package** a klikněte na **Next** (jak lze vidět na obrázku č. 1)
4. Zadejte cestu k AAR souboru, který chcete do projektu zahrnout
5. Případně zadejte vlastní název importovaného modulu a klikněte na **Finish**



Obrázek č. 1 – Tvorba nového modulu.

Po provedení těchto akcí se vytvoří nový balíček, obsahující AAR soubor, IML soubor, popisující modul a soubor build.gradle, ve kterém jsou důležité následující řádky

```
configurations.create("<název konfigurace>")
artifacts.add("<název konfigurace>", file('<jméno souboru>.aar'))
```

Zároveň se do závislostí v **build.gradle** v modulu s aplikací přidá červeně vyznačený řádek v následující ukázce kódu.

```
dependencies {
    compile fileTree(include: ['*.jar', '*.aar'], dir: 'libs')
        … další závislosti
    compile project(":<název importovaného modulu>")
}
```

Nakonec do **settings.gradle** v kořenovém adresáři projektu přibude include

```
include ':<název modulu s aplikací>', ':<název importovaného modulu>'
```

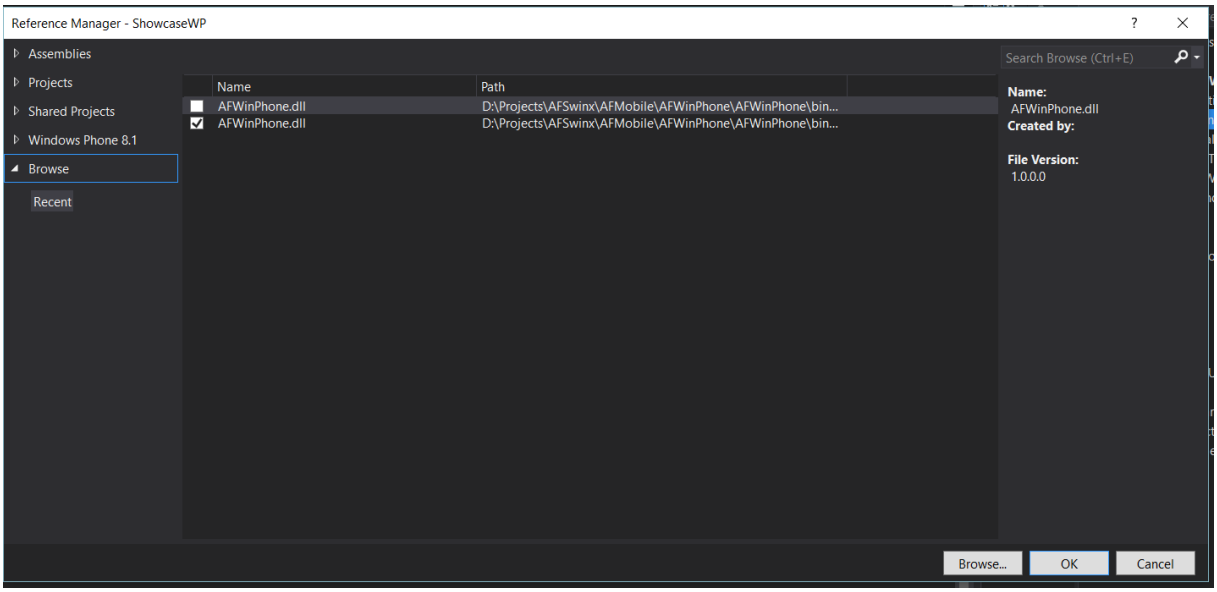
Nyní stačí provést gradle build přes **Build-> Build project** v hlavní liště IDE a lze knihovnu používat.

Závislosti – Windows Phone

Pro využití ve Windows Phone aplikacích je nutné přidat DLL knihovnu s frameworkem. Popíšeme způsob přidání ve vývojovém prostředí Visual Studio 2015.

Ve Visual Studiu má každá aplikace ve svém kořenovém adresáři položku References, která obsahuje knihovny používané aplikací.

1. Klikněte pravým tlačítkem na **References** a poté na **Add Reference**.
2. Objeví se dialog, kde klikněte na Browse… a vyhledejte příslušný DLL soubor
3. Zkontrolujte, zda je nalezený soubor v dialogu zaškrtnutý (tak jak lze vidět na obrázku č. 2)
4. Povtrďte volbu stisknutím **OK**



Obrázek č. 2 – Import DLL knihovny přes Reference Managera ve VS 2015

I zde je potřeba provést **Build -> Build Solution** a je poté možné knihovnu použít.

Základní použití

Oba frameworky generují dva typy prvků UI – formulář a list (neboli seznam položek). Tyto komponenty lze vložit do jakéhokoliv jiného prvku UI, a tedy lze použít framework pouze v části UI aplikace. Zbytek UI může být vytvořen jakýmkoliv jiným způsobem dle uvážení vývojáře.

Nutné předpoklady

Nutným předpokladem je mít server, který využívá framework AFSwinx, kterému se tato uživatelská příručka nevěnuje. Aby bylo jasné, proč se frameworky využívají právě tímto způsobem, je doporučené si tuto část v druh uživatelské příručce přečíst.

Velmi zkráceně, předpokládáme existenci serveru, který poskytuje přes webové služby (například pomocí REST) popis uživatelského rozhraní, který se zde za pomocí knihovny AFSwinx (využívající mimochodem k tomu účelu knihovnu AspectFaces) generuje z modelů. Tato data oba frameworky AFAndroid a AFWindowsPhone získávají pomocí http dotazu na definované zdroje webových služeb. Získají tak data ve formátu JSON, které frameworky zpracují a automaticky na jejich základě vytvoří formulář či list, který lze vložit do kterékoliv části UI nebo s nimi dále pracovat.

Existují tři základní zdroje, které by měl server poskytovat, aby byl klient schopen vytvořit plnohodnotnou funkční komponentu.

- Zdroj definice komponenty
- Zdroj dat, kterými se má prvek naplnit (pokud chceme komponentu naplnit)
- Zdroje akcí (vytvoření, aktualizace atp.)

Klient musí nadefinovat, kde tyto zdroje na serveru najde. Tomu se věnuje následující kapitola.

Definice zdrojů

Frameworky předpokládají určitý formát, ve kterém je potřeba tyto zdroje nadefinovat. K tomuto účelu je nutné v aplikaci vytvořit XML soubor, který bude definovat zdroje. Následující ukázka kódu ukazuje definici všech tří zmíněných zdrojů pro profilový formulář.

```
<?xml version="1.0" encoding="UTF-8"?>
<connectionRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<connection id="personProfile">
    <metaModel>
      <endPoint>toms-cz.com</endPoint>
      <endPointParameters>/AFServer/rest/users/profile</endPointParameters>
      <protocol>http</protocol>
      <port></port>
      <header-param>
        <param>content-type</param>
        <value>Application/Json</value>
      </header-param>
    </metaModel>
    <data>
      <endPoint>toms-cz.com</endPoint>
      <endPointParameters>/AFServer/rest/users/user/#{username}</endPointParameters>
      <protocol>http</protocol>
      <port></port>
      <security-params>
        <security-method>basic</security-method>
        <userName>#{username}</userName>
        <password>#{password}</password>
      </security-params>
    </data>
    <send>
      <endPoint>toms-cz.com</endPoint>
      <endPointParameters>/AFServer/rest/users/update</endPointParameters>
      <protocol>http</protocol>
      <method>post</method>
      <port></port>
      <security-params>
        <security-method>basic</security-method>
```

```
        <userName>#{username}</userName>
        <password>#{password}</password>
      </security-params>
    </send>
  </connection>

</connectionRoot>
```

- **<connectionRoot>** je uzel obsahující všechna definovaná připojení
- **<connection id="název připojení">** určuje jedno dané připojení
    - Parametr **id** toto připojení pojmenovává a pomocí něj se v kódu odkazujeme na dané připojení
- V uzlu <**metaModel>** je nadefinován zdroj definice komponenty
    - Pomocí uzlů <endPoint>, <endPointParameters>, <port> a <protocol> nadefinujeme URL adresu zdroje, ze kterého se definice získá.
    - URL adresa se vytvoří v následujícím tvaru *<protocol>://<endPoint>:<port>/<endPointParameters>*
    - Z ukázkové definice zdrojů se v případě uzlu <metaModel> vytvoří URL http://toms-cz.com/AFServer/rest/users/profile (port se neuvažuje neboť je prazdný)
    - Uzel <header-params> určuje další parametry pro připojení – např. očekáváný formát dat
        - V ukázkové definici se definuje, že očekáváme data ve formátu JSON
        - Prozatím nejsou jiné formáty než JSON podporovány
- Uzel **<data>** určuje zdroj pro data, kterými se komponenta naplní
    - URL adresa je definována stejným způsobem, jako je popsáno výše
    - Některé zdroje mají omezený přístup z hlediska bezpečnosti, tedy pouze někteří uživatelé mohou data ze zdroje získat – k tomuto účelu slouží uzel <**security-params>**
        - V <security-params> lze nastavit v uzlu <security-method> , který typ autentifikace uživatele se používá.
            - Prozatím frameworky počítají s BASIC autentifikací
        - <userName> určuje uživatelské jméno uživatele, který se snaží zdroj získat.
        - <password> definuje heslo uživatele, který se snaží zdroj získat.
        - Hodnoty <username> a <password> musí vývojář dodat, jak si popíšeme později.
- Uzel **<send>** definuje zdroj, na který se data z komponenty odešlou – v případě formuláře tedy, kam se odešle uživatelský vstup.
    - URL adresa je opět definována stejným způsobem jako výše.
    - Lze opět nastavit bezpečnostní omezení.

Části souboru obsahující hodnotu v #{} se dynamicky nahradí v průběhu programu. Čím bude hodnota nahrazena se definuje v dodatečných parametrech při inicializaci builder komponenty. Dodatečné parametry mají formu klíč-hodnota, takže se #{klíč} nahradí hodnotou, který je v dodatečných parametrech pod tímto klíčem. Například tedy je v parametrech pod klíčem username hodnota sa2. Potom se v XML souboru nahradí #{username} hodnotou sa2.

Tvorba komponent

Hlavní správu zajišťuje v případě AFAndroid třída AFAndroid a v případě AFWinPhone třída AFWinPhone. Jedná se o singletonové třídy, které slouží k vytváření komponent. Tyto třídy také drží již vytvořené komponenty a je tak možné je odtud získat v jakékoliv části programu pomocí

identifikátorů, které si vývojář při tvorbě komponenty určí. Ukážeme si jejich použití při tvorbě obou druhů komponent – formuláře i listu.

Tvorba formuláře v AFAndroid

```
HashMap<String, String> securityConstrains = ShowCaseUtils.getUserCredentials(getActivity());

try {
    AFForm form = AFAndroid.getInstance().getFormBuilder().initBuilder(getActivity(),
            ShowcaseConstants.PROFILE_FORM, getResources().openRawResource(
                    R.raw.connection),
            ShowcaseConstants.PROFILE_FORM_CONNECTION_KEY, securityConstrains).createComponent();
    layout.addView(form.getView());
} catch (Exception e) {
    ShowCaseUtils.showBuildingFailedDialog(getActivity(), e);
    System.err.println("BUILDING FAILED");
    e.printStackTrace();
}
```

Obrázek č. 3 – Tvorba formuláře v AFAndroid

Postup

1. Nejprve je nutné získat si instanci třídy AFAndroid, která vše spravuje
2. Potom je nutné získat nový builder pro formuláře
3. Builder je nutno nainicializovat, potřebuje:
    - Aktuální aktivitu
    - Vlastní název komponenty ve formě textového řetězce, který lze poté využít k získání komponenty v jakékoliv části programu.
    - InputStream, který čte XML soubor s nadefinovanými připojeními na zdroje na serveru
        - **InputStream je na jedno použití! Pokud vytváříme více komponent, respektive více builderů, je třeba vždy vytvořit nový InputStream.**
        - Viz. Kapitola **Definice zdrojů**
    - Identifikátor připojení, který slouží k nalezení daného konktrétního připojení v XML souboru se zdroji.
    - Dodatečné parametry definované ve formě HashMap<String, String>
        - Například username a password aktuálně připojeného uživatele
        - **Pokud není nutné definovat žádné dodatečné parametry, tak se bez tohoto argumetu inicializace builderu obejde**
4. Lze také nastavit skin pro úpravu vzhledu komponenty
    - také přes tečkovou notaci tj. initBuilder(…).setSkin(Skin skin).createComponent();
    - **Tato metoda může být vynechána, v tom případě se použije základní vzhled, který je ve frameworku předdefinován.**
    - Více o skinech viz. Kapitola **Úprava vzhledu komponenty**
5. Pomocí createComponent() se vytvoří formulář
6. Metoda getView() na formuláři získá jeho grafickou reprezentaci, kterou je možno vložit, kam bude vývojář chtít.
7. V průběhu tvorby formuláře se může stát spousta výjimek, proto je nutné obalit celý proces tvorby do try-catch bloku a s těmito výjimkami se náležitě vypořádat.
8. Tvorba formuláře **NEZAHRNUJE** vytvoření tlačítek sloužících pro odeslání formuláře atp.

Tvorba listu v AFAndroid

Tvorba listu je obdobná jako tvorba formuláře. Kód se příliš nezmění, pouze stačí vyměnit FormBuilder za ListBuilder. To lze vidět i na následujícím obrázku s ukázkou kódu.

```
try {
    AFList list = AFAndroid.getInstance().getListBuilder().initBuilder(getActivity(),
            ShowcaseConstants.MY_ABSENCES_LIST , getResources().openRawResource(R.raw.connection),
            ShowcaseConstants.MY_ABSENCES_CONNECTION_KEY, securityConstrains)
            .setSkin(new MyAbsencesListSkin(getContext())).createComponent();
    myAbsencesLayout.addView(list.getView());
} catch (Exception e) {
    ShowCaseUtils.showBuildingFailedDialog(getActivity(), e);
    e.printStackTrace();
}
```

Obrázek č. 4 – Tvorba listu ve frameworku AFAndroid

Tvorba formuláře v AFWinPhone

Tvorba formuláře je v podstatě identická s tvorbou v Androidu, obsahuje pouze malé odlišnosti.

```
Dictionary<string, string> securityConstraints = ShowcaseUtils.getUserCredentials();

try
{
    AFForm profileForm = (AFForm)AfWindowsPhone.getInstance().getFormBuilder()
        .initBuilder(ShowcaseConstants.PROFILE_FORM, "connection.xml",
            ShowcaseConstants.PROFILE_FORM_CONNECTION_KEY, securityConstraints).createComponent();

    ContentRoot.Children.Add(profileForm.getView());
}
catch (Exception ex)
{
  //zpracování výjimky
}
```

Obrázek č. 5 – Tvorba fomuláře v AFWinPhone

- Jelikož se aplikace na Windows Phone píšou v C# a ne v Javě jako na Androidu jsou zde syntaktické odlišnosti
    - Dictionary je C# ekvivalent HashMapy
- Soubor s definicemi zdrojů se nepředává jako InputStream, ale určuje se pouze název souboru (respektive cesta k němu)
- Jinak je způsob tvorby obdobný jako v případě AFAndroid

Tvorba listu v AFWinPhone

Tvorba listu je obdobná jako tvorba formuláře. Kód se příliš nezmění, pouze stačí vyměnit FormBuilder za ListBuilder. To lze vidět i na následujícím obrázku s ukázkou kódu.

```
try
{
    AFList myAbsencesList = (AFList)AfWindowsPhone.getInstance()
        .getListBuilder()
        .initBuilder(ShowcaseConstants.MY_ABSENCES_LIST, "connection.xml",
            ShowcaseConstants.MY_ABSENCES_CONNECTION_KEY, ShowcaseUtils.getUserCredentials())
        .setSkin(new MyAbsencesSkin())
        .createComponent();
    ContentRoot.Children.Add(myAbsencesList.getView());
}
catch (Exception ex)
{
    //zpracování výjimky
}
```

Obrázek č. 6 – Tvorba listu v AFWinPhone

Úprava vzhledu komponenty

Pro tento účel existují Skiny. Skin lze nastavit builderu komponenty ještě předtím, než se komponenta vytvoří. To lze vidět v následujícím obrázku.

```
try
{
    AFList myAbsencesList = (AFList)AfWindowsPhone.getInstance()
        .getListBuilder()
        .initBuilder(ShowcaseConstants.MY_ABSENCES_LIST, "connection.xml",
            ShowcaseConstants.MY_ABSENCES_CONNECTION_KEY, ShowcaseUtils.getUserCredentials())
        .setSkin(new MyAbsencesSkin())
        .createComponent();
    ContentRoot.Children.Add(myAbsencesList.getView());
}
catch (Exception ex)
{
    //zpracování výjimky
}
```

Obrázek č. 7 – Nastavení vlastního skinu komponenty

Skin vývojář vytvoří tak, že založí novou třídu a určí, že dědí ze třídy z frameworku, která se jmenuje DefaultSkin. DefaultSkin definuje velké množství metod, které nastavují vzhled komponenty z hlediska velikosti, barev textů či pozadí, typu fontu atd. Tyto metody lze přetížit a změnit tak vzhled komponenty.

Pokud uživatel skin neurčí, bude použit DefaultSkin.

Problém listu při úpravě barvy pozadí položek

V AFAndroid i v AFWinPhone nastává určitá komplikace při úpravě pozadí jednotlivých položek listu. Pozadí položek se nastaví pouze tam, kde je v položce nějaký text, nikoli na celou položku. Jako ilustraci problému použijeme ukázku z Windows Phone aplikace.

Obrázek č. 8 – Problém s pozadím v WP aplikaci

Řešení v AFAndroid

V AFAndroid k tomuto účelu vznikla v DefaultSkin metoda getTopLayoutParams(), která určuje layout nejvyšší (obalující) vrstvy komponenty. V základu je tento layout nastaven na WRAP_CONTENT, pro

šířku i výšku obalující vrstvy. Tedy obaluje pouze to, co v komponentě je a proto se i barva pozadí položky v listu vztáhne jen k oblasti, ve které se vyskytuje nějaký text.

Abychom toto chování napravili, je nutné vytvořit nový skin, ve kterém tuto metodu getTopLayoutParams() přetížíme a nastavíme šířku layoutu na MATCH_PARENT, tedy každá položka bude dědit šířku celé komponenty.

Poznamenejme ještě, že u Android aplikací se toto netýká pouze barvy pozadí, ale celé tappable části. Tedy když toto uživatel nenastaví, nebude část mimo text ani reagovat na dotyk!!

Řešení v AFWinPhone

Zde byla situace o trochu složitější a nešla řešit pomocí skinů. Na rozdíl od Androidu je část mimo text dotyková a problém se skutečně týká jen barvy pozadí. Lze to vyřešit tak, že do App.xaml dovnitř uzlu <Application>, který by měl být v kořenovém adresáři souboru, vývojář dodá následující kód.

```
<Applicaiton …>
<Application.Resources>
        <Style TargetType="ListViewItem" x:Key="ItemContainerStretchStyle">
            <Setter Property="HorizontalContentAlignment" Value="Stretch">
            </Setter>
        </Style>
</Application.Resources>
</Application>
```

Funkčnost řešení zobrazuje následující screenshot z Windows Phone aplikace

Obrázek č. 9 – Ukázka vyřešení problému s pozadím ve WP aplikaci.

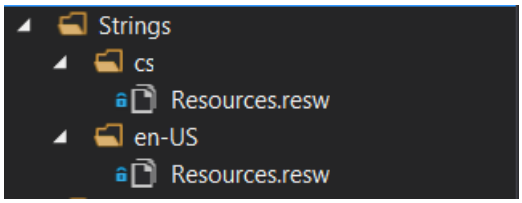
Lokalizace textů

Ze serveru mohou přicházet i lokalizační texty, tedy texty, které mají být přeloženy. K tomutu účelu byla v obou frameworcích vytvořena třída **Localization,** která umí texty přeložit i změnit jazyk za běhu aplikace. Mimo texty přícházející ze serveru, lze třídu využít i pro překlad jakýchkoliv jiných textů v aplikaci.

Nutnou podmínkou pouze je, aby vývojář vytvořil překlady a to na správném místě.

Windows phone

WP framework předpokládá existenci lokalizačních souborů ve formátu Resources.resw, které jsou umístěny ve složkách pojmenovaných podle zkratky jazyka. To vše je umístěno ve složce Strings. Na velikosti písmen v názvech složek nezáleží.

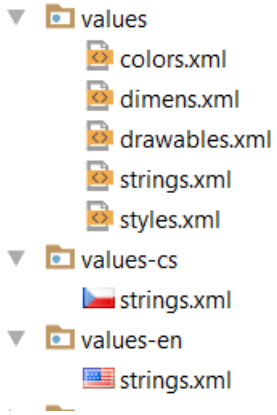


Obrázek č. 10 – Struktura lokalizačních souborů ve WP aplikaci

Možná může zmást název anglické složky se zdrojem předkladů. U anglického jazyka, na rozdíl od češtiny, je možné specifikovat jeho původ. En-US tedy specifikuje americkou variantu anglického jazyka.

Android

Android framework je trochu náročnější. Za prvé předpokládá, že překlady budou vytvořeny v blaičku **res** se zdroji, kde lze nalézt složku **values** a v ní **strings.xml**. Zde je vhodné nadefinovat texty v nějakém defaultním jazyce, neboť se použijí, pokud nebudou k dispozici předklady přesně pro daný jazyk. Pro další jazyky je nutné vytvořit kopii složky values (ve které stačí mít jen strings.xml) a přidat kopiím k názvu pomlčku a zkratku jazyka, kterou představují. Pro češtinu to bude např. **values-cs**. Strukturu v projektu, lze vidět na následujícím obrázku.



Obrázek č. 11 – Struktura lokalizačních souborů v Android aplikaci

Za druhé je nutné Android frameworku říct, že chceme jeho lokalizační třídu použít v naší tvořené aplikaci a nastavit ji **kontext** a **defaultní balíček naší aplikace**. To lze udělat kdekoliv při startu aplikace, například v MainActivity při startu onCreate metody.

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

    //set localization
    Localization.setContext(getThisActivity());
    Localization.setPathToStrings("cz.cvut.fel.matyapav.showcase");
```

Obrázek č. 12 – Nastavení lokalizační třídy v Android aplikaci

Práce s komponentou

S komponentou se dá dále pracovat. Formuláře lze například odeslat na server nebo lze propojit list s formulářem.

Získání komponenty

K tomu je nutné umět získat již vytvořenou komponentu z jakékoliv části aplikace a ne pouze tam, kde jsme ji vytvořili. Vytvořené komponenty jsou ukládány pod identifikátorem, který si vývojář sám při inicializaci builderu komponenty nadefinuje, do HashMapy (Android) nebo Dictionary (Windows Phone) vytvořených komponent. Ta lze referencovat pomocí v případě Windows Phone třídy AFWinPhone a v případě Androidu třídy AFAndroid, které jsme již potřebovali při vytváření komponent. Získat komponentu lze následujícím kouskem kódu.

```
Android verze
AFForm form =
(AFForm)AFAndroid.getInstance().getCreateComponentByName("název komponenty, určený při vytváření");

Windows Phone verze
AFForm form = (AFForm) AfWindowsPhone.getInstance().getCreatedComponentByName("název komponenty, určený při vytváření");
```

Stejně tak to funguje i v případě listu.

Je doporučené v programu kontrolovat po získání komponenty, zda není **null** v případě, že by komponenta ve vytvořených komponentách z nějakého důvodu neexistovala

Odeslání formuláře na server

Tuto akci je samozřejmě nutno nějak spouštět, proto si vývojář musí nejprve vytvořit tlačítko. Na to navěsí poslouchač události, ve kterém si získá formulář. Tento formulář disponuje metodou **sendData()**, kterou lze použít pro odeslání dat na server. Metoda vrací **boolean**, aby vývojář mohl kontrolovat, zda bylo odeslání úspěšné a popřípadě provést nějaké akce. Také metoda vyhazuje v určitých případech výjimku.

- SendData() vrací false, pokud se nepovede validovat uživatelský vstup nebo se nevygenerují z formuláře žádná data.
- SendData() vyhodí výjimku, pokud se stane něco v procesu odeslání, tedy nepůjde například kontaktovat zdroj, na který by se formulář měl odeslat atp.

```
private View.OnClickListener onLoginButtonClick = (v) -> {
        AFForm form = (AFForm) AFAndroid.getInstance().getCreatedComponents().get(ShowcaseConstants.LOGIN_FORM);
        if (form != null) {
            try {
                if(form.sendData()) {
                    doLogin(form);
                }
            } catch (Exception e) {
                //login failed
            }
        }
};
```

Obrázek č. 12 – Ukázka odeslání přihlašovacího formuláře v Android aplikaci

```
if (AfWindowsPhone.getInstance().getCreatedComponents().ContainsKey(ShowcaseConstants.LOGIN_FORM))
{
    var form = (AFForm)AfWindowsPhone.getInstance().getCreatedComponents()[ShowcaseConstants.LOGIN_FORM];
    try
    {
        if(await form.sendData())
        {
            doLogin(form);
        }
    }
    catch (Exception ex)
    {
        //login failed
    }

}
```

Obrázek č. 13 – Ukázka odeslání přihlašovacího formuláře ve Windows Phone aplikaci

Zde si lze všimnout, že je u sendData() navíc **await**. Metoda sendData() je totiž asynchronní a await umožňuje počkat na její výsledek, což je žádoucí.

Další operace s formulářem

Dalšími operacemi s formulářem jsou:

- **Resetování formuláře**
  - form.reset()
  - pokud je formulář předvyplněn nějakými daty a uživatel provede změny, resetuje tyto změny na aktuální data
- **Vyčištění formuláře**
  - form.clear()
  - vyčistí formulář, respektive nastaví všechna pole na prázdná

Propojení formuláře a listu

List lze propojit s formulářem v tom smyslu, že pokud například klikneme na položku v listu, naplní se příslušný formulář (který vznikl ze stejné definice jenom výměnou builderů) daty, které v položce jsou.

Pro Android aplikace toho lze dosáhnout následujícím kódem.

```
//connect list and form
final AFList countryList = (AFList) AFAndroid.getInstance().getCreatedComponents().get(ShowcaseConstants.COUNTRY_LIST);
final AFForm countryForm = (AFForm) AFAndroid.getInstance().getCreatedComponents().get(ShowcaseConstants.COUNTRY_FORM);

if(countryList != null && countryForm != null) {
    countryList.getListView().setOnItemClickListener((parent, view, position, id) -> {
            countryForm.insertData(countryList.getDataFromItemOnPosition(position));
    });
}
```

Obrázek č. 14 – Propojení formuláře a listu v Android aplikacích

- Je potřeba si obě komponenty získat z vytvořených komponent
- V tomto specifickém případě jsme naplnění formuláře navázali na událost kliknutí na položku listu
- Formulář disponuje metodou insertData, která umí vložit data z JSON stringu
- Tento JSON string lze z položky v listu vytvořit metodou getDataFromItemOnPosition s pozicí v listu v argumentu.

Pro Windows Phone aplikace je to opět obdobné, kód se liší opět jen v maličkostech.

```
private void OnItemClick(object sender, ItemClickEventArgs itemClickEventArgs)
{
    var countryForm =
        (AFForm)AfWindowsPhone.getInstance().getCreatedComponentByName(ShowcaseConstants.COUNTRY_FORM);
    var countryList =
        (AFList)AfWindowsPhone.getInstance().getCreatedComponentByName(ShowcaseConstants.COUNTRY_LIST);
    if(countryForm != null && countryList != null) {
        var position = countryList.getListView().Items.IndexOf(itemClickEventArgs.ClickedItem);
        countryForm.insertData(countryList.getDataFromItemOnPosition(position));
    }
}
```

Obrázek č. 15 – Propojení formuláře a listu ve Windows Phone aplikacích

Změna orientace zařízení v Android aplikacích

Toto je důležitá sekce, neboť výrazně ovlivňuje fungování Android aplikací. Android vývojáři jistě ví, že se při změně orientace obsah obrazovky znovu překresluje, respektive aktivita se ukončí a znovu spustí. V případě, že ale vytváříme dynamicky GUI formuláře pomocí Android frameworku, provede se při změně orientace opět celý proces jeho stavění. Data, která uživatel vyplnil před změnou orientace zařízení, se tedy ztratí. Toto se dá řešit mnoha způsoby, jedním z nich je, že se přepíše metoda onConfigurationChanged(), kde se obsah polí uloží do startActivity bundlu a při opětovném načtení aktivity se zase odtud data do polí nahrají. To je však ta složitější možnost.

Android aplikaci stačí jednoduše říct, že chceme změnu orientace řešit sami a respektive zakážeme všechny akce kromě samotného otočení v souboru **AndroidManifest.xml.** Zde stačí příslušné aktivitě, které chceme znovunačítání zastavit dodat parametr **android:configChanges="orientation"** jako na následujícím obrázku.

```
<activity
    android:name=".MainActivity"
    android:label="AFAndroid"
    android:theme="@style/AppTheme.NoActionBar"
    android:configChanges="orientation|screenSize"
    >
```

Obrázek č. 16 – Nastavení změny orientace

Zde bylo navíc přidána i změna velikosti displeje (respective rozlišení).