

ROZWIĄZANIA BIOMETRYCZNE W IDENTYFIKACJI UŻYTKOWNIKÓW · REACTIVEX

Index: 285358

Magazyn programistów i liderów zespołów IT

**programista**

1/2017 (56)

Cena 23,90 zł (w tym VAT 5%)

luty/marzec

# ANDROID NA STERYDACH

MINIMUM KODU, MAKSIMUM EFEKTYWNOŚCI

Poznaj Unraco z THE COGWORLD! SZCZEGÓŁY W ŚRÓDKU

ISSN 2084-9400

ZRÓDŁO: FOTOLIA

9 772084 940701

NGINX – (NIE)ZWYKŁY SERWER WWW

Jak przygotować skalowalny i wydajny serwis WWW w oparciu o otwarte narzędzia

SZABLONY W ANDROID STUDIO

Jak oszczędzić czas, stosując automatyczne generowanie fragmentów kodu źródłowego

ZARZĄDZANIE ENERGIĄ W ARMV8

Oprogramowanie mówi maszynie "Dobranoc!". Dwa światy nowych ARMów

Join our team of Makers!

[career.cybercom.com](http://career.cybercom.com)

## Change tomorrow with us

[www.cybercom.pl](http://www.cybercom.pl)

Cybercom Poland Sp. z o.o.  
ul. Hrubieszowska 2, 01-209 Warszawa / ul. Dowborczyków 25, 90-019 Łódź

CYBERCOM GROUP

**Aby żyło nam się łatwiej**

Gdy zaczynałem swoją przygodę z komputerami prawie trzydzieści lat temu, praktycznie każda linijka pisanej kodu tworzyła się ręcznie. Nawet w najbardziej wymyślnych marzeń nigdy nie przychodziła do głowy myśl o możliwości automatycznego generowania kodu. Dzisiaj część monotonicznej pracy programisty można zautomatyzować, duże fragmenty kodu mogą zostać wygenerowane praktycznie bez naszego udziału. Przykład wykorzystania możliwości „łatwiej” sobą życia we współczesnych środowiskach programistycznych można przeczytać w artykule „Szablony w Android Studio”.

W ciągu tych lat wszystko uległo ogromnym zmianom, wszelkobecny podczas przypisywania praktycznie wszystkiego odcisnął znaczące pieczę na naszej branży. Część popularnych narzędzi funkcjonuje w sieci jako serwisy WWW, gdzie wydajność, odporność na awarie oraz skalowalność i łatwość zarządzania są decydującymi wymaganiami. Przykładem rozwiązań tych problemów opisano w artykule „Nginx – (nie)wykrywalny serwer WWW”.

Jednak wzrost wydajności powoduje problemy związane z oszczędnością energii. Praktycznie każdy producent nowoczesnych CPU implementuje w nich technologie oszczędzające, np. wyłączające własne rozwiązania, często bardziej efektywne. Przykładem takich rozwiązań są „dwa światy” w ARMv8, o czym można przeczytać w artykule „Porządkuj zarządzanie energią w ARMv8”.

Ten ciągły podział do przodu z rozwojem technologii doprowadził do tego, że to, o którym mówimy w dawnych filmach opowiadających o przyszłości, staje się dla nas rzeczywistość, autoryzacja przy pomocy oka, skanowanie dłoni i nie tylko – to nadal brzmiące futurystyczne możliwości, jednak dostępne już dziś, o czym możemy przekonać, zapoznając się z artykułem „Rozwiązania biometryczne w identyfikacji użytkowników”.

Mam nadzieję, że każdy znajdzie coś interesującego dla siebie w tym wydaniu. Zyczymy udanej lektury!

Mariusz „maryush” Witkowski

**BIBLIOTEKI I NARZĘDZIA**

Szablony w Android Studio Wojciech Sura 4

ReactiveX – programowanie reaktywne w niemalże każdym języku i systemie Marcin Moskwa 10

**PROGRAMOWANIE URZĄDZEŃ MOBILNYCH**

Azure Mobile Apps i platforma Xamarin Daniel Kryzickowski 16

Android na sterwach: Native Activity w parze z OpenGL ES 3.0 Mateusz Semegen 20

**PROGRAMOWANIE ROZWIĄZAŃ SERWEROWYCH**

Nginx – (nie)wykrywalny serwer WWW i nie tylko Maciej Naboński 28

**PROGRAMOWANIE SYSTEMOWE**

Pora spać! Zarządzanie energią w ARMv8 Marcin Wójcik 36

**PROGRAMOWANIE APLIKACJI WEBOWYCH**

Własny blog z Umbraco w 1,5 godziny Marcin Zaikowski 42

**BEZPIECZEŃSTWO**

Nietypowe metody wykorzystywane w atakach phishingowych Artur Czyż 48

**TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ**

Zastosowanie grafów w projektowaniu przypadków testowych. Segmentacja grafów Marek Żukowicz 56

**ZARZĄDZANIE PROJEKTAMI**

Jak zmienia się Scrum – ewolucja Scrum Guides Michał Cicha 62

**FELIETON**

Rozwiązywanie biometryczne w identyfikacji użytkowników Maciej Szymkowiak 66

**KLUB LIDERA IT**

Idealne dopasowanie Grzegorz Olsender 72

**PLANETA IT**

angielski dla programistów. Lekcja 1 Łukasz Piwko 76

Magazyn Programista wydawany jest przez Dom Wydawniczy Anna Adamczyk

Wydawca/Redaktor naczelny: Anna Adamczyk (anna.adamczyk@programistamag.pl).

Redaktor techniczny: Michał Leszczyński (michal.leszczyński@programistamag.pl).

Korekta: Tomasz Łopatarski. Kierownik produkcji: Paweł DTR: Paweł.

Dział reklamy: Małgorzata Bartylewicz, Dawid Kaliszewski, Marek Saverwijn, Lukasz Mazur, Lukasz Topuzanski, Jacek Matulewski, Slawomir Sobkowicz, Dawid Borycki, Gwylia Colwind, Bartosz Chrabski, Rafal Kocisz, Michał Sajdak, Michał Bentkiewski, Mariusz „maryush” Witkowski, Paweł „krzaoQ” Zakrzewski.

Współpraca: Michał Bartylewicz, Mariusz Sierackiewicz, Dawid Kaliszewski, Marek Saverwijn, Lukasz Mazur, Lukasz Topuzanski, Jacek Matulewski, Slawomir Sobkowicz, Dawid Borycki, Gwylia Colwind, Bartosz Chrabski, Rafal Kocisz, Michał Sajdak, Michał Bentkiewski, Mariusz „maryush” Witkowski, Paweł „krzaoQ” Zakrzewski.

Adres wydawcy: Dzierżoniowa 4/47, 02-776 Warszawa.

Druk: Drukarnia Edit u. Dworcowka 2, 05-682 Włocławek, Nakład: 4500 egz.

## reklama

## Nota prawna

Redakcja zastrzega sobie prawo do skrótu i opracowania tekstów oraz do zmiany planów wydawniczych, tj. zmian w zapowiadanych tematach artykułów i terminach publikacji, a także nakładów i objętości czasopisma.

O ile nie zaznaczono inaczej, wszelkie prawa do materiałów i znaków Oznakiem autorskim zamieszczonych na łamach magazynu Programista są zastrzeżone. Kopiowanie i rozpowszechnianie ich bez zezwolenia jest zabronione.

Redakcja magazynu Programista nie ponosi odpowiedzialności za szkody bezpośrednie i pośrednie, jak również za inne straty i wydatki poniesione w związku z wykorzystaniem informacji prezentowanych na łamach magazynu Programista.

**ZOSTAŃ KODOŁAMACZEM**

**Zdobądź zawód w 7 tygodni**

Dowiedz się więcej na [odołamacz.pl](http://odołamacz.pl)

Organizator: **SAGES**

**Junior Front-end Developer**

**Junior Data Scientist**

**Junior Java Developer**

PIERWSZY  
BOOTCAMP  
DATA SCIENCE  
W POLSCE

Dowiedz się więcej na [odołamacz.pl](http://odołamacz.pl)

Organizator: **SAGES**

**BIBLIOTEKI I NARZĘDZIA****Szablony w Android Studio**

Android Studio zawiera wiele szablonów aktywności, fragmentów i innych gotowych kawałków kodu. Umożliwiają one szybkie generowanie zamkniętych kawałków projektu – oszczędzając w ten sposób trochę czasu programisty. Spróbujmy napisać taki szablon samodzielnie.

## Szablon samodzielny dla aplikacji Android

W poprzednich numerach magazynu opisalem, w jaki sposób projektować aplikację dla Androida w sposób (prawie) zgodny ze wzorcem MVVM. Dowiedzieliśmy się też, jak skorzystać z biblioteki Data Binding Library, by uprościć prezentowanie w widoku danych publikowanych przez.viewmodel. O ile technika ta pozwala znacznie ograniczyć ilość glue-code, to można jednak jednorazowo dodać pewien wzorzec, który często się tam powtarza:

- » Struktura pliku layoutu musi być odpowiednia (mowa o tagach layout, data, variable itp.).
- » Kod aktywności musi zainstancować.viewmodel i przypisać go do zmiennej,
- » Musimy przygotować klasę.viewmodel, który będzie współpracował z aktywnością,
- » Jeżeli przewidujemy interakcję pomiędzy.viewmodelem i aktywnością, aktywność musi zaimplementować interfejs `T<nazwa>ActivityAccess`, który z kolei trzeba przekazać.viewmodelu przez parametr konstruktora i tam też zapisać,
- » Do `AndroidManifest.xml` musimy dodać odpowiedni wpis.

Nie jest to wprawdzie zbyt dużo, ale czynności te musimy wykonać dla każdej aktywności, a pominięcie którego punktu lub drobna pomyłka skutkuje irytującym błędami komplikacji (lub wykonania).

rzystywan w pluginie ADT dla Eclipse, a bazowany na systemie o nazwie FreeMarker. I o ile ten ostatni jest dosyć dobrze udokumentowany, to niestety dokumentacji mechanizmu szablonów w Android Studio trzeba się nieźle poszukać, a i tak nie jest ona kompletna – twórcy tego środowiska najprawdopodobniej nie przewidywali, że użytkownicy będą pisali własne szablony, choć samo środowisko takie działania wspiera.

**STRUKTURA SZABLONU**

Przygotowanie własnego szablonu będzie wymagało niewielkiej ilości reverse-engineeringu. Po pierwsze, musimy zlokalizować istniejące szablony – i na szczęście okazuje się, że są one dostępne w postaci zwykłych plików tekstowych w katalogu `C:\Program Files\Android\Android Studio\plugins\android\lib\templates`. Co więcej, Android Studio przeszukuje te miejsca za wszelką cenę, więc wystarczy utworzyć tam nowy katalog z odpowiednią strukturą plików; by nasz szablon stał się dostępny do użycia.

Struktura plików w katalogu naszego szablonu będzie wyglądała następująco:

```
MMActivity
root
res
layout
activity.xml.ftl
values
manifest_strings.xml.ftl
src
app_package
Activity.java.ftl
ActivityAccess.java.ftl
ViewModel.java.ftl
AndroidManifest.xml.ftl
globals.xml.ftl
recipe.xml.ftl
template.xml
template_mvvm_activity.png
```

Katalog `root` reprezentuje główny katalog projektu, w którym uruchomimy plugin – jest wymogiem szablonu, by tak właśnie się nazywał. Katalogi `res` i `src` powinniśmy skojarzyć już z ich projektowymi odpowiednikami, natomiast `app_package` zastępuje nazwę głównego pakietu projektu.

Zainteresujmy się teraz, co znajduje się w każdym z plików.

**TEMPLATE.XML**

Plik `template.xml` jest podstawowym (lub obowiązkowym) plikiem, który opisuje szablon. W naszym przypadku będzie on wyglądał następująco:



Rysunek 1. Szablony wbudowane w środowisko

Okazuje się, że nie jest to ani rozwiązanie specyficzne dla Android Studio, ani dla IntelliJ, ale jest to ten sam silnik, który był wyko-

## Szablony w Android Studio

Android Studio zawiera wiele szablonów aktywności, fragmentów i innych gotowych kawałków kodu. Umożliwiają one szybkie generowanie zamkniętych kawałków projektu – oszczędzając w ten sposób trochę czasu programisty. Spróbujmy napisać taki szablon samodzielnie.

W poprzednich numerach magazynu opisalem, w jaki sposób zaprojektować aplikację dla Androïda w sposób (prawie) zgodny ze wzorcem MVVM. Dowiedzieliśmy się też, jak skorzystać z biblioteki Data Binding Library, by uprosić prezentowanie w widoku danych publikowanych przez.viewmodel. O ile technika pozwala znacząco ograniczyć ilość glue-code, to można jednak dostrzec pewien wzorzec, który często się tam powtarza:

- » Struktura pliku layoutu musi być odpowiednia (mowa o tagach layout, data, variable itp.),
- » Kod aktywności musi zainstancjonować.viewmodel i przypisać go do zmiennej,
- » Musimy przygotować klasę.viewmodel, który będzie współpracował z aktywnością,
- » Jeżeli przewidujemy interakcję pomiędzy.viewmodel i aktywnością, aktywność musi zaimplementować interfejs `I<nazwa>ActivityAccess`, który w kolei trzeba przekazać.viewmodelowi przez parametr konstruktora i tam też zapisać,
- » Do `AndroidManifest.xml` musimy dodać odpowiedni wpis.

Nie jest to wprawdzie zbyt dużo, ale czynności te musimy wykonać dla każdej aktywności, a pominięcie któregoś punktu lub drobna pomyłka skutkuje rytmującym błędami komplikacji (lub wykonania).

### MECHANIZM SZABLOŃÓW

Android Studio ma wbudowany mechanizm szablonów, dostarczający gotowe aktywności, fragmenty itp. – jak widać na Rysunku 1.



Rysunek 1. Szablony wbudowane w środowisko

Okazuje się, że nie jest to ani rozwiązywanie specyficzne dla Android Studio, ani dla IntelliJ, ale jest to ten sam silnik, który był wyko-

rzystywany w pluginie ADT dla Eclipse, a bazowany na systemie o nazwie FreeMarker. I o ile ten ostatni jest dosyć dobrze udokumentowany, to niestety dokumentacji mechanizmu szablonów w Android Studio trzeba się nieźle poszukać, a i tak nie jest ona kompletna – twórcy tego środowiska najprawdopodobniej nie przewidywali, że użytkownicy będą pisali własne szablony, choć samo środowisko takie działania wspiera.

### STRUKTURA SZABLONU

Przygotowanie własnego szablonu będzie wymagało niewielkiej ilości reverse-engineeringu. Po pierwsze, musimy zlokalizować istniejące szablony – i na szczęście okazuje się, że są one dostępne w postaci zwykłych plików tekstowych w katalogu `C:\Program Files\Android\Android Studio\plugins\android\lib\templates`. Co więcej, Android Studio przeszukuje to miejsce zawsze podczas uruchamiania, więc wystarczy utworzyć tam nowy katalog z odpowiednią strukturą plików, by nasz szablon stał się dostępny do użycia.

Struktura plików w katalogu naszego szablonu będzie wyglądała następująco:

```
MVVMActivity
root
+-- res
|   +-- layout
|   |   +-- activity.xml.ftl
|   +-- values
|       +-- manifest_strings.xml.ftl
+-- src
    +-- app_package
        +-- Activity.java.ftl
        +-- ActivityAccess.java.ftl
        +-- ViewModel.java.ftl
        +-- AndroidManifest.xml.ftl
    +-- globales.xml.ftl
    +-- recipe.xml.ftl
    +-- template.xml
    +-- template_mvvm_activity.png
```

Katalog `root` reprezentuje główny katalog projektu, w którym uruchomimy plugin – jest wymogiem szablonu, by tak właśnie się nazywał. Katalogi `res` i `src` powinniśmy skojarzyć już z ich językowymi odpowiednikami, natomiast `app_package` zastępuje nazwę głównego pakietu projektu.

Zainteresujmy się teraz, co znajduje się w każdym z plików.

### TEMPLATE.XML

Plik `template.xml` jest podstawowym (i obowiązkowym) plikiem, który opisuje szablon. W naszym przypadku będzie on wyglądał następująco:

# Zintegruj własną aplikację z SMS API

Skorzystaj z gotowych bibliotek w językach:



Załóż konto firmowe z kodem polecenia i odbierz pakiet SMS-ów na przetestowanie naszych usług:

FORMULARZ REJESTRACYJNY:

[www.smsapi.pl/rejestracja](http://www.smsapi.pl/rejestracja)

KOD POLECENIA:

PR56

500 SMS-ÓW

An evaluation version of [novaPDF](#) was used to create this PDF file.

**Listing 1. Plik template.xml**

```
<template format="4"
revision="1"
name="MVM Activity"
description="Creates MVM activity">

<category value="Other"/>

<parameter id="activityName"
name="Activity Name"
type="string"
constraints="class|unique|nonempty"
default="New"
help="Name of the new activity, excluding the word &quot;Activity&quot;"/>

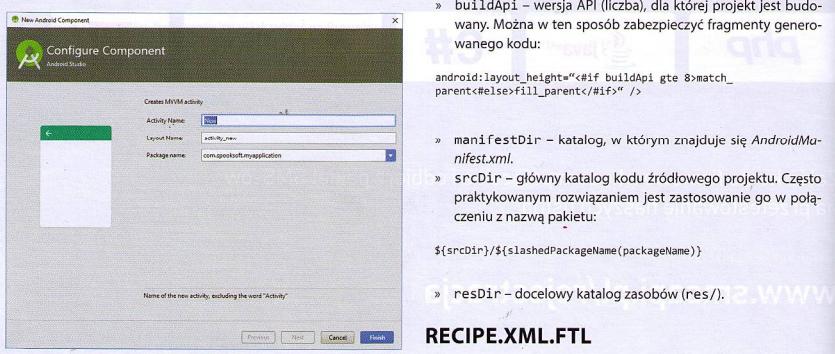
<parameter id="layoutName"
name="Layout Name"
type="string"
constraints="layout|unique|nonempty"
suggest="#{activityToLayout(activityName)}"
default="activity_main"
help="The name of the layout to create for the activity" />

<parameter id="packageName"
name="Package name"
type="string"
constraints="package"
default="com.spooksoft.myapp" />

<thumbnails>
<!-- default thumbnail is required -->
<thumb>template_mvm_activity.png</thumb>
</thumbnails>

<globals file="globals.xml.ftl" />
<execute file="recipe.xml.ftl" />
</template>
```

Wewnątrz znacznika `template` definiujemy nazwę i opis szablonu; możemy również określić kategorię, w której się on znajdzie. Następnie pojawia się seria parametrów: każdy z nich pozwala na zdefiniowanie zmiennej, której możemy potem użyć wewnątrz szablonu. Android Studio każdy z tych parametrów przekształci na element interfejsu użytkownika – edytor wygenerowany z powyższego pliku możemy zobaczyć na Rysunku 2.



Rysunek 2. Edytor szablonu

Po parametrach zdefiniowane są odnośniki do wartości globalnych oraz do pliku „recepty”, który opisuje działanie szablonu. Skupmy się na razie na tym pierwszym.

**GLOBSLS.XML.FTL**

Plik ten wygląda następująco:

**Listing 2. Plik globals.xml.ftl**

```
<?xml version="1.0"?>
<globals>
<global id="manifestOut" value="${manifestDir}" />
<global id="resOut" value="${resDir}" />
<global id="srcOut" value="${srcDir}/${slashedPackageName}(packageName)" />
</globals>
```

Każdy z wpisów wewnątrz tagu `globals` pozwala zdefiniować pewną wartość, która będzie dostępna podczas generowania każdego z plików szablonu. W ten sposób przygotowujemy trzy stałe: ścieżkę do katalogu z manifestem, do katalogu zasobów oraz do katalogu zawierającego kod źródłowy aplikacji.

Mogliśmy skorzystać również z wartości dostarczanych przez środowisko (zarówno tutaj, jak i w innych plikach szablonu), wśród nich:

- » `packageName` – nazwa pakietu projektu, np. `com.spooksoft.myapp`
- » `applicationPackage` – zostanie ustawiona na pakiet aplikacji, jeżeli docelowy pakiet szablonu będzie inny. W przeciwnym wypadku ta wartość będzie pusta.
- » `isNewProject` – wartość boolean, która określa, czy szablon jest wywoływany jako część procesu tworzenia nowego projektu.
- » `minApi` – minimalna wersja API, którą wspiera projekt (może być w formie ciągu znaków).
- » `minApiLevel` – minimalna wersja API w postaci liczby. Można na niej używać do generowania kodu bazowanego na poziomie API, na przykład:

```
int resourceId = android.R.layout.simple_list_item_<if
minApiLevel gte 11>activated_</if>1;

» buildApi – wersja API (liczba), dla której projekt jest budowany. Można w ten sposób zabezpieczyć fragmenty generowanego kodu:

android:layout_height="#{if buildApi gte 8>match_
parent:#else;fill_parent#if}" />

» manifestDir – katalog, w którym znajduje się AndroidManifest.xml.
» srcDir – główny katalog kodu źródłowego projektu. Często praktykowanym rozwiązaniem jest zastosowanie go w połączeniu z nazwą pakietu:

${srcDir}/${slashedPackageName}(packageName)

» resDir – docelowy katalog zasobów (res/).
```

**RECIPE.XML.FTL**

Plik ten zawiera instrukcje, które będą wykonywane podczas generowania kodu za pomocą wybranego szablonu. Nazwa pliku jest dowolna, ale według konwencji powinien ona nazywać właśnie `recipe.xml.ftl`. Miejmy na uwadze, że możemy tu korzystać ze stałych zdefiniowanych w pliku `globals.xml.ftl`.

**Listing 3. Plik recipe.xml.ftl**

```
<variable name="viewModel" type="${packageName}.viewmodels.${activityName}ActivityViewModel" />
</data>
<Linearlayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
<include
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="10dp"
    android:layout_weight="1"
    android:background="@color/white"
    android:padding="10dp"
    android:scrollbarStyle="outsideInset"
    android:scrollbars="vertical"
    android:theme="@style/Theme.AppCompat.NoActionBar" />
</Linearlayout>
```

**SZABLONY W ANDROID STUDIO**

Ponieważ częściej korzystam z `LinearLayout` niż `RelativeLayout`, zdecydowałem się również zmienić główny tag layoutu. Można go oczywiście ustawić według własnych preferencji.

**MANIFEST\_STRINGS.XML.FTL**

Zgodnie z instrukcją plik ten zostanie scalony z plikiem `strings.xml`, zawierającym tekstowe zasoby aplikacji. Zauważmy, że do jednego pliku możemy scalić kilka innych. Dla przykładu stałe związane z manifestem i aktywnością możemy przechować w dwóch osobnych plikach szablonu, scalając je potem oba ze `strings.xml` – możemy w ten sposób zachować w plikach szablonu większy porządek.

**Listing 6. Plik manifest\_strings.xml**

```
<resources>
<string name="title_${activityToLayout(activityName)}"${{escapeXmlString(activityName)}}></string>
</resources>
```

**ACTIVITY.JAVA.FTL**

Oprócz tego dostępna jest akcja `copy`, która po prostu kopiuje plik z jednej lokalizacji do drugiej (bez przetwarzania go), oraz `open`, która instruuje środowisko, aby otworzyło zadany plik po zakończeniu generowania szablonu.

**ANDROIDMANIFEST.XML.FTL**

Tutaj definiujemy fragment kodu XML, który zostanie scalony z plikiem manifestu aplikacji.

**Listing 4. Plik AndroidManifest.xml.ftl**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android" >
<application>
<activity android:name=".activities.${activityName}Activity"
        android:label="@{string:title_${activityToLayout(activityName)}}"
        </activity>
</application>
```

**ACTIVITY.XML.FTL**

Znajdujący się w podkatalogu layout plik określa, jak będzie wyglądał layout aktywności. Możemy tu od razu zdefiniować zmienną `viewModel` oraz zamknąć wszystko w tagu layout, pamiętając o zdefiniowaniu odpowiednich przestrzeni nazw XML – oszczędzi to nam czasu w przyszłości.

**Listing 5. Plik activity.xml.ftl**

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:bind="http://schemas.android.com/apk/res-auto"
        tools:context="${packageName}.activities.${activityName}Activity">
<data>
```

variable name="viewModel" type="\${packageName}.viewmodels.\${activityName}ActivityViewModel" />
</data>
<Linearlayout
 android:layout\_width="match\_parent"
 android:layout\_height="match\_parent"
 android:orientation="vertical">
</Linearlayout>

Ponieważ częściej korzystam z `LinearLayout` niż `RelativeLayout`, zdecydowałem się również zmienić główny tag layoutu. Można go oczywiście ustawić według własnych preferencji.

**MANIFEST\_STRINGS.XML.FTL**

Zgodnie z instrukcją plik ten zostanie scalony z plikiem `strings.xml`, zawierającym tekstowe zasoby aplikacji. Zauważmy, że do jednego pliku możemy scalić kilka innych. Dla przykładu stałe związane z manifestem i aktywnością możemy przechować w dwóch osobnych plikach szablonu, scalając je potem oba ze `strings.xml` – możemy w ten sposób zachować w plikach szablonu większy porządek.

**Listing 6. Plik manifest\_strings.xml**

```
<resources>
<string name="title_${activityToLayout(activityName)}"${{escapeXmlString(activityName)}}></string>
</resources>
```

**ACTIVITY.JAVA.FTL**

Oprócz tego dostępny jest akcja `copy`, która po prostu kopiuje plik z jednej lokalizacji do drugiej (bez przetwarzania go), oraz `open`, która instruuje środowisko, aby otworzyło zadany plik po zakończeniu generowania szablonu.

**ANDROIDMANIFEST.XML.FTL**

Tutaj opisujemy kod aktywności, a w tym przywiązanie layoutu do bindingu, zainstancjonowanie viewmodelu oraz przypisanie go do bindingu.

**Listing 7. Plik Activity.java.ftl**

```
package ${packageName}.activities;

import android.content.Context;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.appcompat.app.AppCompatActivity;
import ${packageName}.R;
import ${packageName}.dataBinding.Activity${activityName}Binding;
import ${packageName}.viewmodels.${activityName}ActivityViewModel;
import ${packageName}.viewmodels.interfaces.I${activityName}ActivityAccess;
public class ${activityName}Activity extends AppCompatActivity
implements I${activityName}ActivityAccess {
    // Private fields
    private Activity${activityName}Binding binding;
    private ${activityName}ActivityViewModel viewModel;
    // Protected methods
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = DataBindingUtil.setContentView(this,
R.layout.${layoutName});
        viewModel = new ${activityName}ActivityViewModel(this);
        binding.setViewModel(viewModel);
    }
}
```

## BIBLIOTEKI I NARZĘDZIA

### ACTIVITYACCESS.JAVA.FTL

Teraz możemy zdefiniować pusty interfejs ActivityAccess, za pomocą którego będziemy mogli z poziomu viewmodelu zlecić coś aktywności.

Listing 8. Plik ActivityAccess.java.ftl

```
package ${packageName}.viewmodels.interfaces;
public interface IS{${activityName}ActivityAccess {
```

↳ nowy plik pojawi się w katalogu szablonów

### VIEWMODEL.JAVA.FTL

Na koniec pozostało już tylko kod viewmodelu. Pamiętamy o tym, żeby odziedziczyć go po BaseObservable – w ten sposób będziemy mogli łatwo korzystać z klas ObservableField<T> do automatycznego notyfikowania bindingu o zmianach.

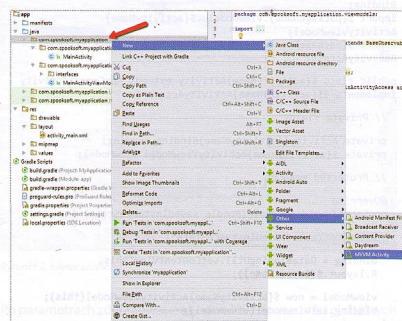
↳ nowy plik pojawi się w katalogu szablonów

Listing 9. Plik ViewModel.java.ftl

```
package ${packageName}.viewmodels;
import android.databinding.BaseObservable; import android.databinding.ObservableField;
import ${packageName}.viewmodels.interfaces.IS${activityName}ActivityAccess;
public class ${activityName}ActivityViewModel extends BaseObservable {
    // Private fields
    private IS${activityName}ActivityAccess access;
    // Private methods
    public ${activityName}ActivityViewModel(IS${activityName}ActivityAccess access) {
        this.access = access;
    }
}
```

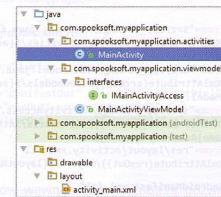
### TESTUJEMY

Tak przygotowany szablon umieszczamy w podkatalogu Other katalogu szablonów i wywołujemy na głównym pakiecie aplikacji (Rysunek 3). Można również zdefiniować własny pakiet (w oknie edycyjnym), ale szablon utworzy w nim pakiet activities\$viewmodels.



Rysunek 3. Uruchamiamy szablon

Po wypełnieniu pól Android Studio automatycznie wygeneruje szablon (Rysunek 4).



Rysunek 4. Efekt działania szablonu

### WIĘCEJ MOŻLIWOŚCI

FreeMarker oferuje pewien zbiór funkcji, które pozwalają przekształcać tekst w trakcie generowania szablonu. Oprócz standardowego zestawu udostępnianego przez silnik, w szablonach możemy skorzystać z kilku wygodnych funkcji dodanych przez Android Studio:

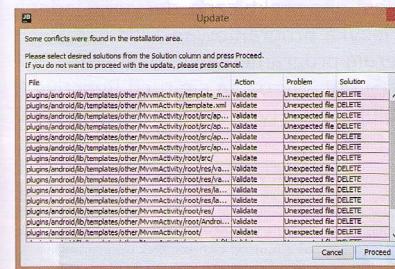
- » activityToLayout(activityClass) – konwertuje nazwę klasy (np. MainActivity) na nazwę odpowiadającego pliku zasobu layoutu (activity\_main).
- » camelCaseToUnderscore(camelStr) – konwertuje ciąg w postaci CamelCase, (np. MainActivity) na snake\_case (np. main\_activity).
- » escapeXmlAttribute(attribute) – przekształca tekst w taki sposób, aby można go było wstawić jako wartość atrybutu XML (zabezpiecza znaki „, „, < i >)
- » escapeXmlText(str) – przekształca tekst, aby można go było użyć w XMLu – zabezpiecza znaki &, < i >, ale nie zabezpiecza znaków „ „.
- » escapeXmlString(str) – przekształca tekst, aby można go było użyć w XMLu – zabezpiecza wszystkie znaki specjalne: &, <, >, „, „, „, „. Oprócz tego przeprowadza dodatkowe konwersje, zabezpieczając znaki specyficzne dla Androida – na przykład poprzedza znak apostrofu backslashem. Ta wersja przydaje się wówczas, gdy przetwarzany ciąg będzie wykorzystany jako zasób tekstowy.
- » extractLetters(str) – usuwa z tekstu wszystkie znaki poza literami.
- » classToResource(str) – konwertuje nazwę klasy na nazwę przyjazną dla zasobów, na przykład MainActivity przekonwertuje na main. Oprócz Activity funkcja usuwa również sufiks Fragment, Provider i Service.
- » layoutToActivity(str) – wykonuje operację odwrotną do activityToLayout().
- » slashedPackageName(str) – przekształca nazwę pakietu na ścieżkę, np. com.spooksoft.myApp na com/spooksoft/myApp.
- » underscoreToCamelCase(str) – wykonuje operację odwrotną do camelCaseToUnderscore().

### PROBLEMY

Podczas pisania własnych szablonów należy mieć na uwadze, że jest to działanie nieprzewidziane (a przynajmniej niewspierane) przez twórców Android Studio. Jeżeli w szablonie jest błąd, może to spowodować zawieszenie się środowiska, dosyć trudno też od-

zyskać informację, co faktycznie poszło nie tak: czasami będziemy więc skazani na metodę próby i błędów.

Oprócz tego nasze szablony podczas aktualizacji są uznawane za nieprawidłowe (mechanizm aktualizacji nie spodziewa się, że w katalogu szablonów znajdzie nowe pliki) i są usuwane (Rysunek 5), dlatego po zakończeniu tego procesu należy przywrócić wszystkie usunięte szablony ręcznie.



Rysunek 5. Problem przy aktualizacji środowiska

## SZABLONY W ANDROID STUDIO

### W sieci:

Silnik FreeMarker:

► <http://freemarker.org/>

Artykuł o szablonach w Android Studio:

► <https://coderwall.com/p/fsvyw/file-templates-in-android-studio>

Jeszcze jeden artykuł na ten temat:

► <https://riggaroo.co.za/custom-file-template-group-android-studio/intelliij/>

Szerzy opis struktury szablonu (format 3):

► <http://www.i-programmer.info/professional-programmer/resources-and-tools/6845-android-ads-sample-format-document.html>

Szerzy opis struktury szablonu (format 4):

► <https://android.googlesource.com/platform/sdk/+refs/heads/master/templates/docs/index.html>

### WOJCIECH SURA

wojciechsura@gmail.com

Programuję od przeszło dziesięciu lat w Delphi, C++ i C#, prowadząc również prywatne projekty. Obecnie pracuję w polskiej firmie PGS Software S.A., zajmującej się tworzeniem oprogramowania i aplikacji mobilnych dla klientów z całego świata.

Daj Się Poznać 2017  
dajsiepoznać.pl

Konkurs programistyczny dla ambitnych

Wyjdź z cienia

Programuj • Bloguj • Dziel się wiedzą • Daj się poznać

## ReactiveX – programowanie reaktywne w niemalże każdym języku i systemie

Programowanie reaktywne, razem z funkcyjnym, już od lat jest zapowiadane jako paradymat przyszłości. O tym, na ile jest ono praktyczne, decyduje jednak głównie wsparcie od strony biblioteki lub języka programowania. W ostatnich latach powstało kilka bibliotek, które wspierają programowanie reaktywne. Pośród nich, szczególnie w świecie Javy i JavaScript, wyróżnia się ruch ReactiveX. Zapewnia on rozbudowane i funkcjonalne biblioteki z tymi samymi narzędziami dla wielu różnych języków i platform.

**S**am ruch ReactiveX koncepcyjnie nie jest nowością. Może się on wreszcie wydawać pewnym déjà vu osobom, które mają doświadczenie z Akka, Rabbit czy Reaktor. Bibliotek, które konkurują o wprowadzenie reaktywności do Javy, jest kilka. Na szczęście powstała także inicjatywa, by wznieść się w Javie ponad te różnice. Mam na myśli Reactive Streams, która działa, by wprowadzić do Javy 9 strumienie, które mogłyby współpracować z tymi bibliotekami. Pokażę nam to, jak duży jest zainteresowanie wprowadzeniem reaktywności do popularnych języków programowania. Jednocześnie fakt, że biblioteki te działają w większości niemalże identycznie, pokazuje, jak dojrzała jest ta technologia. Pojawia się jednak pytanie: Skoro jest tak wiele bibliotek, to czemu akurat skupia się na RxJava? Głównymi powodami, dlaczego uważam, że to właśnie jej należy poświęcić ten artykuł, jest jej uniwersalność i charakter. Jest ona używana w wielu językach (ma wsparcie dla 13 popularnych języków, w tym dla JavaScript, Java, C#, PHP, Ruby, C++, Python i Swift). Społeczność ReactiveX ma ambicję, by wspierać wszystkie popularne platformy i języki programowania. Jest tu temu na dobrą drodze. Nie jest także zależna od framework'a (jak Akka czy Spring Cloud Stream). Także sposób, w jaki RxJava jest napisany, świadczy o jej dojrzałości. Posiada dużą liczbę metod, a przy tym jest na tyle intuicyjna, że większość osób, które korzystały z innej biblioteki do programowania reaktywnego, nie powinna mieć trudności z jej zrozumieniem.

W tym artykule zaprezentuję bibliotekę ReactiveX i pokażę, co może ona wnieść do projektu. Skupią się przy tym na przykładach konkretnych zastosowań. Zaczynając od minimalistycznego wstępu do programowania reaktywnego, mającego formę czystelkowej wejści w temat.

Zdecydowaliśmy się prezentować przykłady w Kotlinie, ponieważ jest on bardzo podobny do Javy, ale bardziej zwięzły i czytelny. Same metody i funkcje pochodzą jednak z RxJava, ponieważ Kotlin jest z Java w pełni kompatybilny. W zasadzie biblioteka RxKotlin dostarcza tylko dodatkowe metody, charakterystyczne dla Kotlin (nie będącymi z nich korzystałi). Warto jednak zauważyć, że w innych językach obsługiwanych przez ReactiveX kod ten będzie wyglądał niemal identycznie. Funkcje są wszędzie takie same i różnica tkwi głównie w elementach składowych języka.

### PROGRAMOWANIE REAKTYWNE

Programowanie reaktywne jako paradymat jest bardzo szeroką koncepcją. W świecie komputera najbardziej znaną jej implemen-

tacją są komórki w programie Microsoft Excel. Definiujemy w nich formuły, które obliczają wartość na podstawie innych komórek. Gdy jednak któraś z wartości tych komórek ulegnie zmianie, to zmieniają się wartości wszystkich komórek od niej zależnych. Jeśli więc komórka zależy od innych, to będzie reagowała zmianą swojego stanu na zmianę stanu dowolnej komórki, od której zależy. Nie wiem, jak realnie działa to w Excelu, ale moim zdaniem najbardziej intuicyjną implementacją byłoby ustalenie po zmianie formuły obserwatorów (observer lub listener) na komórkach, od których wartość zależy. Tak by dowolna ich zmiana powodowała ponowne obliczenie wartości komórek, na które wpływają. Jak widać, programowanie reaktywne jest silnie powiązane ze wzorcem obserwatora. Na końcu artykułu przedstawiony jest kod definiujący zachowanie podobne do komórek programu Microsoft Excel.

COUNT	A	B	C	D	E	F	G
				=B2*C2			
1	Product	Quantity	Price	Amount			
2	bread	2	1.5	=B2*C2			
3	butter	1	1.2				
4	cheese	3	2				
5	ham	3	1.8				
6							
7							

Rysunek 1. Komórki w programie Microsoft Excel (źródło: <http://www.excel-easy.com/functions/cell-references.html>)

Programowanie reaktywne jest w pewien sposób odwróceniem kierunku przepływu danych. Propagacja zmian zaczyna się wtedy, kiedy zmieniony zostanie obiekt, od którego zależą inne. Dotyczy także wyłącznie tych, które od zmienionych obiektów zależą.

W praktyce programistycznej programowanie reaktywne wygląda różnie dla różnych języków programowania i bibliotek. Mimo wielu różnic nieodłącznymi jego elementami są:

- » Strumienie danych (ang. *dataflow*) – jest to sposób programowania, gdzie definiujemy kolejne kroki przetwarzania danych. W kolejnych krokach przyjmuję się jakieś dane i zamienia się je na inne dane. Poszczególne kroki powinny być bezstanowe. Często do przetwarzania danych wykorzystuje się uniwersalne metody, które jako argument przyjmują funkcje lambda. Typowym przykładem jest strumieniowe przetwarzanie list.
- » Propagacja zmian – wcześniej przedstawiona na przykładzie komórek Excela. Polega zwykle na tym, że obiekty definiują,

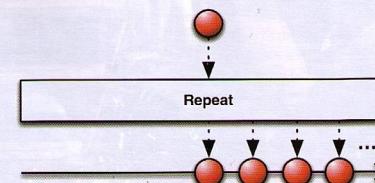
jak obliczany jest ich stan i od czego zależy. W ten sposób zmiana jest propagowana.

Warto zauważyć na tym etapie, że programowanie reaktywne dobrze zgrywa się z wielowątkowością. Zarówno strumienie danych, jak i propagacja zmian nadają się do uruchomienia na wielu wątkach. Był to istotny powód popularyzacji paradymatu reaktywnego.

W praktyce programistycznej częściej używa się strumieni danych i to wokół nich skupione są funkcjonalności ReactiveX. Tworzący jest obiekt, który jest w stanie wyemitować zdarzenie, czylis coś innego jak jakieś dane, które mogą być przekazane dalej. Dla przykładu, możemy utworzyć obiekt, który w odpowiedzi na kliknięcie przycisku wyemituje zdarzenie.

Żeby jednoznacznie nazwać tego typu obiekt, będę używał w tym artykule nazwy Observable. Jest to odpowiednik nazwy Publisher z nomenklatury Javy 9 (Reactive Streams). Wysłane zdarzenie powinno zostać obsłużone. Wcześniej jednak można na nim wykonywać liczne operacje strumieniowe.

Poszczególne operatory w świecie ReactiveX przyjęto się dokumentować dodatkowo za pomocą tzw. Marble Diagrams. Jeden z nich przedstawiony został na Rysunku 2. Metody są tu reprezentowane jako bloki, gdzie nad nimi i pod nimi znajdują się przykładowa oś czasu z zaznaczonymi zdarzeniami. W dalszej części przedstawione będą inne Marble Diagrams, by ułatwić zrozumienie bardziej skomplikowanych funkcji.



Rysunek 2. Marble Diagram metody repeat. Zamienia one Observable (którego jest metodą) na taki, który po odebraniu zdarzenia emituje wiele radośnie co pewien odstęp (źródło: [www.reactivex.io](http://www.reactivex.io))

Przejdźcie do programowania reaktywnego dostarcza szereg korzyści:

1. Obsługa przepływu zawarta jest w jednym miejscu, dzięki czemu dobrze widać, co się dzieje w aplikacji. Dodatkowo biblioteki takie jak ReactiveX posiadają szereg funkcji związanych z definiowaniem tego strumienia, dzięki którym możemy tworzyć bardziej rozbudowaną logikę (na przykład oczekując na kilka odpowiedzi HTTP i wyemitować zdarzenie dopiero po otrzymaniu ostatniej).
2. Wynosimy obsługę wątku, na którym działa kod wyżej, do obsługi strumienia. Wielowątkowość staje się więc bardzo prosta.
3. Tworzone bloki są naturalnie oddzielone od tego, jak oddziałują z resztą kodu. Dzięki temu częściej nadają się do ponownego użycia oraz są łatwiejsze do analizy.

Zdania te mogą brzmieć abstrakcyjnie, ale w dalszej części postaram się przybliżyć te korzyści poprzez konkretne przykłady.

### PRZETWARZANIE STRUMIENIOWE

Zanim przejdę do bardziej praktycznych przykładów działania biblioteki RxJava (i innych z ReactiveX), chciałbym tytułem wstępem

pokażć kilka prostszych przykładów jej działania. Przetwarzanie strumieniowe w RxJava wygląda bardzo podobnie do wprowadzonego w Javie 8 strumieniowego przetwarzania kolekcji. Przykładowe przetwarzanie strumieniowe w RxJava przedstawione zostało w Listingu 1. W Listingu 2 pozostawione zostało natomiast analogiczne przetwarzanie zrealizowane przy pomocy narzędzi przetwarzania strumieniowego wprowadzonych w Javie 8.

Listing 1. Przykładowe przetwarzanie strumienia zdarzeń w RxJava, które wygląda podobnie do przetwarzania kolekcji znanego z Kotlin, Scala lub Javy 9

```

Observable.range(1, 10) // 1
    .map { it * 2 } // 2
    .sumInteger() // 3
    .subscribe { println(it) } // 4
    // sum = 110
  
```

1. Funkcja range generuje zdarzenia kolejno dla liczb z podanego w argumencie zakresu.
2. Metoda map mapuje każde zdarzenie na inne.
3. Metoda sum na ostatnie zdarzenie i wtedy liczy sumę dla wszystkich otrzymanych zdarzeń.
4. Metoda subscribe wykonuje podaną w argumencie operację dla każdego odebranego zdarzenia. Tutaj jest to wyświetlenie sumy.

Listing 2. Działania analogiczne do Listingu 1, ale wykonywane na strumieniach Java 8

```

long sum = IntStream.range(0, 10)
    .asLongStream()
    .map(i1 -> i1 * 2)
    .sum();
System.out.println(sum);
  
```

Istonią różnicę między przetwarzaniem kolekcji a zdarzeń możemy zaobserwować, gdy wprowadzimy opóźnienie między elementami. W Listingu 3 zaprezentowany został przykład, w którym, dzięki wprowadzeniu opóźnienia przez funkcję delay, otrzymamy efekt odliczania.

Listing 3. Przykład kodu, który co sekundę wywołuje funkcję showCountDownNumber z kolejnymi liczbami od 10 do 1

```

Observable.from(10 downTo 1) // 1
    .concatMap { s -> Observable.just(s).delay(1, SECONDS) } // 2
    .subscribe { num -> showCountDownNumber(num) }
  
```

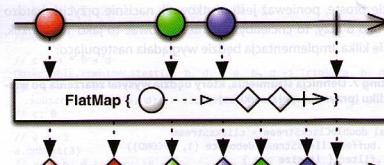
1. Funkcja from tworzy kolejne zdarzenia dla kolejnych elementów kolekcji.
2. Metoda delay wprowadza opóźnienie przed zdarzeniem, a metoda concatMap zmusza eventy, by każdy zaczekał na swojego poprzednika.

Widac tutaj, że w strumieniach ReactiveX kluczowy jest czas nadania zdarzenia. Każdy z kroków przetwarza zdarzenie zaraz po jego otrzymaniu. Część zawarta w concatMap przetrzymuje je jednak i推迟 po kolejnym, a do tego metoda delay wprowadza opóźnienie.

Spójrzmy na bardziej praktyczny przykład. Założmy, że dostajemy zadanie, by na kliknięcie przycisku wysypane było zapytanie o jakiś element, po czym byłyby ony wyświetlane w określonym polu. W większości projektów realizuje się tego typu zadanie poprzez zagnieźdzenie callbacków. Przykładowy kod znajduje się w Listingu 4.

**Listing 4.** Klasyczna realizacja ciągu zdarzeń zrealizowana przez zagnieźdzenie callbacków. Funkcje `thread` i `uiThread` zmieniają wątek zagnieździonego wywołania odpowiednio na nowy wątek oraz na wątek główny

```
button.setOnClickListener {
    button.setOnClickListener {
        thread {
            LoadElement() { elem -> // tutaj zmiany też jest obecna
                uiThread {
                    showElement(elem)
                }
            }
        }
    }
}
```



Rysunek 3. Działanie funkcji `flatMap` na Marble Diagram (źródło: <http://reactivefx.org>)

Już na tak matym przykładzie widać, że wielokrotne zagnieżdżanie jest dobre. Największy problem występuje zwykle w JavaScript (zwłaszcza w Node.js), gdzie programiści wielokrotnie zagnieżdżają kolokalnie „pielęknik callbacków” (*ang. callback hell*). Nie powiedzmy – zmniejsza ono czytelność kodu i utrudniają wprowadzanie zmian. Wydzielanie każdego poziomu do osobnej funkcji niewiele zmienia. Takie funkcje nie nadawały się raczej do ponownego użycia, ponieważ zawsze były w siebie dalszą częśćą przepływu aplikacji. Także obserwacja tego, co każda z nich robi, wymagał będzie schodzenia w dół do coraz bardziej szczegółowych funkcji. Ten sposobu działania nie stanowi więc realnej pomocy dla programisty.

ReactiveX daje jednak alternatywę. Zamiast zagnieździć wywołania, stworzymy `Observable` i ustawimy je w strumieniu. Każdy z nich uruchamiany będzie w odpowiedzi na zdarzenie wysiane przez poprzedni. Definicja takiego strumienia w RxJava przedstawiona jest w Listingu 5.

**Listing 5.** Realizacja ciągu zależnych zdarzeń w RxJava. Oczekiwanie na załadowanie strony, nasłuchiwanie na naciśnięcie przycisku, dodanie elementu z API oraz docelowo wyświetlenie go przy pomocy funkcji `showElement`

```
getUsersObservable() // 1
    .map { u: User -> u.id } // 2
    .observeOn(Schedulers.newThread()) // 3
    .flatMap { id -> getShoppingListObservable(id) } // 4
    .observeOn(Schedulers.io()) // 5
    .subscribe(
        { elem -> addListToView(elem) }, // onNext, 6
        { e -> logError(e) }, // onError, 7
        { informantaboutlistsFilled() } // onComplete, 8
    )
```

1. Tworzymy `Observable`, który wysyła zapytanie do API o listę użytkowników.
2. Mapujemy otrzymanego użytkownika na jego id.
3. Zmieniamy wątek obserwacji na nowy.
4. Dla każdego otrzymanego zdarzenia tworzymy `Observable`, które pobiera listę zakupów. Zdarzenia zawiadamiają o użytkowniku, ponieważ zostały zmapowane w 2.
5. Zmieniamy wątek obserwacji na główny, by móc zmieniać elementy widoku (wymóg Androida).
6. Ustawiamy funkcję lambda, która zostanie wywołana po otrzymaniu poprawnego zdarzenia zawierającego element listy zakupów.
7. Ustawiamy funkcję lambda, która zostanie wywołana po otrzymaniu błędu. Taka sytuacja ma miejsce, gdy dowolny z pośrednich `Observable` się zwróci błędem.
8. Ustawiamy funkcję lambda, która zostanie wywołana po zamknięciu strumienia. Stanie się tak, gdy otrzymamy ostatnią odpowiedź zarówno na zapytanie o użytkowników, jak i o listy zakupów.

W przykładzie wykorzystana została funkcja `flatMap`. Przyjmuje ona jako parametr funkcję lambda, która zwraca nowy obiekt `Observable`. Dla każdego otrzymanego zdarzenia funkcja `flatMap` wywołuje funkcję z argumentem i tworzy nowego `Observable`. Obiekt `Observable`, który jest zwracany przez `flatMap`, będzie emitował wszystkie zdarzenia z utworzonych przez `flatMap` obiektów `Observable`. Zostało to przedstawione na Rysunku 3. W większości przypadków mamy do czynienia z sytuacją, gdy pierwszy `Observable` generuje tylko jedno zdarzenie (jak w Listingu 2). Wtedy dodawanie kolejnych `Observable` przez `flatMap` tworzy klasyczne przetwarzanie strumieniowe. Później zobaczymy jednak, że funkcja `flatMap` ma znacznie większe możliwości.

Łatwo zauważycie, że pojawiły się jednak tutaj elementy, których wcześniej nie było. Przez wszystkim widzimy dodanie obsługi eventów. Dowolny błąd rzucony przez jednego z obserwatorów wpłynie właśnie na `onError`. Zobaczyliśmy także zmianę wątku obserwacji. Wielowatkowość jest silnie powiązana z wzorcem reaktywnym. W ReactiveX wystarczy jedno polecenie, by każde działanie w strumieniu wykonywane było w osobnym wątku. Widać ją

W przykładzie wciąż jednak używamy wyłącznie dwóch metod dostarczanych przez RxJava: `flatMap` i `map`. Jeśli zajrzymy do dokumentacji, to zobaczymy, że tak naprawdę są dziesiątki funkcji, które pozwalają na strumieniowe przetwarzanie obiektów `Observable`. Istnieje szereg metod, które pozwalają na ich łączenie, filtrowanie, agregację i wiele innych operacji. Zobaczmy inny przykład – chcemy zrobić obsługę dwukliku danego elementu. To nie jest

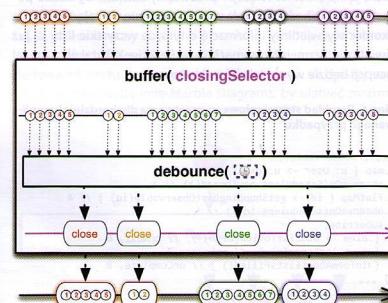
## BIBLIOTEKI I NARZĘDZIA

takie proste, ponieważ jeśli użytkownik naciśnie przycisk bardzo szybko 6 razy, to chcielibyśmy interpretować to jako 1 wieloklik, a nie kilka. Implementacja będzie wyglądała następująco:

**Listing 7. Definicja strumienia, który będzie wysyłał zdarzenia po wielokliku (przynajmniej dwukliku)**

```
val doubleClickStream = clicksStream
    .buffer(clicksStream.debounce(1, SECOND))
    .filter { it.size > 2 }
```

Jest to bardziej skomplikowane przetwarzanie zdarzeń, ponieważ łączy dwie funkcje: `buffer` i `debounce`. `buffer` grupuje zdarzenia w listę. Jest ono sterowane zdarzeniami wysyłanymi przez element obserwowy z argumentem (tutaj `debounce`). `debounce` emituje sygnał po określonym czasie od ostatniego zdarzenia, po którym nie nastąpiło inne. W tym przypadku po 1 sekundzie. Tutaj zdarzenia emitowane przez `debounce` oznaczają uciecie nastawienia dla metody `buffer`. Zostało to przedstawione na Rysunku 4.



Rysunek 4. Jak działa połączenie `buffer` i `debounce` służące do wychwytywania wieloklików (źródło: <https://github.com/RxJava/RxJava/wiki/Backpressure>)

Wygląda to bardzo dobrze, ale co tak naprawdę kryje się pod tymi wszystkimi funkcjami, takimi jak `onClickListener` czy `getUsersObservable`?

## TWORZENIE OBIEKTÓW TYPU OBSERVABLE

Większość tworzonych `Observable` jest w pewnym stopniu odpowiednikiem funkcji z callbackiem. Przykładowa taka funkcja przedstawiona jest w Listingu 8. Tworzenie analogicznego do niej `Observable` zostało natomiast przedstawione w Listingu 9. Jak widać – funkcje te nie wiedzą się różnić. To jest jednak prosty przykład, gdzie generowane jest tylko jedno zdarzenie `onNext`.

**Listing 8. Przykład funkcji wykorzystującej backtracking**

```
fun makeLongOperation(callback: (Data)->Unit) {
    thread {
        val d: Data = LongOperation()
        callback(d)
    }
}
```

Przedstawiona tutaj funkcja `create` to tylko jeden ze sposobów tworzenia `Observable`, ponieważ ReactiveX dostarcza szereg innych metod, dzięki którym możemy je tworzyć z list, wartości, przedziałów czasowych, funkcji i wiele innych. Zobaczmy już w Listingu 3 zastosowanie funkcji `from`, która z przedziału liczb (`IntRange`) wygenerowała zdarzenia zawierające kolejne liczby.

W Androidzie rzadko kiedy definiuje się samodzielnie `Observable`, ponieważ ReactiveX ma dodatkowe wsparcie dla Androida (`RxBinding`), które dostarcza szereg funkcji tworzących różnego rodzaju `Observable` dla elementów widoku. Podobne wsparcie ReactiveX ma dla Netty i Cocoa. W Androidzie jest znacznie więcej bibliotek, które silnie wspierają RxJava, takich jak Retrofit (najpo-

**Listing 9. Przykład funkcji tworzącej `Observable`, analogiczny do funkcji z callbackiem z Listingu 5**

```
fun makeLongOperationObservable() = Observable.create<Data> {
    subscriber ->
    val d = LongOperation()
    subscriber.onNext(d)
}
```

`Observable` mogą generować wiele zdarzeń `onNext`. Zostało to przedstawione w Listingu 10. Tam generowane jest zdarzenie dla każdej z kolejnych liczb. Bardziej praktyczny przypadek tworzenia obiektu obserwowlanego możemy zobaczyć w Listingu 11, gdzie tworzymy obiekt generujący zdarzenie `onNext` za każdym razem, gdy naciśnięty zostanie przycisk podany w argumencie. Poza zdarzeniem `onNext` wygenerowane mogą zostać jeszcze: `onComplete` – zdarzenie generowane, gdy zakończenu ulega obserwacja. `onError` – zdarzenie generowane, gdy wystąpi jakiś błąd.

Obydwia te zdarzenia standardowo zamykają strumień. Przykład `Observable`, który reaguje (wysyła `onNext`) na naciśnięcie przycisku, ale może zostać zamknięty przy pomocy innego przycisku, pokazany został w Listingu 12.

**Listing 10. Tworzenie `Observable`, który emituje wiele zdarzeń i kończy działanie**

```
fun counter(countTo: Int, millisBetween: Int) = Observable.create<Int> {
    subscriber ->
    createInts { subscriber ->
        for (i in 0 until countTo) {
            subscriber.onNext(i)
            Thread.sleep(millisBetween)
        }
    }
}
```

**Listing 11. Tworzenie `Observable`, który wysyła zdarzenia za każdym razem, gdy naciśnięty zostanie przycisk**

```
fun onClicked(view: View) = Observable.create<View> {
    subscriber ->
    view.setOnClickListener { v -> subscriber.onNext(v) }
}
```

**Listing 12. Tworzenie `Observable`, który wysyła zdarzenia za każdym razem, gdy naciśnięty zostanie przycisk, ale może zostać zamknięty poprzez naciśnięcie drugiego przycisku**

```
fun onClickCloseable(eventButton: View, closeObservableButton: View) = Observable.create<View> {
    subscriber ->
    eventButton.setOnClickListener { v -> subscriber.onNext(v) }
    closeObservableButton.setOnClickListener { v -> subscriber.onComplete() }
}
```

Przedstawiona tutaj funkcja `create` to tylko jeden ze sposobów tworzenia `Observable`, ponieważ ReactiveX dostarcza szereg innych metod, dzięki którym możemy je tworzyć z list, wartości, przedziałów czasowych, funkcji i wiele innych. Zobaczmy już w Listingu 3 zastosowanie funkcji `from`, która z przedziału liczb (`IntRange`) wygenerowała zdarzenia zawierające kolejne liczby.

W Androidzie rzadko kiedy definiuje się samodzielnie `Observable`, ponieważ ReactiveX ma dodatkowe wsparcie dla Androida (`RxBinding`), które dostarcza szereg funkcji tworzących różnego rodzaju `Observable` dla elementów widoku. Podobne wsparcie ReactiveX ma dla Netty i Cocoa. W Androidzie jest znacznie więcej bibliotek, które silnie wspierają RxJava, takich jak Retrofit (najpo-

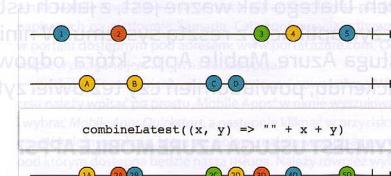
popularniejsza aktualnie biblioteka do operacji sieciowych w Androidzie (czy Nucleus (popularna biblioteka wspomagająca MVP)). Dlatego bardzo rzadko kiedy sami musimy tworzyć Observable.

### PROPAGACJA ZMIAN

Wspomniana na wstępnie propagacja zmian też jest obecna w ReactiveX, ale można ją zrealizować na wiele różnych sposobów. Zazwyczaj uwzględniają one obiekt typu Subject (temat). Jest to obiekt, który jest jednocześnie Observable oraz pozwala na przyjmowanie zdarzeń (poprzez onNext). Przykład jego działania przedstawiony został w Listingu 13.

**Listing 13.** Przykład działania PublishSubject. Do obiektu tego typu w dowolnym momencie można podpiąć kolejnego nasłuchującego oraz można przy jego użyciu wyemitować zdarzenie, które dotrze do wszystkich nasłuchujących

```
// b = a * 2
a.subscribe { i -> b.onNext(i * 2) }
// b: 0
// c = a * b + d
Observable.combineLatest(a, b, d, { a, b, d -> Triple(a, b, d) })
    .subscribe { (a, b, d) -> c.onNext(a * b + d) }
// c: 0
// a <= 3
a.onNext(3)
// a: 3 b: 6 c: 0 c: 18
// d <= 4
d.onNext(4)
// d: 4 c: 22
```



Rysunek 5. Marble Diagram funkcji combineLatest (źródło: <http://reactivex.io/documentation/operators/combinelatest.html>)

### PODSUMOWANIE

Programowanie reaktywne nie tylko wygląda obiecująco, ale także sprawdza się w praktyce. ReactiveX pozwala na tworzenie strumieni i zamiast struktur callbacków, w których obiektów, które o tym stanie powinny decydować. Obiekty typu Subject potrafią także przechowywać stan. Przykład, naśladujący komórki znane z programu Microsoft Excel, przedstawiony został w Listingu 14. Wykorzystana została tam funkcja combineLatest, która emituje zdarzenie w odpowiedzi na zdarzenie z dowolnego Observable przekazanego przez argument. Marble Diagram funkcji combineLatest przedstawiony został na Rysunku 5. Implementacja, przedstawiona w przykładzie z Listingu 14, jest typowym przypadkiem propagacji zmian.

**Listing 14.** Przykład działania BehaviorSubject. Działanie na podobnie jak PublishSubject, ale przechowuje wartość i umożliwia do niej dostęp w dowolnym momencie. Tutaj przykład naśladuje komórki znane z programu Microsoft Excel

```
val a = BehaviorSubject.create()
val b = BehaviorSubject.create()
val c = BehaviorSubject.create()
val d = BehaviorSubject.create()

a.subscribe { i -> println("a: $i") } // a: 0
b.subscribe { i -> println("b: $i") } // b: 0
c.subscribe { i -> println("c: $i") } // c: 0
d.subscribe { i -> println("d: $i") } // d: 0
```

Pozwala na rozwiązywanie wielu problemów oraz wprowadzenie konceptu programowania reaktywnego do języków, które nie posiadają dla niego wbudowanego wsparcia. Sprawdza się nie tylko dla aplikacji komunikujących się z użytkownikiem, ale także dla aplikacji serwerowych. Pozwala, na przykład, na proste wprowadzenie łatwością do projektu, dzięki czemu lepiej są wykorzystywane zasoby urządzenia. Osobibicie polecam ją na podstawie ponad roku doświadczeń z intensywnego wykorzystywania jej w projektach.

Dziękuję Kamilowi Karwowskiemu za cenne komentarze przy pisaniu tego artykułu.

**MARCIN MOSKAŁA**

marcin.moskala@docplanner.com

Programista Kotlin i Groovy na platformie Android pracujący dla ZnanegoLekarza, autor bloga KotlinDev.pl oraz współzałożyciel Nootro (www.nootro.pl). Pasjonat samorozwoju oraz tworzenia pięknego i prostego kodu.



## Azure Mobile Apps i platforma Xamarin

Aplikacje mobilne stają się coraz bardziej popularne. Nie tylko ze względu na gry oraz aplikacje, ale również dlatego, że coraz więcej użytkowników korzysta z urządzeń mobilnych w pracy. Są one nieodłącznym elementem coraz większej ilości tworzonych systemów. Duża część aplikacji mobilnych to interfejs między użytkownikiem końcowym a resztą systemu. Każda bardziej złożona aplikacja mobilna jest spięta z backendem, który odpowiada za komunikację i wymianę danych. Dlatego tak ważne jest, z jakich usług decydujemy się skorzystać, żeby połączyć aplikację z resztą systemu. W niniejszym artykule przedstawiona została usługa Azure Mobile Apps, która odpowiada za wsparcie aplikacji pod kontem backenu, powiadomień czy też uwierzytelnienia.

### CZYM JEST USŁUGA AZURE MOBILE APPS?

Zanim przejdziemy do omawiania poszczególnych komponentów Azure Mobile Apps, warto odpowiedzieć na pytanie, czym w ogóle jest ta usługa?

Azure Mobile Apps jest w pełni zarządzaną usługą w modelu Platform as a Service, wchodząącą w skład większej, Azure App Service, oferowanej na platformie chmurowej Microsoft Azure.

Na poniższym rysunku widać, z jakich usług składa się Azure App Service. Jedną z nich jest właśnie Azure Mobile Apps.



Rysunek 1. Komponenty usługi Azure App Service



Rysunek 2. Konfiguracja usługi Azure Mobile Apps

Samo zastosowanie oraz rosnący z nim koszt różnią się w zależności od zastosowań, dlatego możliwe jest wybór odpowiedniego planu. Przedział jest spory – od darmowego (który mówiąc wprost, nie nadaje się do tworzenia rozbudowanego backendu) do 750 Euro za miesiąc.

Przedzielimy zatem do omówienia poszczególnych komponentów usługi Azure Mobile Apps.

### KOMPONENTY USŁUGI AZURE MOBILE APPS



Rysunek 3. Rest API

REST API jest jednym z komponentów, który umożliwia komunikację między naszą aplikacją mobilną a backendem znajdującym się na platformie chmurowej Azure. Dzięki temu developer może w łatwy sposób zapewnić dostęp do określonych części systemu, na przykład baz danych. Dane przesypane są w lekkim formacie JSON, co zapewnia łatwość w komunikacji.

Jeśli chodzi o język, w którym napisany jest backend, mamy dwie możliwości: C# lub Node.js.

## AZURE MOBILE APPS I PLATFORMA XAMARIN

Bardzo dużą zaletą Azure Mobile Apps jest dostarczenie natywnych SDK, które możemy używać w aplikacjach iOS, Android, Windows czy też wieloplatformowych (Xamarin, Apache Cordova).



Rysunek 4. Komponent baz danych

Kolejnym komponentem, który zapewnia usługę, jest baza danych na platformie Azure.

Jak widać na powyższym rysunku, Mobile Apps oferują całą gamę różnych baz danych (w tym relacyjnych, nierelacyjnych), takich jak SQL, MongoDB, DocumentDB czy Table Storage. Bardzo istotnym elementem jest to, że taką bazę możemy w łatwy sposób połączyć ze stworzonym wcześniej backendem. Procedura jest naprawdę prosta i wymaga kilku kliknięć w portalu Azure.

Warto również wspomnieć, że Azure Mobile Apps zapewnia w tym przypadku protokoły Open Data w wersji 3, który jest przyjazny pod kątem wymiany danych z aplikacją mobilną.

Jak wcześniej wspomniałem, istnieje również możliwość synchronizacji danych, które zostały zebrane, kiedy aplikacja nie miała dostępu do Internetu. To wszystko zapewnia wspomniane wcześniej SDK, które pozwala w łatwy sposób taką synchronizację zaimplementować.



Rysunek 5. Komponent uwierzytelnienia

Temat uwierzytelnienia jest jednym z bardziej popularnych, jeśli chodzi o aplikacje mobilne. Często można napotkać problemy, w których sposób zapewnienia dostępu do systemu dla pomoce urządzeń mobilnego. W tej kwestii bardzo pomocny jest komponent uwierzytelnienia, oferowany przez Azure Mobile Apps. Oczywiście oprócz standardowego uwierzytelnienia przy użyciu loginu oraz hasła mamy możliwość dodania łatwego logowania przez serwisy społecznościowe, jak Facebook czy Twitter. Logowanie przy użyciu konta Microsoft czy Google także nie stanowi problemu. Warto również wspomnieć o możliwości logowania za pomocą kont Active Directory, co jest szczególnie przydatne w korporacjach.



Rysunek 6. Komponent powiadomień

Powiadomienia push są coraz częściej wykorzystywane w aplikacjach mobilnych.

Nie tylko ze względu na informowanie użytkownika o nowych funkcjonalnościach, ale również w celu zachęcania użytkownika do częstego zaglądania do aplikacji.

Jeśli zwróciliśmy uwagę na Rysunek 6, komponent Push Notifications zapewnia możliwość integracji z serwisami powiadomień, takimi jak Windows Notification Service, Apple Push Notification Service czy też Google Cloud Messaging. Daje to ogromne możliwości, ponieważ developer ma możliwości integracji tego komponentu z backendiem oraz decyzyjny, na jaką platformę wysłać powiadomienie. Oczywiście Azure Mobile Apps zapewnia SDK, które w łatwy sposób można zintegrować z aplikacją w celu konfiguracji i odbierania powiadomień.

## AZURE MOBILE APPS Z PLATFORMĄ XAMARIN

Azure Mobile Apps zapewnia wsparcie dla aplikacji cross-platform, napisanych na platformie Xamarin. Cała konfiguracja odbywa się w portalu dostępnym pod adresem: [www.portal.azure.com](http://www.portal.azure.com). Oczywiście wymagana jest aktywna subskrypcja.

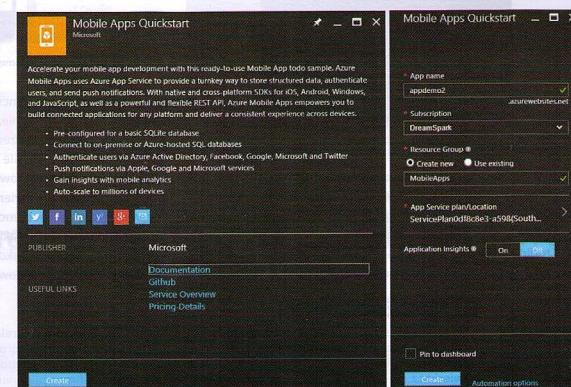
Kiedy już się zalogujemy, możemy stworzyć nową usługę. W tym celu należy wpisać po prostu „Mobile Apps” w oknie wyszukiwania i wybrać Mobile Apps Quickstart, a następnie kliknąć w przycisk Create. W kolejnym oknie, które się pojawi, należy wpisać adres URL, pod którym dostępna będzie nasza usługa. Należy również wybrać subskrypcję oraz wpisać nazwę grupy zasobów (w tym przypadku po prostu MobileApps). Wszystkie kroki zostały przedstawione na poniższych Rysunkach 7 i 8.



Rysunek 7. Wybór usługi Azure Mobile Apps w portalu Azure

Po kilku sekundach powinno się wyświetlić informacja o poprawnie skonfigurowanej usłudze. Przejdzmy do niej zatem i zobaczymy niektóre z możliwości.

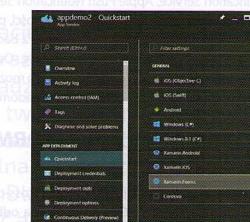
Interesują nas zakładka Quickstart znajdująca się z lewej strony. Jak widać, mamy szereg możliwości, jeśli chodzi o integrację z platformą.



Rysunek 8. Tworzenie nowej usługi Azure Mobile Apps

## PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

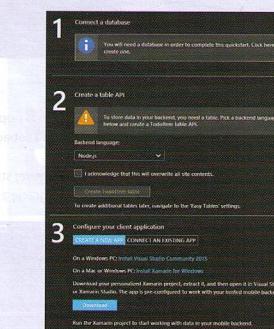
formami mobilnymi. W tym przypadku interesuje nas platforma Xamarin oraz aplikacja napisana w Xamarin Forms.



Rysunek 9. Łączenie Azure Mobile Apps z aplikacją Xamarin Forms

Po wybraniu powinno pojawić się trzecia zakładka. Widać tutaj trzy sekcje:

- » **Connect to database** – zapewnia możliwość wyboru bazy danych
- » **Create a table API** – zapewnia możliwość wyboru języka, w którym napisany jest backend
- » **Configure your client application** – tutaj usługa oferuje dwie możliwości. Pierwszą z nich jest pobranie szablonu aplikacji Xamarin Forms z już wpięтыm SDK. Drugą jest instrukcja, jak zapewnić dostęp do backendu dla już istniejącej aplikacji. Na Rysunku 10 przedstawiono kartę konfiguracyjną.

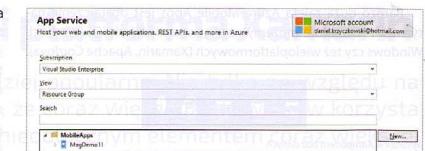


Rysunek 10. Konfiguracja backendu w Azure Mobile Apps

Należy natomiast pamiętać, że kod backendu, który również jest dostępny do pobrania, powinien być opublikowany na Azure w pierwszej kolejności, co przedstawiono na Rysunkach 11 i 12. Trzy proste kroki: kliknięcie prawym przyciskiem na projekt w Visual Studio i wybór opcji „Publish”, wybór subskrypcji, na której stworzyliśmy usługę, kliknięcie przycisku „Publish”.



Rysunek 11. Wybór subskrypcji Azure



Rysunek 12. Publikacja kodu backendu na platformie Azure

Przykład gotowej aplikacji, pobranej z portalu Azure możemy zobaczyć na rysunku poniżej. Zaraz po pobraniu projektu można go uruchomić. Użytkownik ma możliwość dodania i usuwania zadań, które są zapisywane w bazie danych.



Rysunek 13. Aplikacja Xamarin Forms

Oczywiście w tym momencie mamy możliwość konfiguracji innych usług, jak uwierzytelnienie, powiadomienia push czy dodatkowa baza danych oraz połączenia do nich w aplikacji Xamarin Forms.

Jak widać, Azure Mobile Apps nie jest tylko forma backendu dla aplikacji mobilnych, ale również kompleksową usługą, wspierającą powiadomienia push, uwierzytelnienie użytkowników czy mierzenie wydajności. W sieci można znaleźć bardzo dużo przykładów wraz z dokumentacją, jak poprawnie skonfigurować określone komponenty, co nie jest skomplikowane i czasochłonne.

Zachęcam do przetestowania usługi, nie będziecie rozczarowani.

# Android na sterydach: Native Activity w parze z OpenGL ES 3.0

Tworzenie gier na platformy mobilne często wiąże się z trudnymi decyzjami. Jedną z nich jest także wybór technologii. Skorzystanie z C++ na systemie tak mocno kojarzącym się z technologiami zarządzanymi może wydać się bolesną decyzją. Zyski, jakie można jednak osiągnąć z tego tytułu, mogą przekroić kilka niedogodności.

## C++ W TWORZENIU GIER

W branży gier video C++ zdominował się na dobre. Choć procesory są coraz szybsze, a układy graficzne przetwarzają niezliczoną ilość wielokątów, sztuka wykorzystania tej mocy ciągle zależna jest od technologii, z jakiej korzystamy. C++ jednak nie jest naszym ulubieńcem. Jest kryprzyń, niesforną i częstą powodem problemów. W nieodpowiednich ręках potrafi być niebezpiecznym narzędziem, mogącym eksplodować w każdej chwili. Konkurencja jest bardzo silna. Wiele języków przedewszystkiem do bycia, lepszym C++ – prostą składnią, obszernejszą biblioteką standarową, nowoczesne narzędzia.

C++ jednak nigdzie się nie wybiera. Można wręcz rzec, że ma się całkiem dobrze. Rozwój języka nabrął tempa, środowiska programistyczne świetnie wspomagają pracę programisty, zaś narzędzia statycznej analizy kodu pomagają uniknąć niespodziewanych kłopotów. W świecie zdolnym przez maszyny wirtualne i interpretatory natywne podejście oferowane przezowy język wydaje się jednak nieco archaiczne. Kompilacja dla różnych platform czy różnice w zachowaniu kompilatorów wydają się być sporem kłopotem. Jednak ta bliskość sprzętu w połączeniu z dość wysokim poziomem abstrakcji i deterministycznym wykonaniem kodu jest dziś główną siłą C++. To dzięki tym właściwościom programista ma pełną kontrolę nad wykonaniem programu – niezbędna podczas tworzenia gier i symulacji czasu rzeczywistego. Tutaj nie ma miejsca na niespodziewane spadki wydajności, spowodowane działaniem *garbage collector*.

Platformy mobilne nie są tutaj wyjątkiem. Ograniczone możliwości sprzętowe wraz z krótkim czasem działania na baterii stanowią dla twórców niemal wyłącznie wyzwanie. Optymalne wykorzystanie zasobów jest zatem bardzo istotne. Dodatkowym aspektem jest wsparcie dla kolejnych platform. Android, iOS lub Windows wymagają użycia różnych API czy nawet języków programowania, aby aplikacja mogła zostać stworzona natywnie. Z dość dużą przychodzi jednak C++. Jest to świetne narzędzie do implementacji jednolitej logiki programu. Elementy charakterystyczne dla konkretnego systemu operacyjnego mogą być napisane już oddzielnie. Dzięki temu zachowujemy obszerną wspólną bazę kodu, wraz ze wsparciem różnych właściwości konkretnej platformy. Zadania ułatwia także obszerna baza bibliotek napisanych w C lub C++. Ładowanie zasobów, wielowątkowość, sprzętowe renderowanie czy komunikacja sieciowa mogą mieć wspólną implementację, niezależną od systemu, na którym działają.

## JNI – ROZMOWY C++ Z JAVA

NDK (Native Development Kit) zostało wprowadzone wraz z premierą Androida 1.5 Cupcake. Google dało dostęp do tworzenia bibliotek

napisanych w C z minimalnym wsparciem dla C++. W kolejnej wersji programiści uzyskali dostęp do OpenGL w wersji 1.0 i 1.1. Nie było jednak możliwości stworzenia kontekstu renderowania po stronie kodu natywnego. API na poziomie dziewiątym (Android 2.3 i wyższy) dało możliwość stworzenia tzw. NativeActivity. Od teraz cała aplikacja mogła być napisana bez konieczności korzystania z Java. W kolejnych wydaniach wprowadzono wsparcie dla wyższych wersji OpenGL, sprzyjając obsłudze dźwięku za pomocą OpenSL itp. Toolchain uległ także znacznym zmianom, a od kilku ostatnich wersji GCC jest wypierany przez Clanga. Zestaw bibliotek systemowych pozostaje ciągle dość ubogi. NDK jednak jest skierowany głównie do programistów gier. Ci otrzymują wszystko, co niezbędne: dostęp do sensorów, sprzętowego dźwięku i grafiki, a także operacji na systemie plików czy Google Play Games Services.

Co jednak, gdy zaistnieje konieczność komunikacji z bibliotekami napisanymi w Javie?

Zostaje wtedy jedno wyjście: JNI – Java Native Interface. Jest to pojemny łącznik między obiektami obu środowisk. Dlatego jednak jest stosunkowo prosty. GetObjectClass zwrota identyfikator klasy na podstawie uchwytu obiektu. Dzięki temu, za pomocą GetFieldID, można dostać się do pola message. Typ pola określony jest ostatnim parametrem: "Ljava/lang/String;".

Kolejnym krokiem jest utworzenie tekstu w formacie UTF (NewStringUTF) oraz przypisanie go do pola (SetObjectField) obiektu pobranego klasy.

Po stronie Java metoda wywoływaną jest standardowo:

```
Listing 1. Deklaracja metody natywnej w Javie
public class HelloJni extends Activity
{
    ...
    public native void setMessageFromJNI();
    static {
        System.loadLibrary("hello-jni");
    }
}
```

Na pierwszy plan wychodzi tu konieczność skorzystania z modyfikatora native. Jest to informacja, że implementacja metody musi znajdować się w oddzielnej bibliotece (w tym przypadku hello-jni). W przypadku jej braku kompilator nie zgłosi żadnego ostrzeżenia, jednak w trakcie wykonania zostanie wyrzucony wyjątek UnsatisfiedLinkError.

Przedjdźmy do implementacji:

```
Listing 2. JNI w C++
extern "C"
void Java_com_example_hellojni_HelloJni_setMessageFromJNI(
    JNIEnv *pEnv, jobject object)
{
    std::string text("Hello world from JNI!");
}
```

```
jclass clazz = pEnv->GetObjectClass(object); const jmethodID messageField =
pEnv->GetMethodID(clazz, "setMessage", "(Ljava/lang/String;)V");
jstring message = pEnv->NewStringUTF(text.c_str());
pEnv->SetObjectField(object, messageField, message);
```

Jak widać, pierwszym przyjmowanym parametrem funkcji jest JNIEnv. To struktura dająca dostęp do wszystkich niezbędnych funkcji JNI. Następnie przekazywany jest uchwyt na obiekt, który woła funkcję. Dodatkowo, jeśli komplikacja jest po stronie C++, modyfikator extern "C" wyłącza maglowanie nazw (<https://isocpp.org/wiki/faq/mixing-c-and-cpp>).

Implementacja jest stosunkowo prosta. GetObjectClass zwraca identyfikator klasy na podstawie uchwytu obiektu. Dzięki temu, za pomocą GetFieldID, można dostać się do pola message. Typ pola określony jest ostatnim parametrem: "Ljava/lang/String;".

Kolejnym krokiem jest utworzenie tekstu w formacie UTF (NewStringUTF) oraz przypisanie go do pola (SetObjectField) obiektu pobranego klasy.

Po stronie Java metoda wywoływaną jest standardowo:

Listing 3. Wywołanie metody natywnej w Javie

```
public void onCreate(Bundle savedInstanceState)
{
    ...
    newHelloJni.set();
    ...
    set();
}
```

Jak widać, ilość pracy, jaką należy wykonać po stronie kodu natywnego, jest całkiem spora, nawet dla tak prostych operacji jak przypisanie tekstu do pola klasy. Dlatego można skorzystać z gotowych rozwiązań, takich jak Djinni, które w znacznym stopniu ułatwiają korzystanie z JNI.

## NATIVE ACTIVITY. C++ ZOSTAŁ SAM

Dzięki Native Activity istnieje możliwość stworzenia niezależnej aplikacji w C++. Cała komunikacja z frameworkiem jest wykonywana w tle, programista zaś musi zaimplementować kilka callbacków odpowiedzialnych za reakcję aplikacji na poszczególne zdarzenia.

Najprostszy szkielet programu mógłby wyglądać następująco:

Listing 4. Szkielek aplikacji Native Activity

```
static void handleCmd(struct android_app *pAppState,
                      int32_t cmd)
{
    switch (cmd)
    {
        case APP_CMD_GAINED_FOCUS:
            LOGI("Focus gained");
            break;
        case APP_CMD_LOST_FOCUS:
            LOGI("Focus lost");
            break;
        case APP_CMD_INIT_WINDOW:
            LOGI("Created");
            break;
        case APP_CMD_TERM_WINDOW:
            LOGI("Released");
            break;
    }
}

void android_main(android_app *pAppState)
{
    pAppState->onAppCmd = handleCmd;
    pAppState->onInputEvent = handleInput;
    while (0 == pAppState->destroyed)
    {
        int32_t ret = 0;
        int32_t events = 0;
        android_poll_source *pPoolSource;
```

```
while (0 <= (ret = ALooper_pollAll(0, &events, &events,
                                    &pPoolSource)))
{
    if (nullptr != pPoolSource)
        pPoolSource->process();
}
```

```
if (0 <= pPoolSource->process(&events))
{
    if (ALOOP_ERROR_TIMEOUT == events)
        break;
}
```

Punktem wejścia jest `android_main`. Jest to pierwsza funkcja wywoływana przez Native Activity w pliku `android_native_app_glue.c` (domyślnie dołączanym do projektu), po utworzeniu instancji aplikacji. Ustawione są tutaj dwa callbacki na zdarzenia przychodzące do aplikacji. Dodatkowo zaimplementowane jest przetwarzanie komunikatów, za pomocą `ALooper_pollAll`. Istotny jest tutaj pierwszy parametr określający timeout (w milisekundach) czekania na nowe zdarzenia – przekazanie wartości -1 oznacza nieskończoność.

Funkcją zwrotną `handleCmd` odpowiadającą jest obsługa zdarzeń systemowych, takich jak stworzenie lub zniszczenie okna

czy przejście aplikacji w tło. Działanie aplikacji powinno być tutaj skutkowane.

Listing 5. Obsługa komunikatów systemowych w Native Activity

```
static void handleCmd(android_app *pAppState,
                      int32_t cmd)
{
    switch (cmd)
    {
        case APP_CMD_GAINED_FOCUS:
            LOGI("Focus gained");
            break;
        case APP_CMD_LOST_FOCUS:
            LOGI("Focus lost");
            break;
        case APP_CMD_INIT_WINDOW:
            LOGI("Created");
            break;
        case APP_CMD_TERM_WINDOW:
            LOGI("Released");
            break;
    }
}
```

Drugi z callbacków, czyli `handleInput`, wywoływany jest w przypadku wykrycia przez system interakcji z użytkownikiem, np. dotknięcie ekranu.

**EGL** – OpenGL Es 2.0 dla Androida. Dla aplikacji natywnej jest to jedynie interfejs do zarządzania rysunkami.

Mając gotową podstawową strukturę aplikacji, można przejść do inicjalizacji kontekstu renderowania. Służy do tego EGL – multipłatformowe API pośredniczące między interfejsem graficznym (takim jak OpenGL) a systemem okien.

Jako że każdy system okien komunikuje się z aplikacją w innym sposobie, EGL dostarcza uniwersalny typ `EGLDisplay` reprezentujący



## BUFORY GEOMETRII

Podstawowym budulem danych, które wyświetla OpenGL, jest wierzchołek. To z niego składają się trójkąty, które później budują bardziej skomplikowane struktury w grafice trójwymiarowej. Jak było wspomniane wcześniej, wierzchołek może składać się z atrybutów, które opisują właściwości wierzchołka. Zazwyczaj wartości te nie zmieniają się. Wszelkie zmiany dotyczące np. położenia wierzchołka w przestrzeni są wykonywane w shaderach na podstawie macierzy transformacji; dzięki temu model może być odpowiednio przekształcany (podstawowe przekształcenia to translacja, obrót i skalowanie).

Geometria danego modelu może być ładowana z pliku lub wygenerowana proceduralnie. Ważne, aby efektem końcowym była lista wierzchołków, gotowych do namalowania na ekranie. Miejscem przechowywania takiej geometrii powinna być pamięć GPU. Wówczas dostęp do danych jest najszybszy.

Aby geometria znalazła się w pamięci graficznej, należy posłużyć się buforem wierzchołków (Vertex Buffer Object – VBO). Kod tworzący VBO i kopiący do niego dane wygląda następująco:

**Listing 14. Kopiowanie geometrii do VBO**

```
struct Vertex
{
    //... informacje o konkretnym wierzchołku
    Vector3 position;
    Color4 color;
    //... informacje o kolejnych wierzchołkach
};

Vertex vertices[] = {
    //... kolejne wierzchołki
    Vertex{ -0.5f, -0.25f, -1.0f },
    { 0.5f, 0.25f, -1.0f },
    { 0.5f, -0.25f, -1.0f },
    { 0.25f, 0.25f, -1.0f },
    { 0.25f, -0.25f, -1.0f },
    { -0.25f, 0.25f, -1.0f },
    { -0.25f, -0.25f, -1.0f },
    { 0.0f, 0.0f, -1.0f }
};

GLuint vbo = 0;
glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

Jak widać, informacje o geometrii przechowywane są w tablicy `vertices`. Sam wierzchołek jest bardzo prosty i zawiera tylko pozycję w przestrzeni trójwymiarowej i kolor. Kolejnym krokiem jest utworzenie bufora. Zajmuje się tym funkcja `glGenBuffers()`. W pierwszym parametrze przyjmuje ilość buforów, jakie należy utworzyć, i tablice ich identyfikatorów. Każde odwołanie się do tak stworzonego bufora będzie wymagało podania jego identyfikatora zapisanego w zmiennej `vbo`. Funkcja `glBindBuffer` informuje sterownik OpenGL, że identyfikator `vbo` należy powiązać z buforem geometrii (`GL_ARRAY_BUFFER`). Od tej pory wszystkie operacje wykonywane na buforze geometrii będą odnosić się do bufora o numerze zapisanym w `vbo`. Kolejnym etapem jest kopiowanie danych. Zajmuje się tym funkcja `glBufferData`. Kopiuje ona dane do aktywnego bufora geometrii (parametr `GL_ARRAY_BUFFER`). Będą to dane o długości `sizeof(vertices)`, a ich początek wskazuje adres `vertices`. `GL_STATIC_DRAW` informuje sterownika, że dane w buforze nie będą aktualizowane. Informacja ta pozwala odpowiednio umieścić dane w pamięci graficznej.

Zawolanie `glBindBuffer` z ostatnim parametrem ustawionym na wartość 0 spowoduje, że funkcje modyfikujące bufor geometrii nie będą już wykonywane na obiekcie wskazywanym przez zmiennej `vbo`. Jest to bardzo ważna operacja i należy ją zawsze wykonać po skończonej pracy z buforem.

Usuwanie bufora odbywa się analogicznie do jego tworzenia:

**Listing 15. Usuwanie bufora wierzchołków**

```
glDeleteBuffers(1, &vbo);
```

## ŁĄCZENIE GEOMETRII Z SHADERAMI

Zanim cokolwiek zostanie namalowane, należy odpowiednio powiązać geometrię przechowywaną w utworzonym wcześniej buforze wierzchołków z atrybutami odczytywanymi w programie cieniującym wierzchołki. Zostanie wykonane to w dwóch etapach: po stronie programu głównego (API OpenGL) i shadera. OpenGL wykorzystuje do tego dwie funkcje:  `glEnableVertexAttribArray` oraz  `glVertexAttribPointer`. Pierwsza z nich aktywuje możliwość odczytu atrybutu wierzchołka. Atrybut identyfikowany jest po indeksie. Dla zaprezentowanej wcześniej struktury Vertex atrybut `position` ma indeks 0 (wywołanie  `glEnableVertexAttribArray(0)`), zaś `color` 1 ( `glEnableVertexAttribArray(1)`).

Następnie  `glVertexAttribPointer` wskazuje właściwości i położenie konkretnego atrybutu w buforze. Dla wskazania położenia w strukturze `Vertex` wywołanie funkcji wyglądać będzie następująco:

**Listing 16. Wskazanie położenia atrybutu pozycji**

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                      sizeof(Vertex),
                      (void*)offsetof(Vertex, position));
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

Pierwszy parametr to indeks wcześniej aktywowanego atrybutu. Następnie przekazywana jest liczba komponentów atrybutu oraz ich typ (dla pozycji mamy trzy wartości typu `float`). Następnie określamy, czy wartości są znormalizowane. Dwa ostatnie parametry dotyczą bezpośrednio położenia: `sizeof(Vertex)` informuje sterownika OpenGL, że kolejne atrybuty tego typu są przesunięte względem siebie o wielkość wierzchołka, zaś `offsetof(Vertex, position)` zwraca przesunięcie atrybutu w bajtach względem początku struktury. Dla koloru funkcja będzie zawolana następująco:

**Listing 17. Wskazanie położenia atrybutu koloru**

```
glVertexAttribPointer(1, 4, GL_FLOAT, GL_FALSE,
                      sizeof(Vertex),
                      (void*)offsetof(Vertex, color));
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

Skoro OpenGL został poinformowany, jak wygląda ułożenie wierzchołków w buforze, należy informacje te przekazać do shadera wierzchołków.

W nim będziemy mieć dwa atrybuty:

Wszystkie zmiany powinny być dokonane w kodzie, który odkomentował odczyt atrybutów.

**Listing 18. Atrybuty wierzchołka w shaderze**

```
in vec3 attributePosition;
in vec4 attributeColor;
```

Aby powiązać pierwszy (`attributePosition`) z zerowym atrybutem w buforze geometrii, należy skorzystać z kolejnego kwalifikatora: `layout(location = x)`, gdzie `x` odpowiada wartości przekazanej do  `glEnableVertexAttribArray`. Finalnie mamy więc:

**Listing 19. Przypisane atrybuty wierzchołka**

```
layout(location = 0) in vec3 attributePosition;
layout(location = 1) in vec4 attributeColor;
```

## KOMPILACJA I ŁĄCZENIE PROGRAMÓW CIENIUJĄCYCH

Ostatnim etapem przygotowującym do wyświetlenia pierwszego prymitywu na ekranie jest komplikacja i połączenie shaderów w jeden program cieniujący. Na początek należy utworzyć obiekt shadera, przechowywany w zmiennej typu `GLuint`:

**Listing 20. Tworzenie shadera**

```
GLuint shader = glCreateShader(type);
```

Parametr `type` może przyjąć wartość `GL_VERTEX_SHADER` (dla vertex shadera) oraz `GL_FRAGMENT_SHADER` (dla fragment shadera). Kolejnym krokiem jest dołączenie źródeł do utworzonego obiektu:

**Listing 21. Dodawanie źródła do shadera**

```
glShaderSource(shader, 8pSource, nullptr);
```

Funkcja ta może przekazać do obiektu wiele źródeł (stąd podwójny wskaźnik w parametrze). Ostatni `nullptr` informuje API, że wszyscy źródła kończą się znakiem 0 – w przeciwnym wypadku należy jawnie przekazać długość każdego łańcucha znaków. Teraz zostaje już tylko komplikacja:

**Listing 22. Komplikacja shadera**

```
glCompileShader(shader);
```

Wynik komplikacji (może być zakończony błędem) poznamy, wolając:

**Listing 23. Sprawdzanie statusu komplikacji shadera**

```
GLuint status = GL_FALSE;
glGetShaderiv(shader, GL_COMPILE_STATUS,
              &status
            );
```

Jeśli zmienna `status` ma wartość `GL_FALSE`, komplikacja nie powiodła się. Dodatkowo istnieje możliwość odczytania dokładnego błędu komplikacji, wraz z dokładnym miejscem w kodzie powodującym problem:

**Listing 24. Pobieranie tekstu błędu komplikacji shadera**

```
GLint errorStringLength;
glGetShaderiv(shader,
             GL_INFO_LOG_LENGTH,
             &errorStringLength
           );
//...
glGetShaderInfoLog(shader,
                    errorStringLength,
                    &errorStringLength,
                    pMessage
      );
```

Funkcja `glGetShaderiv` pobiera długość wiadomości błędu (`GL_INFO_LOG_LENGTH`) oraz zapisuje ją w `errorStringLength`.

Następnie `glGetShaderInfoLog` zapisuje w `pMessage` komunikat o długości `errorStringLength`, dla obiektu shader, którygo nie udało się skompilować.

Mając skompilowane dwa shadery, możemy je zlinkować w jeden program. Zrobi to funkcja `glLinkProgram`, jednak wcześniej trzeba utworzyć obiekt programu:

**Listing 25. Tworzenie programu**

```
GLuint program = glCreateProgram();
```

Mając go, kolejnym krokiem jest podpięcie skompilowanych shaderów:

**Listing 26. Dodawanie shaderów do programu**

```
glAttachShader(program, vertexShader);
glAttachShader(program, fragmentShader);
```

Gdzie `vertexShader` i `fragmentShader` to uchwyty na wcześniejszej z sukcesem skompilowane obiekty shaderów. Teraz już można wywołać:

**Listing 27. Linkowanie programu**

```
glLinkProgram(program);
```

Także tutaj można uzyskać informacje o statusie powodzenia operacji:

**Listing 28. Sprawdzanie statusu łączenia**

```
GLuint status = GL_FALSE;
glGetProgramiv(program, GL_LINK_STATUS, &status);
```

Jak widać, wygląda to analogicznie jak przy komplikacji shaderów. Do pobrania długości komunikatu o błędzie służy `glGetProgramiv`, natomiast `glGetProgramInfoLog` zapisze go we wskazanym buforze.

Finalnie nasze shadery będą wyglądać następująco:

```
#Vertex shader
#version 300 es
layout(location = 0) in vec3 attributePosition;
layout(location = 1) in vec4 attributeColor;
out vec4 vertexColor;
void main(void)
{
    gl_Position = vec4(attributePosition, 1.0f);
```

## PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

```
vertexColor = attributeColor; glbind uzycie elaznego wierzchołka do głąbni
}

//Fragment shader
#version 300 es
out vec4 fragmentColor;
in vec4 vertexColor;
void main(void)
{
    fragmentColor = vertexColor;
}
```

Nowością jest tutaj `#version 300 es`. Jest to polecenie preprocesora informujące kompilator GLSL o wersji języka, z jakiej korzystamy. Poczytając od OpenGL ES 3.0, wersja języka jest taka sama jak wersja głównego API. Dodatkowo, jako że fragment shader nie może przyjmować atrybutów wierzchołków, kolor musi być dalej przekazany do shadera fragmentów, gdzie jest ostatecznie przypisywany do wyjścia.

### MAŁOWANIE

Malowanie odbywać się powinno w pełni głównej programu, po obsłudze wszystkich komunikatów. Składać się ono będzie z kilku podstawowych etapów: wyczyszczenie bufora tylnego, renderingu obiektów oraz zamiany buforów.

Pierwszy etap ogranicza się do wywołania dwóch funkcji:

```
Listing 30. Czyszczenie buforów głębi i koloru
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
```

Pierwsza z nich, na podstawie przekazanych parametrów, czyści bufor koloru i głębi. Druga zaś ustawia bufor koloru na wartości przekazane w parametrach (kolor biały).

Następnie wskazujemy, jaki program cieniowania wierzchołków należy użyć oraz jakiego bufora geometrii skorzystać:

```
Listing 31. Aktywacja bufora geometrii oraz programu cieniującego
glUseProgram(program);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

Pierwszym parametrem informujemy sterownik, że chcemy, by namalowane zostały trójkąty. Następnie określamy indeks pierwszego wierzchołka (0) oraz liczbę wierzchołków (6).

Teraz pozostało nam tylko odwiązać użytku program i bufor wierzchołków oraz wysłanie danych na ekran:

**Listing 33. Kończyły etap renderowania**

```
glBindBuffer(GL_ARRAY_BUFFER, 0);
glUseProgram(0);
eglSwapBuffers(display, surface)
```

Na wyświetlaczu telefonu powinien ukazać się następujący widok:



Rysunek 1. Wynik działania programu

### MATEUSZ SEMEGEN

Mateusz Semegen, 25 lat, student studiów magisterskich na Wydziale Inżynierii Biomedycznej Politechniki Warszawskiej. Współpracuje z firmą Dynamic Reflectance, zajmującą się tworzeniem aplikacji mobilnych. Interesuje go programowanie, nauka nowych technologii, rozwijanie swoich umiejętności i tworzenie projektów.

Samo malowanie ogranicza się do jednej funkcji:

```
Listing 32. Malowanie
glDrawArrays(GL_TRIANGLES, 0, 6);
```

Jest to całkiem niewiele jak na ilość pracy, jaką wykonaliśmy, ale teraz będzie już tylko lepiej.

Zródła do aplikacji przedstawionych w artykule można znaleźć na repozytorium GitHub autora: <https://github.com/mse-meggen>. W kolejnej części artykułu zostaną przedstawione takie zagadnienia jak bufor indeksów, teksturowanie czy podstawowe przekształcenia.

Duszą startupowiec, sercem freelancer, po godzinach perkusista. Programuje, odkąd pamięta. Fan technologii mobilnych, OpenGL i Visual Studio. W wolnym czasie rozwija kolejny niedokończony silnik: <https://bitbucket.org/dynamicreflectance/multicolor>

  
MATEUSZ SEMEGEN  
Dynamic Reflectance  
mateusz.semegen@dynamicreflectance.com  
Duszą startupowiec, sercem freelancer, po godzinach perkusista. Programuje, odkąd pamięta. Fan technologii mobilnych, OpenGL i Visual Studio. W wolnym czasie rozwija kolejny niedokończony silnik: <https://bitbucket.org/dynamicreflectance/multicolor>

## PROGRAMOWANIE ROZWIĄZAŃ SERWEROWYCH

# Nginx – (nie)zwykły serwer WWW i nie tylko

Rosnący w ostatnich latach ruch w sieci zmusił programistów do opracowania nowych rozwiązań i architektur, które będą w stanie obsłużyć dziesiątki tysięcy zapytań na sekundę. W tym artykule postaram się przybliżyć możliwości, jakie daje jeden z najbardziej popularnych serwerów, który jako pierwszy przekroczył barierę 10 tysięcy zapytań na sekundę.

Podobnie jak w innych serwerach, plik index.html znajdzie się w katalogu `/var/www/html/`.

### NGINX + DJANGO

Aby uruchomić na naszym serwerze coś więcej, niż same statyczne strony w HTMLu, potrzebujemy dodatkowego narzędzia, które zajmie się obsługą naszej strony. Nginx zwykle pełni rolę pośrednika, który zajmuje się obsługą ruchu HTTP, a całą resztę (tj. wykonanie kodu PHP lub uruchomienie konkretnego frameworka) pozostawiamy innemu serwerowi, za pomocą której ukryjemy przed użytkownikiem całą maszynę ogromnych i wydajnych serwerów WWW.

Wchodząc na stronę takie jak Facebook, wykop.pl lub podobne, nie myślimy o tym, co dzieje się po stronie serwera. Strona się ładuje, i gotowe. W czasach, gdy strony zmieniały się co najwyżej kilka razy dziennie i były identyczne dla wszystkich użytkowników (wciąż jest takich dużo), rozproszenie ich w celu zwiększenia wydajności nie było problemem. Wystarczyło postawić *load balancer*, np. oparty o system DNS, i jakoś to działało. Wraz z nadaniem serwisów społecznościowych okazało się, że przygotowanie innej wersji strony dla każdego użytkownika z osobna jest nie lada wyzwaniem. Każdy przecież widzi co innego na swojej stronie głównej Facebooka lub inne znaleziska z Wykopu, w zależności od swoich preferencji. Obciążenie taką funkcjonalnością pojedynczego serwera raczej nie wchodzi już w grę.

### INSTALACJA

Zainstalowanie standardowej wersji nginx jest proste. W Ubuntu, Debianie lub Linux Mint za pomocą `apt-get` lub innego managera pakietów instalujemy odpowiednią paczkę:

```
sudo apt-get install nginx
```

Po chwili serwer powinien być gotowy i uruchomiony. Można to sprawdzić, łącząc się z adresem `http://localhost` w przeglądarce:



Rysunek 1. Domyślna strona serwera

Aby stworzyć naszą prostą stronę WWW opartą o Django, utworzymy nowy projekt, a w nim aplikację ze stroną:

```
django-admin startproject mywebsite
cd mywebsite
./manage.py startapp article
```

Dla uściślenia – projekt Django to cały serwis WWW, w skład którego wchodzą aplikacje – poszczególne funkcjonalności naszej strony. Powyższe polecenia utworzyły katalog `mywebsite` z całym projektem. W podkatalogu `mywebsite` znajdują się wszystkie ustawienia projektu (`settings.py`) oraz kilka innych plików, do których wrócimy później. Komenda `startapp` stworzyła jedną aplikację o nazwie `article`, do której dodamy prostą funkcję wyświetlającą „Hello World” w formie strony WWW.

Katalog `article` z nową aplikacją zawiera zwykłe następujące pliki:

- » `views.py` – skrypt z funkcjami obsługującymi poszczególne strony WWW, zwykłe mają one za zadanie przygotowanie danych z bazy oraz przedstawienie ich w odpowiedniej formie, z użyciem szablonów HTML.
- » `models.py` – skrypt z klasami opisującymi model danych dla

aplikacji, np. czym jest artykuł, kategoria itd. W naszym przykładzie to pominiemy.

- » urls.py – lista par: adres URI - funkcja z views.py obsługująca go. Pozwala odpowiednio połączyć strony z ich adresami i tworzy tzw. routing w całym serwisie. To z tych plików Django wie, jaką funkcję ma wygenerować zawartość strony pod odpowiednim adresem.

Oprócz tego cały projekt ma swoją główną listę URI<sup>1</sup> w pliku mywebsite/urls.py. Zwykle dodaje się tam odniesienia do poszczególnych plików urls.py w aplikacjach. Dodajemy do powyższego pliku import modułu article.urls oraz nowy wpis w liście urls:

**Listing 1. Główny plik urls.py opisujący, jak obsługiwać adresy zaczynające się od artykułu/**

```
from django.conf.urls import url, include
from django.contrib import admin
from django.urls import path, include, re_path
from . import views
from .views import ArticleList, ArticleShow
from .models import Article

urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', include('article.urls')),
    path('articles//', include('article.urls')),
]
```

Następnie przejdźmy do naszej właściwej aplikacji Django w katalogu article/. W pliku views.py znajdziemy domyślnie jedną linię z importem podstawowych funkcji oraz komentarz. Dodajmy do tego pliku następującą zawartość:

**Listing 2. Prosta funkcja obsługująca listę artykułów**

```
from django.shortcuts import render
from django.shortcuts import HttpResponse
from django.shortcuts import redirect
from django.shortcuts import reverse
from django.shortcuts import get_object_or_404
from .models import Article
from .forms import ArticleForm

def list_articles(request):
    return HttpResponse('Hello World!')  
def show_article(request, name):
    article = get_object_or_404(Article, name=name)
    return HttpResponse(f'Lorem ipsum dolor sit amet, {name} nomen vnde  
consecetur... + name)
```

Na razie wystarczą nam dwie proste funkcje: list\_articles i show\_article, które zwrócą proste odpowiedzi. Można to oczywiście rozbudować o model danych i obsługę artykułów pobieranych z bazy, ale to temat na osobny artykuł o Django. Dla nauki podstaw konfiguracji nginx wystarczy nam taka „zaślepka”. Warto zauważać w tym miejscu, że funkcja show\_article przyjmuje parametry nam – czyli teoretyczną nazwę artykułu. Jego obsługą zajmiemy się za chwilę w routingu. W kolejnym kroku musimy dodać listę urls.py dla naszej aplikacji, która utworzy odpowiedni routing do poszczególnych funkcji z views. W przeciwieństwie do głównego pliku mywebsite/urls.py, możemy teraz wtrzcić wszystko, co ma być podstroną konkretnej aplikacji – URI zaczynają się od artykułu/ (jak określiliśmy w mywebsite/urls.py). Stwórzmy plik article/urls.py z zawartością:

**Listing 3. Plik article/urls.py z definicją routingu URI dla aplikacji article**

```
from django.conf.urls import url, include
from . import views
from .views import ArticleList, ArticleShow
from .models import Article

urlpatterns = [
    url(r'^$', ArticleList.as_view(), name='article_list'),
    url(r'^(?P<name>[a-zA-Z0-9-_\.]+)$', ArticleShow.as_view(), name='article_show'),
]
```

1. URI – ang. Uniform Resource Identifier – czyli strona, która identyfikuje poszczególne strony na serwerze.

W powyższym kodzie mamy określone dwa adresy. Pierwszy, pusty (określony wyrażeniem regularnym ^\$) będzie kierował do funkcji wyświetlającej listę artykułów. Finalnie, po złożeniu z głównym routingu projektu, będzie to http://mojastrona.pl/article. Drugi wpis określa ścieżkę do konkretnego artykułu, czyli np. http://mojastrona.pl/article/jak-zrobic-bigos. Fragment (?P<name>[a-zA-Z0-9-\_\.]+)\$ powinien wypływać wszystkie stringi składające się z małych i dużych liter, cyfr, kreski, kropki lub podkreslenia. Rozwijając dalej ten projekt, moglibyśmy położyć się o wyciągnięcie z bazy danych artykułu o takiej nazwie. Takie dość dokładne określenie w URI, z którego ma się składać nazwa, może nas po części ochronić przed atakiem typu SQL Injection. Poza tym Django ma jeszcze bibliotekę ORM do komunikacji z bazą, która również powinna zabezpieczyć przed tym typu błędami, ale o wadliwości danych warto pamiętać już na samym początku ich przetwarzania, do czego jesteśmy poniekąd zmuszeni.

W Nasz projekt powinien być już gotowy. Można go uruchomić, wpisując polecenie: `python manage.py runserver` lub `cdg mysite; ./runserver`. Po uruchomieniu serwera, otrzymujemy informację, że serwer działa pod adresem 127.0.0.1:8000. Po skopiowaniu tego adresu do schowka i wpisaniu go do przeglądarki, powinniśmy zobaczyć naszą stronę.

**Listing 4. Uruchamianie testowego serwera Django**

```
You have unapplied migrations; your app may not work properly
until they are applied.
Run 'python manage.py migrate' to apply them.
```

February 02, 2017 - 17:20:19

Django version 1.9, using settings 'mywebsite.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CONTROL-C.

## NGINX JAKO REVERSE PROXY

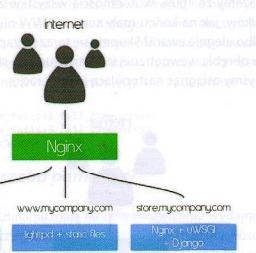
Zwykły jeden serwer WWW obsługuje więcej niż jedna stronę WWW. W tym celu wpisy DNS ustawia się na ten sam adres IP serwera WWW, a on sam, na podstawie nagłówków HTTP, decyduje, jaką stronę ma wyświetlić. Podobnie można skonfigurować serwer nginx, aby obsługiwał wiele serwisów na raz.

### sites-available i sites-enabled

W poprzedniej części artykułu zobaczyliśmy, jak skonfigurować pojedyńczy serwis, używając Django i uWSGI jako backendu. W tym celu dodaliśmy nowy plik do katalogów `/etc/nginx/sites-available` oraz link symboliczny w `/etc/nginx/sites-enabled`. Aby obsługiwać kilka domen, wystarczy utworzyć kilka wpisów w powyższych katalogach i ustawić im odpowiednio parametry `server_name`.

Załóżmy, że główna domena naszej firmy to `mycompany.com` i jest stworzona w formie kilku statycznych plików HTML. Dodatkowo mamy stronę `docs.mycompany.com`, która udostępnia dokumentację stworzoną w oparciu o WordPress i dedykowany skrypt dla sklepu internetowego, pod adresem `store.mycompany.com`, napisany z użyciem Django.

Aby zwiększyć bezpieczeństwo całego serwisu i odporność na awarie, możemy rozdzielić każdy z tych trzech serwisów. W tym celu obsługę stron `docs`, `store` i głównej powierzymy temu różnym masynom. Mogą to być fizyczne serwery, wirtualne maszyny lub nawet instancje w chmurze. Zwykle aby obsługiwać taką konfigurację, wystarczy skonfigurować odpowiednio DNS i wystawić na świat te trzy maszyny przez publiczne IP. Nie zawsze jednak mamy taką możliwość, a czasami nie chcemy wystawiać całych serwisów, które muszą mieć uruchomione też inne usługi w lokalnej sieci. Mamy spróbować skonfigurować dodatkową instancję nginx, która zajmie się przekierowywaniem ruchu do naszych trzech maszyn:



Rysunek 3. Przykładowa konfiguracja nginx jako reverse proxy

Popatrzmy teraz na konfigurację serwera nginx (zielony prostokąt). Potrzebuje on trzy pliki z definicjami vhostów. Pierwsza z nich to główna strona WWW:

### Listing 7. Konfiguracja vhosta nginx dla głównej strony WWW

```
server {
    listen 80;
    listen [::]:80;
    server_name mycompany.com www.mycompany.com;
    ...
```

```
location / {
    include proxy_params;
    proxy_pass http://10.100.0.1:80;
}

location / {
    include proxy_params;
    proxy_pass http://10.100.0.2:80;
}
```

```
location / {
    include proxy_params;
    proxy_pass http://10.100.0.3:80;
}
```

Listing 8. Konfiguracja vhosta nginx dla głównej strony WWW

```
server {
    listen 80;
    listen [::]:80;
    server_name docs.mycompany.com;
    ...
```

```
location / {
    include proxy_params;
    proxy_pass http://10.100.0.1:80;
}
```

```
}
```

Powyższe konfiguracje należy zapisać w osobnych plikach w katalogu `/etc/nginx/sites-available` i utworzyć do nich odpowiednie linki w `/etc/nginx/sites-enabled`.

Po odpowiednim skonfigurowaniu nginxa pozostało uruchomić odpowiednie usługi w naszej lokalnej sieci z poszczególnymi stronami WWW, do których będą miały dołączyć domeny naszych vhostów. W tym celu wystarczy zmienić porty w konfiguracjach

## Różne domeny – różne technologie

Każdy z serwerów, do których przekierowaliśmy ruch z poszczególnych domen, może działać w oparciu o zupełnie różne technologie. Dla nginx, jako reverse proxy, nie ma w tym przypadku znaczenia, czy na końcu dostanie odpowiedź ze skryptu PHP, Ruby on Rails czy jeszcze czegoś innego. W przypadku konfiguracji proxy możemy dowolnie manipułować tym ruchem i dosłownie poskładać w naszym serwerze różne, często nawet wykluczające się technologie.

Zwykle w takim przypadku dobrym rozwiązaniem jest obsługa poszczególnych domen w osobnych kontenerach (np. przez Docker'a) lub wirtualnych maszynach. Nginx w takiej konfiguracji pełni rolę wejściowej bramy, która kieruje ruch dalej.

## proxy\_pass

W tym miejscu zaczynają być widoczne zalety tego, co daje nam nginx. Dodając w pliku konfiguracyjnym dyrektywę `proxy_pass`, możemy przekazać zapytania do obsługi dalej, przez inne serwery WWW w naszej infrastrukturze. Odrodzimy w tym miejscu od problemów związanych ze zmianą i propagacją wpisów w DNS. W przypadku potrzeby przełączenia subdomeny na inny serwer wystarczy tylko przełączać serwer nginx i efekt będzie natychmiastowy. Gdy okaże się, że coś nie działa poprawnie na nowym serwerze – w dowolnym momencie możemy wrócić na pierwszy serwer (któro robi takie rzeczy na instalacji produkcyjnej – niech się nie przyzna :)). Ciągle jednak nasze serwisy nie są odporne na awarie poszczególnych fragmentów infrastruktury.

## Podział zadań między serwery

Obecnie większość aplikacji internetowych działa na jednym serwerze.

Innym rozwiązaniem, które można zastosować równolegle do opisanego wcześniej reverse proxy, jest podzielenie funkcjonalności serwera na wiele mniejszych modułów. Obsługa poszczególnych URI możemy przekazać do różnych aplikacji uruchomionych na wielu maszynach lub do różnych aplikacji. Na przykład opisowane w powyższym przykładzie mamy ustaloną domenę głównej strony dla naszej firmy oraz dodatkową obsługę IPv6. Dla powyższej konfiguracji wszystkie zapytania przychodzące do danej domeny będą przekierowywane (`proxy_pass`) do serwera pod adresem IP 10.100.0.1, na port 80 (jeden z niebieskich prostokątów).

konfiguracji kolejnych vhostów będzie wyglądała podobnie.

Zmieniać się będą adresy IP dla parametru `proxy_pass` oraz nazwy obsługiwanych domen (parametr `server_name`). Obsługa vhosta dla dokumentacji może wyglądać następująco:

`server_name 10.100.0.2; proxy_pass http://10.100.0.1:80;`

W powyższej konfiguracji duplikujemy reverse proxy z poprzedniego przykładu. W ten sposób będzie zapewniona duplikacja tego zasobu, który jest krytyczny w całej infrastrukturze.

To, które reverse proxy będzie wybierane, zależy już od celu, jaka chcemy osiągnąć. Jeżeli musimy zapewnić ciągłość usługi i przetrwać wszelkie awarie, to dobrym rozwiązaniem będzie tutaj pływające IP – floating IP. Jest to mechanizm dostarczany m.in. przez Pacemaker, który pozwala w razie awarii jednego serwera na przeniesienie jego publicznego IP do drugiego. Przy dobrej konfiguracji całość powinna trwać ulamek sekundy.

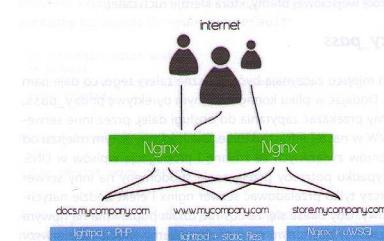
Jeśli natomiast chcemy, aby zwiększyć wydajność naszego serwisu i zsumować wydajność dwóch lub więcej reverse proxy, to możemy poszukiwać o stworzeniu *load balancing* opartego o system DNS. Potrzebujemy do tego dedykowany serwer, w którym ustawimy relatywny krótki czas wygaśnięcia wpisu naszej domeny w DNS. Druga ważna rzecz, to aby serwer DNS za każdym razem zwrać losowe IP naszych reverse proxy. W ten sposób część klientów serwusu będzie łączyła się z jednym reverse proxy, a część z drugim. Dzięki temu możemy też postawić je w odległych od siebie data center i przy lekkiej modyfikacji powyższej metody stworzyć geograficzny *load balancing* oparty o źródłowy kraj klientów. W tym przypadku awaria jednego z reverse proxy może być nieco bardziej dotkliwa. O ile narzędzia typu Pacemaker potrafią wykryć problem w ulamku sekundy, to wygaśnięcie wpisów w DNS może potrwać dłużej.

Innym rozwiązaniem może też być wykorzystanie zewnętrznych serwisów typu CloudFlare, które zapewniają zewnętrzne przekierowanie ruchu w przypadku awarii naszego serwera. Same w sobie pełnią już role reverse proxy.

Dodatkowa sekcja `location` mówi nginxowi, aby wszystkie URI zaczynające się od `/static` przekierować do osobnego serwera WWW. W ten sposób możemy odciągnąć serwer obsługujący generowanie stron i przenieść część ruchu na serwer, którymu zapewniemy lepsze I/O w celu szybszego dostarczania treści.

## HA I REVERSE PROXY DO WIELU SERWERÓW

Powyższe konfiguracje dalej posiadają pojedyncze punkty, których awaria może skutkować niedostępnością całego serwusia. Wspomnialiśmy wcześniej, że nginx może być bramą do wielu różnych serwisów, które zwykle można by obsługiwać przez różne adresy IP przypisane do subdomen. Pójdzmy o krok dalej i stwórzmy konfigurację, w której system DNS zagwarantuje nam *load balancing* pomiędzy wieloma reverse proxy w naszej infrastrukturze.



Rysunek 4. Architektura serwisów z redundantnym reverse proxy

W powyższej konfiguracji duplikujemy reverse proxy z poprzedniego przykładu. W ten sposób będzie zapewniona duplikacja tego zasobu, który jest krytyczny w całej infrastrukturze.

To, które reverse proxy będzie wybierane, zależy już od celu, jaka chcemy osiągnąć. Jeżeli musimy zapewnić ciągłość usługi i przetrwać wszelkie awarie, to dobrym rozwiązaniem będzie tutaj pływające IP – floating IP. Jest to mechanizm dostarczany m.in. przez Pacemaker, który pozwala w razie awarii jednego serwera na przeniesienie jego publicznego IP do drugiego. Przy dobrej konfiguracji całość powinna trwać ulamek sekundy.

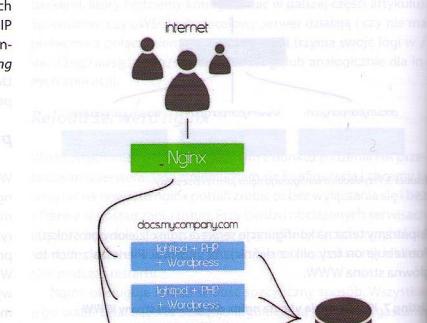
Jeśli natomiast chcemy, aby zwiększyć wydajność naszego serwisu i zsumować wydajność dwóch lub więcej reverse proxy, to możemy poszukiwać o stworzeniu *load balancing* opartego o system DNS. Potrzebujemy do tego dedykowany serwer, w którym ustawimy relatywny krótki czas wygaśnięcia wpisu naszej domeny w DNS. Druga ważna rzecz, to aby serwer DNS za każdym razem zwrać losowe IP naszych reverse proxy. W ten sposób część klientów serwusu będzie łączyła się z jednym reverse proxy, a część z drugim. Dzięki temu możemy też postawić je w odległych od siebie data center i przy lekkiej modyfikacji powyższej metody stworzyć geograficzny *load balancing* oparty o źródłowy kraj klientów. W tym przypadku awaria jednego z reverse proxy może być nieco bardziej dotkliwa. O ile narzędzia typu Pacemaker potrafią wykryć problem w ulamku sekundy, to wygaśnięcie wpisów w DNS może potrwać dłużej.

Innym rozwiązaniem może też być wykorzystanie zewnętrznych serwisów typu CloudFlare, które zapewniają zewnętrzne przekierowanie ruchu w przypadku awarii naszego serwera. Same w sobie pełnią już role reverse proxy.

## Upstream

W powyższej konfiguracji serwera nginx nie ma możliwości wykorzystania zewnętrznych serwisów typu CloudFlare, ponieważ nie ma dedykowanego serwera, który byłby odpowiedzialny za przekierowanie ruchu do naszych reverse proxy. W tym celu możemy skorzystać z funkcji `upstream`.

Wiem już, czego użyć, aby wejście do naszej infrastruktury było możliwe wydajne i odporne na awarie. Pozostaje jeszcze kwestia dostępności serwisów w naszej infrastrukturze. Co z tego, że przekazujemy ze 100% skutecznością wszystkie zapytania od użytkowników, jak na końcu mały serwer WWW nie da rady ich obsługiwać albo ulegnie awarii? Skupmy się teraz na zapewnieniu dostępności w obrębie wewnętrznej infrastruktury, po stronie serwisu. Chcielibyśmy osiągnąć następującą konfigurację:



Rysunek 5. Redundantne serwisy WWW, z których korzysta reverse proxy

Nginx po otrzymaniu każdego zapytania od klienta powinien wybrać jeden z serwerów zdefiniowanych w konfiguracji. Dodatkowo chcemy, aby w razie awarii któregoś z nich został on wykluczony z tej płyty.

Nginx pozwala zdefiniować pulę adresów, do których ma przekazać ruch, tzw. upstream. Zmodyfikujmy konfigurację jednego z hostów z poprzedniego przykładu:

**Listing 10. Konfiguracja vhosta dla redundantnych docelowych serwerów**

```
upstream store_backend {
    server 10.200.0.1;
    server 10.200.0.2;
}

server {
    listen 80;
    listen ::1:80;
    server_name store.mycompany.com;
    location / {
        proxy_pass http://store_backend;
    }
}
```

Powyższa konfiguracja pozwala wykorzystać więcej niż jeden docelowy serwer. W sekcji `upstream` możemy zdefiniować listę serwerów, do których nginx ma przesłać zapytania. Pozwoli to na równomiernie rozłożenie obciążenia. Jeżeli mamy taką potrzebę, to istnieje też możliwość nierównomiernego obciążenia zapytaniami, przez dodanie wag do każdego z wpisów:

```
server 10.200.0.1 weight=1;
server 10.200.0.2 weight=2;
```

W ten sposób serwer o adresie 10.200.0.1 dostanie dwa razy mniej zapytań niż drugi. Powyższa konfiguracja nie daje jednak gwarancji, że w przypadku awarii jednego z docelowych serwerów zostanie wykluczony z płyty, aby nie dawać użytkownikom błędnych odpowiedzi. Aby to zrobić, trzeba dodatkowo określić, ile błędnych zapytań może zostać przesłanych do danego serwera oraz po jakim czasie oczekiwania uznać się, że serwer nie działa poprawnie.

Mogliśmy dodać też pulę zapasowych hostów do sekcji `upstream`, tak aby zostały włączone w miejsce usuniętych, w razie awarii:

```
server 10.200.0.1 max_fails=3 fail_timeout=5s;
server 10.200.0.2 max_fails=3 fail_timeout=5s;
server 10.200.0.3 backup; ..
```

### O czym jeszcze trzeba pamiętać?

Wszystkie powyższe działania mają na celu zwiększenie wydajności i dostępności poszczególnych elementów infrastruktury. Pozostaje jeszcze jedna z najmniej przyjemnych i skalowalnych kwestii – baz danych i storage. Warto o tym pamiętać od samego początku, projektując cały system. O ile samą logikę serwisów WWW moż-

na zwykle prosto zduplikować, to duplikacja bazy danych jest już nieco bardziej skomplikowanym zadaniem, aczkolwiek też możliwe. Przy skali, z jaką spotykają się portale społecznościowe, może się również okazać, że zwykłe podejście do baz danych i wykorzystanie SQL-owych może być ślepką uliczką.

Dynamyczna (re)konfiguracja

Czasami proste przekazanie zapytań do losowych serwerów w naszej infrastrukturze może nie wystarczyć. Trzeba wtedy zrobić coś mądrzejszego i na przykład przekazać ruch na podstawie nazwy (lub ID) użytkownika do odpowiedniego serwera wewnętrznej infrastruktury, na podstawie URI. Założymy, że potrzebujemy przekierować użytkowników na literę „a” do serwera 10.0.0.1, na literę „b” do serwera 10.0.0.2 i tak dalej. Można by to wpisać na sztywno do pliku konfiguracyjnego vhosta, jednak ciężko by było to zmodyfikować na wielu komputerach na raz. Nginx pozwala w łatwy sposób wykorzystać skrypty LUA do wygenerowania wartości zmiennej w jego plikach konfiguracyjnych. A stąd już tylko kilka kroków do zautomatyzowania jego konfiguracji ułatwienia sobie życia (placząc za malym spadkiem wydajności...).

Dostosujemy do nginx moduł `extras`, który dostarcza wsparcie dla skryptów LUA:

`sudo apt-get install nginx-extras`

### Listing 11. Definicja vhosta działająca w oparciu o zewnętrzny skrypt

```
server {
    listen 80;
    server_name wypok.pl;
    location ^~ /user/{a-zA-Z0-9}+/$ {
        set_by_lua $backend
        "local f = assert(io.open(\"wybierz_backend.sh\" .. ngx.arg[1],
        \"r\"))";
        local selected_backend = assert(f:read(\"*a\""));
        return selected_backend
    }
    proxy_pass $backend;
}
```

Warto przyjrzeć się sekcji `location`, której do tej pory używaliśmy do określenia tego, które strony mają być obsługiwane według zadanego kryterium, na przykład przez taki wpis:

```
location /static {
```

W powyższym przykładzie wykorzystaliśmy wyrażenie regularne, aby „zlapać” wszystkie żądania do serwera zaczynające się od `/user`, które zawierają nazwę użytkownika składającą się z liter i cyfr. Dodatkowo taka konstrukcja powoduje, że dopasowana gru-

## PROGRAMOWANIE ROZWIĄZAŃ SERWEROWYCH

pa wyrażenia regularnego zostanie zapisana w sekcji w zmiennej `$1` (i analogicznie kolejne numery dla kolejnych grup wyrażenia). Na przykład URI `/user/bob/` będzie pasował do podanego wzorca i zmiennej `$1` powinna przyjąć wartość `bob`.

Dyrektiva `set_by_lua` mówi, aby nginx określił zawartość zmiennej po wykonaniu skryptu podanego jako drugi parametr (przejdzieliśmy do niego na chwilę). Składnia jest następująca:

```
set_by_lua nazwa_zmiennej "skrypt" parametr1 parametr2 ;
```

```
location /article {
    set_by_lua $memcached_key "$uri$args";
    memcached_pass 127.0.0.1:11211;
    error_page 404 502 504 @fallback;
}
```

```
location @fallback {
    proxy_pass http://10.200.0.1;
}
```

W definicji vhosta jest to rozłożone na kilka linijek. `Set_by_lua` ustala zmiennej `backend` na podstawie wyniku następującego skryptu:

### Listing 12. Skrypt LUA wykorzystany do określenia docelowego backenda

```
local f = assert(io.open("wybierz_backend.sh" .. ngx.arg[1],
"r"))
local selected_backend = assert(f:read("*a"));
return selected_backend
```

Na koniec wywołano `set_by_lua` podajemy jeszcze nazwę użytkownika przechwyconą przez wyrażenie regularne w zmiennej `$1`.

Co powyższy skrypt robi? W dużym skrócie: uruchamia przez `open` skrypt `wybierz_backend.sh` i służy podaje pierwszy parametr skryptu (`ngx.arg[1]`). Jest to parametr całego skryptu LUA. Następnie do zmiennej `selected_backend` jest zapisywane wyjście programu `wybierz_backend.sh`, który powinien w magickim sposób wybrać dla nginxa docelowy serwer, do którego ma zostać przesłane zapytanie od klienta.

Wybrane docelowe backendu na podstawie nazwy użytkownika może być wykonanie nieco prościej, jednak jest łatwe do pokazania tego, jakie możliwości daje nginx. Czasem takie rozwiązanie może być potrzebne w znacznie bardziej skomplikowanym środowisku, w którym nie da się zastąpić wszystkiego sztywno wygenerowanymi plikami konfiguracyjnymi. Przykładem zastosowania tego rozwiązania może być obsługa `CloudInit` w chmurze `CloudOver`, gdzie w zależności od zródłowego adresu IP wirtualnej maszyny potrzeba było zwrócić różną zawartość pliku `CloudConfig`. Z powodu różnych zawiłości sieciowych i bezpieczeństwa nie było możliwości sprawdzenia zdalnego adresu po stronie docelowego serwera udostępniającego skrypty. Niezbędne było wykonanie takiego proxy na każdym compute node w obrębie klastra.

### CACHE

Ostatnim sposobem na polepszenie wydajności serwisu jest wykorzystanie cache wszędzie tam, gdzie to możliwe. Do tymczasowego przechowywania po stronie reverse proxy nadają się wszelkie strony, które zmieniają się rzadko i nie ma potrzeby generować ich za każdym razem. Dobrym przykładem będą wszelkiego rodzaju

newsy, statyczne strony z kontektem `ip`, które są renderowane na serwerze, a zmieniają się bardzo rzadko. Nie powinno się natomiast cacheować stron generowanych dynamicznie, na podstawie wczorza i zmiennej `$1` powinna przyjąć wartość `bob`.

Cache po stronie reverse proxy pozwala odciążyć skrypty generujące strony WWW i wyświetlać ich kopię przygotowaną wcześniej. Dzięki temu serwer może zająć się innymi rzecznymi, a przygotowane raz strony są bardzo szybko zwarcane z kopią trzymanej w pamięci RAM.

Popatrzmy na przykładową definicję vhosta z dokumentacji nginx:

### Listing 13. Konfiguracja vhosta wykorzystująca memcached

```
server {
    ...
    location /article {
        set_by_lua $memcached_key "$uri$args";
        memcached_pass 127.0.0.1:11211;
        error_page 404 502 504 @fallback;
    }
    location @fallback {
        proxy_pass http://10.200.0.1;
    }
}
```

Wszystkie strony zaczynające się od `/article` powinny zostać w pierwszej kolejności wczytane z serwera memcached działającego na lokalnej maszynie. Nginx będzie szukał gotowych stron w cache pod kluczem określonym w zmiennej `memcached_key`. Jeżeli nie zostanie ona tam znaleziona, wtedy nginx przesyła zapytanie standardową drogą, do docelowego serwera pod adresem `10.200.0.1`.

Powyższy moduł wymaga od programisty, aby samodzielnie zapisywał gotowe strony do cache. Niestety nie ma możliwości, aby nginx sam pobierał zapytanie i zapisał je w pamięci.

## SKŁAD BRAĆ INFRASTRUKTURĘ?

Powyższe rozwiązania mogą wymagać nawet kilkunastu maszyn do uruchomienia w pełni redundantnego serwisu WWW, który będzie się łatwo skalować. Skąd zatem wziąć tyle sprzętu? Jednym z rozwiązań jest wynajem wirtualnych maszyn w publicznej chmurze takiej jak Amazon EC2 lub Rackspace, gdzie będziemy mogli pomieścić dowolną ilość zasobów, bez obaw o dobre łączne i stabilne zasilanie. Inną możliwością, nieco bardziej budżetową, jest uruchomienie własnej prywatnej chmury, takiej jak CoreCluster, w której będziemy mogli dowolnie zarządzać zasobami już przy kilku komputerach wchodzących w skład klastra. Któregokolwiek z powyższych rozwiązań nie wybieramy, to na pewno będzie miało to znaczną przewagę nad ręcznym konfigurowaniem serwerów lub stawianiem wirtualnych maszyn na własną rękę. W środowisku chmury nie będziemy musieli zapraszać sobie głowy konfiguracją sieci oraz zapewnieniem odpowiednich zasobów na poszczególnych maszynach.

### MACIEJ NABOŻNY

maciej.nabozny@cloudover.io

Programista i administrator systemów rozproszonych. Od ponad siedmiu lat zajmuje się projektowaniem chmur obliczeniowych laaS, począwszy od małych instalacji, aż po największe klastry obliczeniowe w Polsce. Od 2014 roku rozwija projekt otwartej chmury obliczeniowej CloudOver.org, który ma wyjaśnić naprzeciw potrzebom małych i średnich instalacji, dla których OpenStack to zbyt duzo. Od czasu do czasu organizuje również warsztaty z różnych tematów okołochmurowych. Autora można znaleźć przez grupę KrakCloud na meetup.com.

Przedmiotem publikacji jest opisowanie technologii, narzędzi i procesów, które pozwolą na tworzenie i zarządzanie chmurami obliczeniowymi.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje publikacje o tematyce IT, programistycznej i zarządzania.

Wydawnictwo "Programista" i "Programista Magazyn" wydaje

# Pora spać!

## Zarządzanie energią w ARMv8

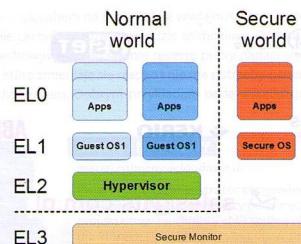
Najnowsza generacja 64-bitowych procesorów ARM z powodzeniem wykorzystywana jest w różnych segmentach rynku nowoczesnej elektroniki. Urządzenia mobilne, coraz śmielsze próby wejścia w zdominowany przez Intel świat serwerów czy czekająca nas nowa rzeczywistość utkana „inteligentnymi” urządzeniami tzw. Internetu Rzeczy (IoT – Internet of Things) – oto przykłady, gdzie ARMv8 buduje swoją pozycję. Jednym z głównych powodów powodzenia tego typu procesorów jest połączenie wysokiej wydajności, niskiego poboru energii i zunifikowanych metod umożliwiających efektywne zarządzanie jej zużyciem. Przekonajmy się, jak wygląda to od strony systemu operacyjnego.

### ARM TRUSTED FIRMWARE

Aby zrozumieć zawiłości implementacyjne zarządzania energią, niezbędnym jest uświadomienie, w jakim środowisku działa system operacyjny (w tym przykładzie Linux) na procesorach ARMv8. Otóż po uruchomieniu OS-u może nie być on jedynym tematem, który sprawuje kontrolę nad maszyną. Konstruktory 64-bitowego ARMA przewidzieli możliwość jego pracy na wielu powiązanych ze sobą tzw. poziomach wyjątków (*Exception Levels* – ELx, gdzie „x” oznacza identyfikator poziomu od 0 do 3). Są one od siebie odseparowane sprzętowo i posiadają własne zestawy rejestrów procesora, m.in. konfiguracyjnych oraz takich jak wskaźnik ramki stossu (SP\_ELx), licznik programu (PC\_ELx). Warto nadmienić, że poziomy większe od 0 mogą korzystać zamiennie z własnego SP\_ELx, bądź z SP\_EL0 jako tzw. *runtime stack pointer*. Przelatanie pomiędzy nimi możliwe jest poprzez ustawienie pola SPSEL w rejestrze SPSR\_ELx.

### Dwa światy

Nowa architektura definiuje tzw. świat „normalny” (Normal world) bądź Non-secure World) oraz świat „zaufany” (Trusted lub Secure World) – dla każdego z nich przewidziany jest określony typ oprogramowania.



Rysunek 1. Stos programowy architektury ARMv8

- » BL1 – pełniejsza konfiguracja sprzętu (nadpisuje częściowo ustawienia z BL1), np. przerwarki, przygotowanie środowiska obsługi PSCI, inicjalizacja kontekstów programowych per-CPU (struktur przechowujących dane o stanie), aby umożliwić współbieżne działanie;
- » BL33 – działający w EL1 lub EL2 „normalnego” świata bootloader, jak U-Boot czy UEFI, lub aplikacja testująca.

Producenti mogą wprowadzić własne modyfikacje do sekwenacji uruchomieniowej. Na przykład obraz do bootowania procesorów z rodziną Armada 7k/8k firmy Marvell posiada dodatkowy FIP plik z programem konfigurującym pamięć DRAM, wraz z metadanymi. Całość przechowywana jest na nośniku, np. SPI flash. Kod wbudowany, tzw. *bootrom*, po włączeniu lub resecie odczytuje metadane i jeśli są poprawne, laduje pierwszą część kodu do pamięci SRAM. Po zakończeniu inicjalizacji pamięci zewnętrznej pozostałe pliki binarne są w niej umieszczane, a następnie stamtąd kolejno wykonywane. Znajomość tej sekwenacji okaza się przydatna w jednym z rozważanych przypadków obsługi usypymania systemu.

### PSCI

PSCI to domyślny interfejs do zarządzania energią poszczególnych rdzeni w architekturze ARMv8. Został stworzony, aby zapewnić ustandaryzowany, wspólny mechanizm dla softwaru działającego na różnych poziomach wyjątków (także w środowisku virtualizacji), co niezwykle ułatwia integrację oprogramowania systemu. PSCI uwzględnia zależności pomiędzy „normalnym” a „zaufanym” światem. Rolą firmware'u EL3 jest natomiast arbitraż wywołań i zapewnienie wykonania poszczególnych metod, takich m.in. jak:

- » CPU\_SUSPEND
- » SYSTEM\_SUSPEND
- » CPU\_ON
- » CPU\_OFF
- » SYSTEM\_RESET
- » SYSTEM\_POWERDOWN
- » SYSTEM\_OFF
- » SYSTEM\_RESET

Powyższe operacje wspierane są w generycznym kodzie ATF, jednak wymagają również implementacji platformowych funkcji zwrotowych definiowanych w strukturze *plat\_psci\_ops\_t*. Poniżej znajduje się przykład *callbacków* dla procesora Armada 3720 firmy Marvell.

### Listing 1. Platformowe funkcje zwrotne PSCI dla procesora Armada 3720

```

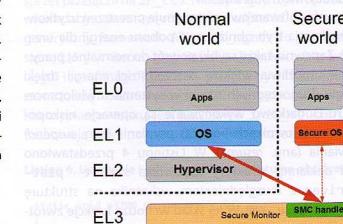
const plat_psci_ops_t plat_arm_psci_pm_ops = {
    .cpu_standby = a3700_cpu_standby,
    .pwr_domain_on = a3700_pwr_domain_on,
    .pwr_domain_off = a3700_pwr_domain_off,
    .pwr_domain_suspend = a3700_pwr_domain_suspend,
    .pwr_domain_on_finish = a3700_pwr_domain_on_finish,
    .pwr_domain_suspend_finish = a3700_pwr_domain_suspend_finish,
    .get_sys_suspend_power_state = a3700_get_sys_suspend_power_state,
    .system_on = a3700_system_on,
    .system_off = a3700_system_off,
    .system_reset = a3700_system_reset,
    .validate_power_state = a3700_validate_power_state,
    .validate_ns_entrypoint = a3700_validate_ns_entrypoint
};

plat_psci_ops_t plat_arm_psci_pm_ops;
  
```

Funkcje są rejestrowane podczas inicjalizacji BL31 w funkcji *plat\_setup\_psci\_ops()*.

### SMC

SMC, czyli *Secure Monitor Call*, to typ funkcji ARMv8 używanej do generowania synchronicznych wyjątków obsługiwanych w firmware poziomu EL3. Jej argumenty i informacje zwrotne przekazywane są przez rejesty Xn. Opcjonalnie *Secure Monitor* może przekazywać wyniki również do świata „zaufanego”. Przykład przejścia pomiędzy poziomami wyjątków wskutek wykonania instrukcji SMC przedstawiony jest na Rysunku 2.



Rysunek 2. Schemat wywołania SMC

Zastosowanie wywoływa SMC można znaleźć w Linuksie, w sterowniku do obsługi PSCI (*drivers/firmware/psci.c*). W Listingu 2 przedstawiono metodę *CPU\_ON*, której zadaniem jest włączenie rdzeni w systemie wieloprocesorowym. Argumentami są ID procesora oraz adres, do którego ma on skoczyć po uruchomieniu i przejściu do EL1. Dalej, wraz z unikalnym numerem funkcji (fn), przekazywane są one bezpośrednio do SMC.

### Listing 2. Przykład wywołania SMC w Linuksie – CPU\_ON

```

static int psci_cpu_on(unsigned long cpuid, unsigned long entry_point)
{
    int err;
    u32 fn;

    fn = psci_function_id[PSCI_FN_CPU_ON];
    err = Invoke_psci_fn(fn, cpuid, entry_point, 0);
    return psci_to_linux(err);
}
  
```

Po stronie ATF, wskutek wyjątku, wykonywana jest funkcja *psci\_smc\_handler()*, której zadaniem jest odczytanie numeru funkcji i zawołanie odpowiedniej obsługi w poziomie EL3 w wykorzystaniu makra pomocniczego *SMC\_RET1* (Listing 3).

### Listing 3. Przykład obsługi SMC w ATF - CPU\_ON

```

uint64_t psci_smc_handler(uint32_t smc_fid,
                          uint64_t x1,
                          uint64_t x2,
                          uint64_t x3,
                          uint64_t x4,
                          void *cookie,
                          void *handle,
                          void *err,
                          uint64_t flags)
{
    switch (smc_fid) {
        case PSCI_CPU_ON_AARCH64:
            SMC_RET1(handle, psci_cpu_on(x1, x2, x3));
    }
}
  
```

```

[...]
    case PSCI_SYSTEM_SUSPEND_AARCH64:
        SMC_RET(handle, psci_system_suspend(x1, x2));
    [...]
    [...]
    } // end of handle->ops->system_suspend()
} // end of handle->ops->system_suspend()

static int psci_system_suspend_enter(suspend_state_t state)
{
    return cpu_suspend(0, psci_system_suspend);
}

static const struct platform_suspend_ops psci_suspend_ops = {
    .valid      = suspend_valid_only_mem,
    .enter     = psci_system_suspend_enter,
};

static void __init psci_init_system_suspend(void)
{
    int ret;
    if (!IS_ENABLED(CONFIG_SUSPEND))
        return;

    ret = psci_features(PSCI_FN_NATIVE(1_0, SYSTEM_SUSPEND));
    if (ret != PSCI_RET_NOT_SUPPORTED)
        suspend_set_ops(&psci_suspend_ops);
}

```

## STANY UŚPIENIA W LINUKSIE

System Linux umożliwia korzystanie z czterech poziomów uśpienia: *freeze*, *standby*, *mem* oraz *disk*:

- » *freeze* – czysto softwareowy, wstrzymuje przestrzeń użytkownika i uruchamia tryb obniżonego poboru energii dla urządzeń I/O. Zapewnia także szybki powrót do normalnej pracy.
- » *standby* – umożliwia większe oszczędności energii dzięki wyłączaniu drugorzędnych rdzeni w systemach wieloprocesorowych. Dodatkowo wykonywane są operacje niskopoziomowe w sterownikach podczas uśpiania (ang. *suspend*) i wznowiania (ang. *resume*). W Listingu 4 przedstawiono przykład deklaracji struktury sterownika struct *platform\_driver* z uwzględnieniem wskaźnika na strukturę dev\_pm\_ops (powyżej wskaznika na rejestrację funkcji *dev\_pm\_ops*).

### Listing 4. Deklaracja struktury sterownika sdhci\_pxav3.c

```

static const struct dev_pm_ops sdhci_pxav3_pmops = {
    SET_SYSTEM_SLEEP_PM_OPS(sdhci_pxav3_suspend,
                            sdhci_pxav3_resume),
    SET_RUNTIME_PM_OPS(sdhci_pxav3_runtime_suspend,
                       sdhci_pxav3_runtime_resume, NULL)
};

static struct platform_driver sdhci_pxav3_driver = {
    .driver = {
        .name = "sdhci-pxav3",
        .of_match_table = of_match_ptr(sdhci_pxav3_of_match),
        .pm = &sdhci_pxav3_pmops,
    },
    .probe = sdhci_pxav3_probe,
    .remove = sdhci_pxav3_remove,
};

/* mem – zwaną także sram (ang. suspend to ram) zakłada utrzymanie zasilania jedynie dla pamięci RAM, utrzymywanej w trybie odświeżania (ang. self-refresh). Ponieważ wszystkie rdzenie są wyłączone, rozwijanie to wymaga dodatkowych układów na płycie i/lub w samym SoC'u (np. dodatkowy maly procesor), który przywróci zasilanie wskutek przerwania. W przeciwieństwie do standby wartość rejestrów nie zachowuje się i wymaga odtworzenia po obudzeniu. Poza tym jednak sekwenca mem z punktu widzenia Linuksa i sterowników nie wykazuje różnic.
```

```

    » disk – najgdyś tryb uśpienia, w którym zawartość pamięci kopiowana jest na dysk.

    Abi poinformować system o wspieranych metodach, należy zarejestrować strukturę platform_suspend_ops zawierającą przypisanie co najmniej dwóch funkcji zwrotnych: .valid, której zadaniem jest określić listę obsługiwanych trybów, oraz .enter, czyli właściwa implementację wejścia w stan uśpienia. Dotychczas każda platforma musiała posiadać w tym celu własną inicjalizację, jednak w ARMv8 cała niskopoziomowa konfiguracja została przeniesiona do firmware'u, co pozwala na korzystanie z wymienionego wyżej generycznego sterownika PSCI.
```

**Listing 5. Rejestracja obsługi uśpiania w ARMv8**

W Listingu 5 przedstawiono rejestrację w Linuksie struktury psci\_suspend\_ops przez funkcję suspend\_set\_ops() po uprzednim sprawdzeniu, czy funkcjonalność została dodana do konfiguracji jądra, i optymalnie oprogramowania układowego przy użyciu SMC (funkcja psci\_features()) o możliwość obsługi. Warto zwrócić uwagę, że polu .valid przypisana jest funkcja suspend\_valid\_only\_mem(), co oznacza, że domyślnie ARMv8 wspiera wyłącznie tryb *mem*. Funkcją zwrotną .enter jest natomiast generyczna metoda cpu\_suspend(), która jako drugi argument przyjmuje psci\_system\_suspend() zawierającą wywołanie SMC.

### Sekwencja zdarzeń w systemie operacyjnym

Linux udostępnia informację o wspieranych trybach oszczędzania energii w systemie plików sysfs. Aby wyświetlić ich listę, należy wykonać polecenie:

```

$ cat /sys/power/state
mem
standby
freeze
disk

```

Natomiast uruchomienie wybranego trybu uzyskuje się przez wpisanie jego nazwy do tego samego pliku, np. dla *mem*:

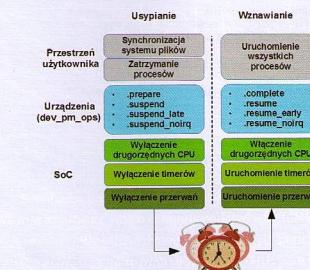
```

$ echo mem > /sys/power/state

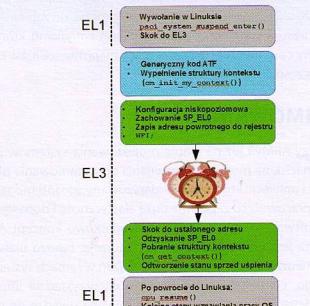
```

Użycie powyższej komendy będzie skutkowało wykonaniem sekwencji uśpienia systemu operacyjnego, która zaczyna się synchronizacją systemu plików, zatrzymaniem procesów przestrzeni użytkownika i pozostałych zadań. Następnie wyłączone są wszystkie urządzenia, co dzieje się w czterech etapach przez wykonanie kolejnych funkcji zwrotnych struktury dev\_pm\_ops: .prepare, opisany wyżej, .suspend, .suspend\_late, .suspend\_noirq. Ostatnie etapy to wyłączenie drugorzędnych procesorów, timerów oraz przerwań, z wyjątkiem tych, które mają za zadanie obudzić system (Rysunek 3).

Linuk oferuje mechanizm kontroli sygnałów budzących system, w przestrzeni użytkownika dostępny przez sysfs. Przed uśpieniem przerwanie musi być zarejestrowane i dodatkowo należy powiadomić jądro, wpisując wybranym urządzeniem tekst enabled (przy wyłączeniu – disabled) w pliku /sys/devices/.../



Rysunek 3. Sekwencja uśpienia/wznowienia w Linuksie



Rysunek 4. Uśpienie i budzenie procesora Armada 3720

power/wakeup. Dzięki temu, aby wznowić normalne działanie, można wykorzystać np. GPIO, UART czy pin detekcji karty SD. Niektóre urządzenia mogą wymagać dodatkowych ustawień niskopoziomowych, jak *wake-on-LAN* dla interfejsu sieciowego.

Podczas wznowiania jądro systemu Linux, poczynawszy od funkcji cpu\_resume(), wykonuje przeciwnie operacje i w odwrotnej kolejności niż podczas uśpiania, uruchamiając timer oraz przerwanie, włączając wszystkie procesory, urządzenia (również w czterech etapach), a na końcu przywracając do działania przestrzeń użytkownika i procesy. Całą sekwencję uśpienia i wznowienia przedstawiono na Rysunku 3.

W ARMv8 powyższa sekwenция korzysta z generycznych funkcji PSCI w trzech momentach – wyłączenia i wzłączania drugorzędnych procesorów oraz wejścia w tryb uśpienia. Powróćmy do Listingu 5 – w definicji psci\_system\_suspend() można zauważyć, że jednym z argumentów funkcji SMC jest wskaźnik na funkcję cpu\_resume(), aby to ona jako pierwsza mogła być wykonywana po powrocie z firmware'u w EL3 do Linuksa.

### WALKA Z KONTEKSTEM, CZYLI DO PLATFORMY PODEJŚCIE PIERWSZE

Platformą, na której zaimplementowano niskopoziomową obsługę uśpiania, był układ Armada 3720 firmy Marvell na bazie rdzeni

ARM Cortex-A53. Jego konstrukcja umożliwia ustalenie w rejestrze adresu, do którego maszyna skoczy po obudzeniu i wykona kod pozwalający na powrót z EL3 do EL1 i kontynuowanie sekwencji wznowiania w Linuksie (Rysunek 4).

Na Rysunku 4 przedstawiono kolejne etapy, które wykonywane są przez generyczny i platformowy kod ATF w sekwencku uśpienia i wznowiania działania SoC-a. Po obudzeniu jedynie licznik programu jest skonfigurowany zgodnie z wcześniejszym wpisem do rejestru, natomiast wskaźnik na ramkę stosu z domyślnego SP\_EL0 sprzątającej na SP\_EL3. Należy więc przywrócić wszystkie ustawienia, aby odtworzyć stan przed uśpieniem. Z pomocą przychodzi generyczny framework ATF (ctx\_mgmt) do zarządzania kontekstem, który w dwóch globalnych wskaźnikach (dla świata „normalnego” i „zaufanego”) pozwala przechowywać struktury z zapisaną zawartością rejestrów układu z różnych poziomów wyjątków. W cyklu uśpiania wolana jest funkcja cm\_init\_my\_context(), która zachowuje adres powrotu do EL1 oraz pozostałe informacje.

### Listing 6. Uśpianie i odtwarzanie kontekstu po obudzeniu

```

static void a3700_pm_suspend(void)
{
    /* Set return address */
    mmio_write_32(MVEBU_PMSU_REG_BASE +
                  0x40,(uintptr_t)&a3700_pm_wake_up);

    /* Store current stack pointer (SP_EL0) into global variable
       before */
    asm volatile("mov %0, sp\n"
                : "=r" (a3700_sp_el0));

    plat_marvell_gic_cpuif_enable();
    [...]
    /* Issue wait for interrupt command */
    wfi;
}

static void a3700_pm_wake_up(void)
{
    cpu_context_t *ctx;

    ctx = cm_get_context(NON_SECURE);
    assert(ctx);

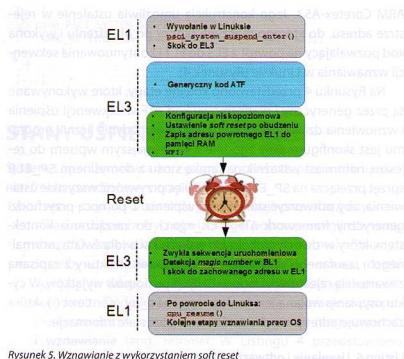
    /* Put ctx structure back on SP_EL3, restore SP_EL0 and switch
       to it */
    __asm__ volatile("mov sp, %0\n"
                    "msr spsel, #0\n"
                    "mov sp, %1\n"
                    : "+r" (ctx), "+r" (a3700_sp_el0));
    plat_marvell_gic_cpuif_enable();
    el3_exit();
}

```

W Listingu 6 przedstawiono zapis adresu funkcji, która ma być wykonywana po obudzeniu, a także zachowanie SP\_EL0 w globalnej zmiennej oraz ostatnią instrukcję, czyli wfi (ang. *wait for interrupt*). Wznowiany system odzyskuje SP\_EL0, pobiera strukturę kontekstu za pomocą cm\_get\_context() i umieszcza do niej wskaźnik w SP\_EL3, z tego wymaga funkcja el3\_exit(). Bezpośrednio po niej następuje powrót do EL1 i wykonywanie linuksowego cpu\_resume().

### A MOŻE LEPIEJ SPRAWDZIĆ POCZĘT?

Nieco prostszym sposobem niż powyższy okazał się mechanizm „skrzynki pocztowej”, przedstawiony na Rysunku 5.



Rysunek 5. Wznowianie z wykorzystaniem soft resetu i ustawianiem paci\_system\_expend\_enter()

Na końcu sekwencji usypiania adres powrotu do EL1 jest zapisywany do ustalonego miejsca w pamięci, natomiast SoC konfiguruje się tak, aby wskutek przerwania został zresetowany. Po tym rozpoczęta się opisywana już sekwencja uruchomieniowa układu. Ponieważ zawartą RAM RAM nie ulega zniszczeniu, na wczesnym etapie można wykryć, czy mamy do czynienia ze startem systemu (ang. *cold boot*) albo resetem po spuścieniu (ang. *soft reset*), co pozwala na powrót do systemu operacyjnego.

Listing 7. Sprawdzanie „skrzynki pocztowej” w BL1

```
/*
 * Main job of this routine is to distinguish between a cold
 * and warm boot.
 * For a cold boot, return 0.
 * For a warm boot, read the mailbox and return the address it
 * contains.
 * A magic number is placed before entrypoint to avoid mistake
 * caused by
 * uninitialized mailbox data area.
 */

```

## W sieci

- ▶ <https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/firmware-design.md> – szczegółowy opis ARM Trusted Firmware
- ▶ <http://www.slideshare.net/linaroorg/arm-trusted-firmware&alcu3> – prezentacja nt. stosu oprogramowania architektury ARMv8
- ▶ [http://infocenter.arm.com/help/topic/com.arm.doc.den028B/ARM\\_DEN028B\\_SMC\\_Calling\\_Convention.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.den028B/ARM_DEN028B_SMC_Calling_Convention.pdf) – SMC
- ▶ [http://infocenter.arm.com/help/topic/com.arm.doc.den0022C/DEN0022C\\_Power\\_State\\_Coordination\\_Interface.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.den0022C/DEN0022C_Power_State_Coordination_Interface.pdf) – PSCI
- ▶ <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/firmware/psci.c?id=refs/tags/v4.10-rc5> – sterownik PSCI w Linuksie
- ▶ <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/documentation/power/states.txt?id=refs/tags/v4.10-rc5> – stany usypiania w Linuksie
- ▶ <https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/documentation/power/devices.txt?id=refs/tags/v4.10-rc5> – sekwencja suspend/resume w jądrze Linuska
- ▶ <http://www.slideshare.net/ennael/kernel-recipes-2015-introduction-to-kernel-power-management> – prezentacja na temat zarządzania energią w Linuksie

## Własny blog z Umbraco w 1,5 godziny

Popularność blogów nieustannie rośnie, a na rynku IT wszystko, co z nimi związane, z reguły sprawdza się do jednego słowa i narzędzia – Wordpress. Idąc na przekór oklepanym schematom, serię artykułów poświęconych innemu systemowi do zarządzania treścią, który zdobywa popularność również w Polsce – Umbraco CMS – chciałbym rozpocząć właśnie od tematu poświęconego temu, w jaki sposób możemy zbudować z jego wykorzystaniem prostego bloga (w dodatku nie poświęcając na to więcej niż 1,5 godz.).

### WPROWADZENIE

Programiści PHP bądź osoby mniej techniczne mające już do czynienia z Wordpressem mogliby w nawiązaniu do tytułu szybko sprawdzić mnie do parteru, mówiąc, iż bloga mamy tam już „out-of-box” i nic nie musimy dodatkowo kodować. To prawda. Wordpress jest doskonałą i niesamowicie popularną platformą publikacyjną. Niemniej jednak posiada również szereg idących za tą popularnością niebezpieczeństw oraz równie sporo ograniczeń wynikających z nastawienia na bycie najlepszym narzędziem do blogowania. My, programiści, chcemy przecież często wykorzystywać systemy, z którymi pracujemy, wręcz do granic ich możliwości, a nieraz również wykrańczać poza nie! W sytuacjach, w których chcemy stworzyć solucję skróconą na miarę oraz mieć nad nią pełną kontrolę, uciekamy się nierzadko do wykorzystania innych narzędzi, bądź co gorsza (w mojej subiektywnej opinii) – budujemy własne, które niesamowicie ciekąco nam później utrzymywamy.

Umbraco to w pełni funkcjonalny opensource'owy system do zarządzania treścią w technologii ASP.NET, który jest na tyle elastyczny, że poradzi sobie z wdrożeniem i funkcjonowaniem zarówno w scenariuszu, który dotyczy malej strony WWW – tworzonej na potrzeby pojedynczej kampanii czy np. wizytówki w sieci, jak i również zaawansowanych aplikacji internetowych budowanych na zasadach topowych firm z największymi portali informacyjnymi i medialnymi na świecie. Jest to także jedyny system CMS oferujący w chwili obecnej model funkcjonowania na zasadach SaaS (patrz: <https://umbraco.com/products/umbraco-cloud/>).

Rysunek 1. Początkowy ekran instalacyjny Umbraco CMS

### CO, GDZIE I JAKI?

Po poprawnej konfiguracji zostajemy przekierowani do panelu administracyjnego. Często okazuje się, że pomimo prostoty i minimalizmu interfejsu, zrozumienie całej i organizacji struktury wewnętrznej Umbraco potrafi zająć dłuższą chwilę tym, którzy nie mieli z nim wcześniej styczności.

Istotą wczuć się w cały workflow pracy z Umbraco jest poznanie hierarchii i relacji pomiędzy elementami definiowanymi w różnych sekcjach. W wielkim skrócie wygląda na następująco: bez typów danych (ang. *Data Type*), osadzonych w sekcji *Developer*, nie możemy utworzyć typów dokumentów (ang. *Document Type*), które budujemy w sekcji *Settings*. Bez typów dokumentów nie jesteśmy w stanie stworzyć żadnej strony w sekcji *Content* ani wyświetlić żadnego szablonu (już po stronie frontendu).

### TYPY DOKUMENTÓW

Proces budowania rozwiązania z Umbraco należy zatem rozpocząć od zaplanowania typów dokumentów oraz typów danych, które będą je tworzyć. Można by rzec, że potrzebujemy modeli danych, na bazie których utworzymy ich reprezentacje. Z reguły wystarczą



```

/* Image Background */
var backgroundImage = Umbraco.TypedMedia(Model.Content);
if (backgroundImage != null)
{
    var backgroundImageUrl = backgroundImage;
    GetCropUrl("header-image");
    headerStyle = string.Format("background-image: url('{0}');", backgroundImageUrl);
}
}

<header class="intro-header" style="@headerStyle">
<div class="container">
<div class="row">
<div class="col-lg-8 col-lg-offset-2 col-md-10 col-md-offset-1">
@if (Model.Content.DocumentTypeAlias == BlogPost.
ModelTypeAlias)
{
    <div class="post-heading">
        <h1>@Html.Raw(Model.Content.BasePageTitle)</h1>
        <h2 class="subheading">@Html.Raw(Model.Content.
BasePageSubTitle)</h2>
        <span class="meta">@Html.Raw(Model.Content.Oftype<BlogPost>.
PublishedDate.ToString("d MMMM, yyyy"))</span>
    </div>
}
else
{
    <div class="site-heading">
        <h1>@Html.Raw(Model.Content.BasePageTitle)</h1>
        <hr class="small">
        <span class="subheading">@Html.Raw(Model.Content.
BasePageSubTitle)</span>
    </div>
}
</div>
</div>
</header>

```

W tym przypadku również skorzystaliśmy z bazowego kodu HTML z szablonem, dodając do niego jedynie wartości przechowywane w Umbraco na danej stronie. Tym sposobem otrzymaliśmy dynamiczny nagłówek, który w zależności od dodanych w podstronie informacji oraz jej typu wyświetli nam piękny nagłówek strony.

## LISTING POSTÓW

Najbardziej ze złożonych typów stron, w naszym prostym mimo wszelko blogu, jest strona główna z listiem postów. Złożona ze względu na to, iż wypadłooby zadbać o odpowiednie stronnicowanie wpisów, tak by nie skorzyć z nieskończoną listą elementów. Do tego celu wykorzystamy prosty parametr `QueryString` w adresie strony, na bazie którego odfiltrujemy elementy na danej stronie. Nie jest to najbardziej optymalne rozwiązanie, ale na potrzeby stosunkowo niewielkich witryn w zupełności wystarczy.

### Listing 4. Szablon (fragment) strony głównej z listiem wpisów

```

@inherits UmbracoTemplatePage<Blog>
@{
    Layout = "Master.cshtml";
}

// Paging Values
const int pageSize = 6;
int page;

if (!int.TryParse(Request.QueryString["p"], out page))
{
    page = 1;
}

var posts = Model.Content.Children<BlogPost>()
    .OrderByDescending(x => x.PublishedDate);

var totalNodes = posts.Count();

```

## RSS

Celem funkcjonalnego dopełnienia naszego bloga do absolutnej wersji minimum niezbędną funkcjonalnością jest jeszcze kanał RSS. Zakładamy bowiem, że nasz blog będzie bardzo pocztynym, a nasza grupa docelowa będzie chciała mieć najbliższe wpisy zawsze pod ręką.

W szablonie dedykowanym dla strony RSS należy zatem utworzyć kod, którywróci w odpowiedzi plik XML z określona dla standardu RSS strukturą zawartości. Do wyムuszenia przez przeglądarkę traktowania naszej strony jako pliku XML posłużym obiekt `Response` reprezentujacy odpowiedź serwera, w którym czyszcymy obecny typ odpowiedzi oraz określamy ten, który jest przez nas wymagany:

### Listing 6. Ustawienie odpowiedniego kodowania oraz typu odpowiedzi

```

Response.Clear();
Response.ContentType = "text/xml";
Response.ContentEncoding = Encoding.UTF8;

```

Dodatkowo do kompletu danych początkowych potrzebujemy: absolutnego adresu naszej strony z wpisami, listy wszystkich artykułów oraz daty publikacji ostatniego z wpisów. Korzystając z dynamicznego obiektu `CurrentPage` oraz helpera Umbraco, pobieramy te dane, wykorzystując następujące metody i zapisując je w zmiennej:

### Listing 7. Pobranie niezbędnych do wygenerowania kanału RSS danych ze struktury treści w Umbraco

```

var rssPage = CurrentPage;
var blog = Umbraco.TypedContentAtRoot()
    .First(x => x.DocumentTypeAlias == Blog.ModelTypeAlias)
    .OfType<Blog>();
var siteurl = blog.UrlWithDomain().TrimEnd("/").ToCharArray();
var articles = blog.Children<BlogPost>()
    .OrderByDescending(x => x.PublishedDate);
var lastUpdate = articles.Any() ?
    DateTime.Now :
    articles.First().PublishedDate;

```

Na koniec renderujemy jedyne odpowiednią strukturę dokumentu, wykorzystując do tego pobrane elementy oraz pętlę po wszystkich artykułach w naszym blogu.

### Listing 8. Budowa dokumentu RSS

```

<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
    <atom:link href="String.Format("{0}/{1}", siteurl, rssPage.
    Url)" rel="self" type="application/rss+xml" />
    <title>@blog.MetaTitle</title>
    <@Html.Raw("<link href='@siteurl@Html.Raw("</link>")" />">
    <description>@blog.BasePageSubTitle</description>
    <lastBuildDate>@String.Format("{0:ddd, dd MMM yyyy HH:mm:ss",
    EST, lastUpdate}</lastBuildDate>
    <language>pl</language>
    <generator>Umbraco</generator>
    <foreach (var article in articles)>
        <var articleTitle = !string.IsNullOrWhiteSpace(article.
        BasePageTitle) ? article.BasePageTitle : article.Name;
        var articleDescription = article.BasePageSubTitle;
        var articleUrl = article.UrlWithDomain();
        var articlePublishedDate = article.PublishedDate;

```

```

        <item>
            <title>@articleTitle</title>
            <@Html.Raw("<link href='@articleUrl@Html.Raw("</link>")" />">
            <comments>@string.Format("{0}#disqus_thread",
            articleUrl)</comments>
            <description>@articleDescription</description>
            <pubDate>@string.Format("{0:dd/MM/yyyy
            :HH:mm:ss} EST", articlePublishedDate,
            articlePublishedDate)</pubDate>
            <guid>@string.Format("{0}/{1}", siteurl, article.Id)</
            guid>
        </item>
    </channel>
</rss>

```

## PODSUMOWANIE

Systemy do zarządzania treścią to nie tylko gotowe narzędzia, z wykorzystaniem których możemy budować ścisłe określone rozwiązania. Niektóre z nich to swoiste frameworki, które w ręках programistów stają się narzędziem znacznie skracającym drogę do wytworzenia oprogramowania o różnej specyfikacji. Umbraco CMS pozwala nam na dosyć elastyczne rozszerzanie i tworzenie dowolnych, mniej lub bardziej skomplikowanych, projektów. Korzystając z funkcjonalności dostarczonych w systemie oraz poświęcając na to stosunkowo niewielką ilość czasu, jesteśmy w stanie stworzyć gotowego i funkcjonalnego bloga z pełną możliwością zarządzania treścią, elementami dynamicznymi oraz zintegrowanym szablonem HTML, który możemy dowolnie modyfikować. Jeśli ktokolwiek chciałby zobaczyć bloga, który powstał w trakcie pracy nad tym artykułem, serdecznie zapraszam na <http://udfind.pl/>.



Rysunek 7. Ekran startowy bloga powstającego w trakcie tworzenia artykułu

Wszystkie listingi zawierają kod w języku C#, a dokładniej składni Razor.

## W sieci:

- ▶ Strona z możliwością pobrania Umbraco CMS: <https://umbraco.com/products/umbraco-cms/download-umbraco/>
- ▶ Repozitorium kodu Umbraco: <https://github.com/umbraco/Umbraco-CMS>
- ▶ Grupa polskiej społeczności Umbraco: <https://www.facebook.com/groups/umbracopoland/>



MARCIN ZAJKOWSKI

[marcin.zajkowski@thecogworks.com](mailto:marcin.zajkowski@thecogworks.com)

Certyfikowany trener i pierwszy w Polsce Umbraco Certified Master. Na co dzień pracuje jako Senior Umbraco Developer w The Cogworks, organizując spotkania grupy społeczności w Polsce oraz ewangelizując Umbraco, prowadząc szkolenia i prezentacje na jego temat podczas wszelkiego rodzaju wydarzeń IT.

# Nietypowe metody wykorzystywane w atakach phishingowych

Celem artykułu jest zwiększenie powszechniej świadomości, że phishing to już nie tylko proste wysłanie wiadomości z podrobioną stroną WWW i próba wyłudzenia naszego hasła do poczty – czasami jest to znacznie bardziej zaawansowany i lepiej zaplanowany strategicznie atak. Warto poznać te „nietypowe” metody, ponieważ być może ich znajomość uchroni nas przed niebezpieczeństwem.

## WSTĘP

W ciągu ostatnich lat phishing bardzo ewoluował. Powstanie wielu nowych technik – i w związku z tym modyfikacja dostępnych rozwiązań – przenosi ten typ ataku na wyższy poziom. Atakując coraz częściej do tego dysyrybucji wykorzystują pocztę elektroniczną, strony WWW lub wiadomości prywatne w komunikatorach. W tym artykule postaram się przybliżyć jedne z najpopularniejszych metod wykorzystywanych obecnie w kampaniach phishingowych.

### Typosquatting

Typosquatting polega na rejestracji domen z zastosowaniem prostych błędów literowych, które mogą zostać popełnione przez internautów podczas wpisywania nazwy strony w pasku adresu przeglądarki. Przykładem mogłyby być pominięcie jednej z liter lub wprowadzenie takiej, która sąsiaduje z daną literą na klawiaturze.

Inne proste zastosowania tej techniki:

- » wwwnazwadomeny (bez kropki pomiędzy „www” a nazwą domeny)
- » www-nazwadomeny (kreska pomiędzy „www” a nazwą domeny)
- » nazwadomeny-dowolnazwaz (kreska pomiędzy nazwą oryginalnej domeny a nazwą domeny)

Warto dodać, że w przeszłości najbardziej popularne było umieszczanie na „przejrzystej” przez atakującego domenie katalogu z nazwą domeny, pod którą chciał podłożyć się atakującemu.

Z ciekawą analizą typosquattingu, opartą na przeglądzie ponad 500 popularnych stron, można się zapoznać pod poniższym adresem:

- » <https://lirius.kuleuven.be/bitstream/123456789/471369/3/typos-final.pdf>

### Homoglyph & homograph

Aatak polega wykorzystaniu podobieństw w wyglądzie danej litery, cyfry bądź znaku, poprzez użycie zestawu innych znaków, m.in. cyfr i liter. Często stosowaną metodą jest używanie tych samych liter z innych języków bądź kodowań (więcej o tym pod adresami: <https://sekurak.pl/rośnienie-przejeli-%c9%a2ogoogle.com/> i <https://sekurak.pl/przejety-przez-polscy-hackerow/>).

Najprostszą techniką tego typu jest łączenie liter oraz znaków i ich zamiana na inne. Do najpopularniejszych przykładów możemy zaliczyć ponizsze (w nawiasie przedstawiono czytelniejszą wersję):

- » rm jako m (male litera R i N jako mała litera M)
- » cl jako d (male litera C i l jako mała litera D)
- » q jako g (mała litera Q jako mała litera G)
- » cj jako g (male litera C i j jako mała litera G)
- » vv jako w (podwójna litera V jako litera W)

- » ci jako a (małe litery C oraz I jako mała litera A)
- » I jako l (duża litera I jako mała litera L)
- » I jako l (mała litera L jako duża litera I)
- » 1 jako l (cyfra 1 jako mała litera L)
- » 1 jako l (cyfra 1 jako duża litera I)
- » I jako 1 (mała litera L jako cyfra 1)
- » I jako 1 (duża litera I jako cyfra 1)

W przypadku innego języka i kodowania na poniższych stronach znajdziemy dużą listę odpowiedników znaków:

- » <https://github.com/codobox/homoglyph>
- » <http://homoglyphs.net/?text=tutajnazwa&update=update&unicodepos=1&print=1&lang=en&exc2=1&exc3=1&exc7=1&exc8=1&exc13=1&exc14=1>
- » <http://www.fileformat.info/info/unicode/block/index.htm>

Warto wykorzystać również windowsowe narzędzie „Windows Character Map” (charmap.exe).

Rejestrujący domen starają się chronić właścicieli przed „po-dejrzanym” zakupem domeny, która już istnieje. W tym celu przekształkają znaki lub ich zestawy na wspomniane wyżej odpowiedniki, a następnie odmawiają rejestracji (przykładem może być firma GoDaddy, która przy próbie rejestracji takiej domeny wyświetli informację, że jest już ona „zajęta”).

Dodatekowe informacje:

- » <https://eurid.eu/en/register-a-eu-domain/domain-names-with-special-characters-ids/>

### Punycode

Punycode to specjalne kodowanie, które jest wykorzystywane w celu konwersji Unicode na zestaw dozwolonych znaków w nazwie (tj. liter ASCII, cyfr i myślniki). Domena, która zawiera znaki spoza ASCII, nazywamy Internationalized Domain Name (IDN) – dzięki Punycode jesteśmy w stanie przedstawić jej nazwę w wersji obsługiwanej przez serwery DNS:

Unicode	Punycode
sekurak.pl	xn--seura-v5ad.pl

IDN Conversion Tool: <http://mct.verisign-grs.com/>.

W celu ochrony przed ww. atakami nowoczesne przeglądarki internetowe wyświetlają punycode zamiast oryginalnej nazwy IDN (jeżeli np. znaki zawarte w nazwie domeny nie znajdują się na liście zainstalowanych języków/stowników w przeglądarce). Zamiana

inni ot pidots sinologsin mīnos

znaków oparta jest o ASCII Compatible Encoding (ACE), poprzez m.in. charakterystyczne dodanie prefiku „xn–” – tak jak widzimy to w tabelce powyżej.

### Bitsquatting

Bitsquatting to technika polegająca na rejestracji domeny różniącej się o jeden bit od tej „popularnej”, pod którą chciałby się podać atakujący. Całość opiera się na oczekiwaniu na losowe błędy spowodowane sprzętem komputerowym (najczęściej RAM-em).

Ciekawe opracowanie tego zagadnienia zostało przedstawione pod linkiem: <http://dinaburg.org/bitsquatting.html>.

Natomiast domenę stosującą www. metodę można wygenerować, korzystając np. ze skryptów: <https://github.com/benjaminpetrin/bitsquatting> lub <https://github.com/artedminaburg/bitsquat-script>.

### Right-to-Left Override [RTLO / RLO]

Ponad ostatnie kilka lat mogliśmy widzieć różne metody na wykorzystanie znaku „Right-to-Left Override” (Kod: U+20E; szczegółowe informacje: <http://www.fileformat.info/info/unicode/char/20e/index.htm>):

- » od zmiany rozszerzenia pliku:



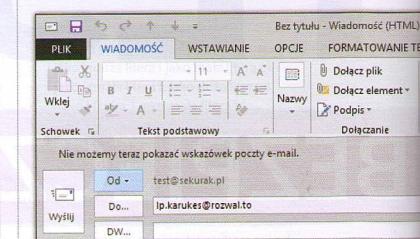
Atakujący umieszcza na stronie WWW kod HTML zawierający link „mailto:” ze znakiem RTLO (URL Encode – %E2%80%AE):

```
<a href="mailto:%E2%80%A0.ip.karukes@rozwal.pl">ip.karukes@rozwal.to</a>
```

Celem przedstawionego powyżej przykładu było zaprezentowanie linku „mailto:ip.karukes@rozwal.pl” w programie pocztowym, np. Microsoft Outlook, ale docelowo wysłanie go na adres „ot.lawzor@sekurak.pl”.

W momencie kliknięcia przez ofiarę ataku w powyższy link w programie pocztowym zobaczymy:

Kontakt: ip.karukes@rozwal.to



Rysunek 2. Okno tworzenia nowej wiadomości w programie Microsoft Outlook 2013 z fałszywym adresem w polu „Do...”

Jednak po kliknięciu przycisku „Wyślij” e-mail zostanie wysłany na inny adres. Jeżeli ofiara ataku przesunie myszkę na pole z adresem, również zobaczy „fałszywy” e-mail:



Rysunek 3. Szczegóły adresu e-mail po najechaniu naiego wskaznikiem myszy

Zawsze uważnie sprawdzajmy adresata naszej wiadomości e-mail.

### URI scheme

Uniform Resource Identifier (URI) to ciąg znaków wykorzystywanych do identyfikacji zasobów. Dzięki niemu możemy m.in. umieścić obrazek na stronie (w kodzie HTML) bez potrzeby dołączania dodatkowych plików graficznych. To znacznie ułatwia przepływem wiele działań, a jednocześnie zmniejsza wykrywalność (generowany jest mniejszy ruch, sprawdzający się do pobrania 1 strony HTML) i sprawia, że nic nie jest ładowane z zewnątrz.

Do najpopularniejszych schematów wykorzystywanych przez phisherów” należy zaliczyć:

- » <data:MIME;WARTOSC>
- » [data:MIME;charset=KODOWANIE;base64,WARTOSC\\_W\\_BASE64](data:MIME;charset=KODOWANIE;base64,WARTOSC_W_BASE64)
- » <data:WARTOSC>

Pole WARTOSC uzależnione jest od typu pliku lub zastosowanego kodowania.

- » po podmianie paska przeglądarki (o czym więcej m.in. tutaj: <https://sekurak.pl/podrabianie-paska-url-w-przegladarkach-poprzez-mechanizm-rtl-right-to-left/>).

Dzisiaj przedstawię mniej popularną metodę, którą atakujący mogą wykorzystywać do „spoofowania” linków z adresem e-mail.

Napisalem „moga”, ponieważ nie jestem pewien, czy rzeczywiście ją stosują. Na taką możliwość użycia znaku RTLO wpadłem samemu – jednakże jest to dość proste wykorzystanie, więc prawdopodobnie ktoś inny również już odkrył.

## NIETYPOWE METODY WYKORZYSTYWANE W ATAKACH PHISHINGOWYCH

Znając ogólny schemat, możemy przystąpić do prezentacji różnych przykładów. Na poniższej stronie znajdziemy prostsze i bardziej zaawansowane zastosowanie Data URI:

» <https://sekurak.pl/przykłady/dataurischeme/>

Przedstawiamy kilka przykładów, które wykorzystują technikę do wykonywania ataków.

### HTML5 Fullscreen API Attack

Atak „HTML5 Fullscreen API” wykorzystuje mechanizm prezentowania zawartości stron WWW w trybie pełnoekranowym. Autorem „nietypowej” prezentacji implementacji ww. API jest Feross Aboukhadijeh (<http://feross.org>). Przestępco, stosując mechanizm, siedzie w stanie zaprezentować fałszywy widok górnego kraja przeglądarki oraz załadować dowolną dolną treść, dzięki czemu może uważyć użytkownika może nie zauważyc „podmiany”.

Na szczęście przeglądarki przed przejściem w tryb pełnoekranowy przez chwilę wyświetlają komunikat podobny do tego: „Naciśnij klawisz ESC, aby wyłączyć tryb pełnoekranowy” – dlatego bądźmy uważni!

» [https://sekurak.pl/przykłady/html5\\_fullscreen\\_api\\_attack/](https://sekurak.pl/przykłady/html5_fullscreen_api_attack/)

### Blankshield & Reverse Tabnabbing

Atak polega na podmianie zawartości zakładki, z której nastąpiło kliknięcie (target="blank") na inną stronę.

Jak atak działa w praktyce, możemy sprawdzić, otwierając w przeglądarce (najlepiej Google Chrome) stronę [https://sekurak.pl/przykłady/blankshield\\_reverse\\_tabnabbing/](https://sekurak.pl/przykłady/blankshield_reverse_tabnabbing/) i klikając na dostępny tam link, a następnie przechodząc na poprzednią kartę.

Na stronie <https://samiux.blogspot.com/2016/09/firefox-480-does-not-vulnerable-to.html> zaprezentowano ciekawe podsumowanie tego, jakie przeglądarki są podatne na ww. atak.

Warto wspomnieć, że Google postanowił, aby nie wypłacać nagród (Bug Bounty) za błędy tego typu: <https://sites.google.com/site/bughunteruniversity/nonvuln/phishing-with-window-opener>.

### Clickjacking

Atak polega na stworzeniu elementu (najczęściej ramki iframe), który zostanie wyświetlony ponad inną funkcjonalności serwisu. Implementację tego ataku mogliśmy wielokrotnie obserwować jakiś czas temu, gdy powstawało wiele stron, które przykrywały przycisk „Lubię to!” z portalu Facebook (obecnie popularność tego ataku zmalała). Niesiadomy użytkownik, klikając w tany link, tak naprawdę zgadzał się na polubienie danej strony fanpage.

Właściciele portali internetowych poprzez zdefiniowanie nagłówka „X-Frame-Options” mogą zdefiniować swoje preferencje odnośnie umieszczania strony głównej w ramce i tym samym zwiększyć bezpieczeństwo swojego portalu.

Prostą prezentację działania ataku można zobaczyć tutaj:

» <https://sekurak.pl/przykłady/clickjacking/>

### URL Redirect

Technika polega na wykorzystaniu zaufania do innej witryny oraz znalezieniu na niej błędu lub funkcjonalności umożliwiającej wprowadzenie adresu, na który nastąpi przekierowanie (OWASP TOP 10 z 2013 roku, punkt A10). W związku z tym schemat ataku mógłby wyglądać następująco:

1. Atakujący znajduje na zaufanej stronie funkcjonalność przekierowania, która nie waliduje adresu docelowego.

2. Atakujący tworzy w swoim serwerze fałszywą stronę phishingową.

3. Atakujący wysyła „socjotechniczną” wiadomość e-mail do ofiary ataku wraz z linkiem:

» [https://zaufanastrona/index.html?przekierowanie=https://stronaatakuującego/phishing.html](https://zaufanastrona/index.html?przekierowanie=https://stronaatakującego/phishing.html)

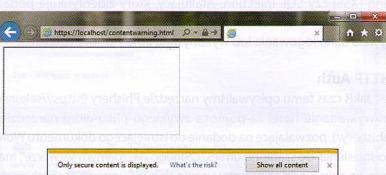
» [https://zaufanastrona/index.html?przekierowanie=javascript:window.location.replace\('https://zaufanastrona/\\*document.location.href='http://stronaatakuującego/phishing.html'\);](https://zaufanastrona/index.html?przekierowanie=javascript:window.location.replace('https://zaufanastrona/*document.location.href='http://stronaatakuującego/phishing.html');)

Ofiara ataku, najeżdżając myszką na hiperłącze, widzi na początku adres „https://zaufanastrona/” co może utwierdzić ją w przekonaniu, że strona jest bezpieczna.

Obecnie nowe przeglądarki internetowe przy próbie przejścia na stronę, na której następuje przekierowanie na adres „javascript:” – wyświetlają komunikat błędu, np. „zawartość została uszkodzona”.

### Content Warning Bypass

Prawdopodobnie czytelnik miał już okazję zobaczyć komunikat informujący o tym, że zostały wyświetlane jedynie treści dostępne za pośrednictwem HTTPS. Dzieje się tak głównie w momencie, gdy jesteśmy na stronie z HTTPS, ale np. w ramce iframe zostaje wyświetlona strona, która nie posiada szyfrowanego połączenia (HTTP). Poniżej zamieszczam przykład:



Rysunek 4. Przeglądarka z komunikatem informującym, że jedynie treści dostępne za pośrednictwem HTTPS zostały załadowane

Jakiś czas temu pojawił się jednak artykuł na stronie BrokenBrowser.com, na której autor informuje o ciekawej sztuczce. W celu załadowania na stronie z HTTPS np. ramki iframe ze źródłem HTTP wystarczy, że wykorzystamy inną stronę z HTTPS, która posiada funkcjonalność przekierowania użytkownika na dolną witrynę.

Przykładowy kod mógłby wyglądać następująco:



Rysunek 5. Omijanie blokady ładowania treści z nieszyfrowanego adresu HTTP na stronie z HTTPS

Dzięki tej technice przestępco mogą w prosty sposób ominąć to ograniczenie.

## BEZPIECZEŃSTWO

### Favicon & Green Padlock

Jest to prosta technika, obecnie już prawie niewykorzystywana z powodu zmian prezentacji miejsca wyświetlania ikony strony. Atakujący tworzy favicon z zieloną lub żółtą kłódką, która sugerowała, jakoby strona posiadała bezpieczne połączenie HTTPS.



Rysunek 6. Podmieniona ikona strony WWW w celu stworzenia złudzenia w postaci bezpiecznego połączenia HTTPS

W przeglądarkach Mozilla Firefox, Safari oraz Google Chrome ikona strony nie jest wyświetlana przy jej adresie, co znacznie zmniejsza ryzyko ataku.

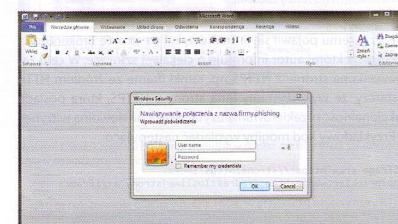
Przypomnijmy, że w przypadku CTFTH nowy elementem jest skrócony URL, który pozwala na skrócenie nazwy strony.

### Content Injection

Podmiana treści na stronie może nastąpić w wyniku pełnego włamania na serwer lub istnienia podatności, np. Cross-Site Scripting (XSS) czy SQL Injection. Atakujący, który zidentyfikował podatność, wpływa na zawartość strony i modyfikuje ją, tak aby przechwytywała dane logowania lub wykonywała inne złośliwe działania.

### HTTP Auth

Jakiś czas temu opisywaliśmy Phishery (<https://sekurak.pl/wykradanie-hasel-za-pomocą-zwyklego-pliku-word-narzedzia-phishery/>), pozwalające na dodanie do istniejącego dokumentu Word „odniesienia” do szablonu umieszczonego na „zdalnym serwerze”, mające na celu kradzież/przechwytcie danych wprowadzonych przez użytkownika w oknie podobnym do poniższego:



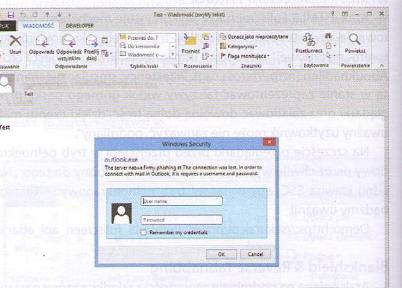
Rysunek 7. Dokument programu Microsoft Word z wyświetlonym oknem do podania poświadczeń

Jeżeli ofiara ataku poda dane logowania, np. do swojego konta domenowego, to atakujący automatycznie je otrzyma:



Rysunek 8. Widok uruchomionego serwera atakującego wraz z fragmentem logu z przechwyconymi danymi do logowania

Warto wiedzieć, że ten typ ataku można również wykorzystać w wiadomościach e-mail – w aplikacji Microsoft Outlook po przesłaniu wiadomości z linkiem, który zabezpieczony jest przez HTTP Basic Authentication. Ofiara ataku, która otrzyma tak przygotowany e-mail, zobaczy ekran podobny do poniższego (w zależności od konfiguracji):



Rysunek 9. Podgląd wiadomości e-mail w programie Microsoft Outlook z widocznym oknem z prośbą o podanie logowania i hasła

Atakujący liczy na to, że ofiara ataku wpisze w takie wyskakujące okno swoje dane logowania.

Domyślnie komunikat posiada schemat: „The server NAZWA\_SERWERA\_ATAKUJĄCEGO at WARTOŚĆ\_AUTHNAME\_Z\_PLIKU\_.HTACCESS requires a username and password”. W polu wartości AuthName z pliku .htaccess atakujący najczęściej wpisuje swoją prośbę. W powyższym przykładzie prośba, tj. „The connection was lost. In order to connect with mail in Outlook, it is...” została napisana w taki sposób, aby „łączyła się” z domyślnym tekstem, tj. „requires a username and password”.

Pamiętajmy, aby nigdy nie wpisywać hasel w takich wyskakujących oknach, których domeny nie znamy! Uważajmy na ten typ ataku, ponieważ staje się on coraz bardziej popularny!

### Pole „Name” z adresem e-mail i spację

Kilką lat temu (ok. 2011 roku), realizując testy socjotechniczne dla jednego z klientów, wpadłem na pomysł związany z wykorzystaniem „ukrywania adresu e-mail”, jeżeli pole „Name” jest zbyt długie.

Nie widziałem wykorzystania tej techniki w praktyce ani nie czytałem o niej, jednakże warto mieć na uwadze taką możliwość. W celu wykonania tego ataku przestępca wprowadza w pole „Name” wartość w schemacie:

Imię i nazwisko <fałszywy@adres.email> [300 SPACJI]

Czyli po wysłaniu wiadomości z powyższym polem (ale bez 300 spacji) odbiorca zobaczy (Rysunek 10):

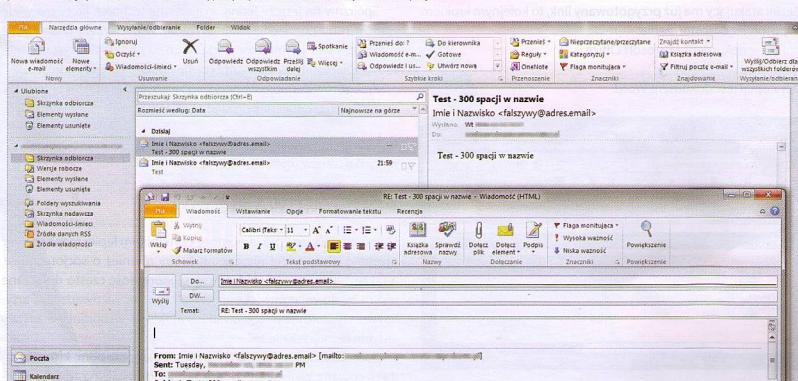
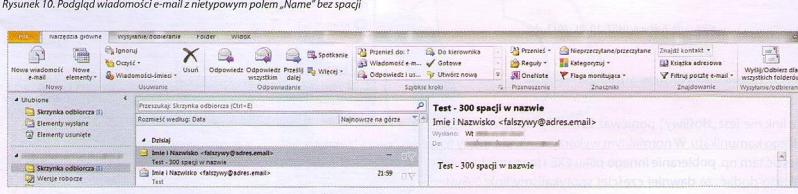
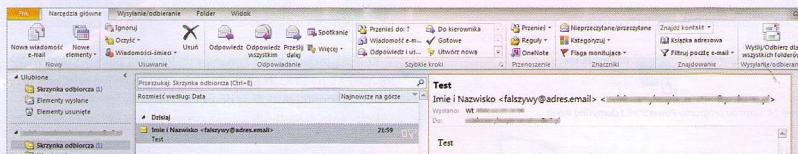
Imię i nazwisko <fałszywy@adres.email> <prawdziwy@adres.email>

Oczywiście prawdziwy adres został zamazany jedynie na potrzeby rysunku do tego artykułu.

Co się jednak stanie, gdy dodamy wspomniane 300 spacji? Wiadomość będzie następujący (Rysunek 11).

Dzieje się tak, ponieważ w zależności od rozdzielczości znaki, które się nie mieścią na ekranie, są ukrywane przez program

## NIETYPOWE METODY WYKORZYSTYWANE W ATAKACH PHISHINGOWYCH



Microsoft Outlook – w związku z tym nasza „długa wartość” pola „Name” nie jest widoczna (w większości przypadków wystarczy 300–500 spacji). A tak wygląda widok, gdy klikniemy przycisk „Odpowiedź” (Rysunek 12).

### .LNK

Plik .LNK zyskały ostatnio ponownie na popularności w trakcie kampanii z złośliwym oprogramowaniem typu „ransomware” (m.in. Locky). Atak polega na stworzeniu skrótu do danego oprogramowania wraz z dodatkowymi poleceniami, które uruchomią bardziej zaawansowane komendy. Przyjrzymy się zasadom działania tej techniki, aby wiedzieć, jak się przed nią bronić.

### PowerShell

PowerShell jest chętnie wykorzystywany w związku z szerokimi możliwościami obfuscacji danego polecenia. W przypadku tej metody atakujący wykonuje prawdopodobnie następujące kroki:

» Tworzy nowy skrót wraz z poleceniem PowerShell:

```
Polecenie: %SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe -command "[System.Reflection.Assembly]::LoadWithPartialName('System.Windows.Forms');[System.Windows.MessageBox]::Show(\"This is PowerShell - Simple Command!\")"
```



## BEZPIECZEŃSTWO

» Po stworzeniu link wygląda następująco:



» Następnie zmienia jego nazwę i ikonę na inną, np.:



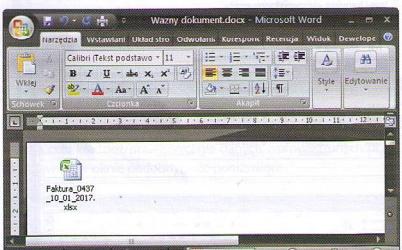
Nasz link nie jest „złośliwy”, ponieważ zawiera jedynie wyświetlenie prostego komunikatu. W normalnym wykorzystaniu atakujący może umieścić tam np. pobieranie innego pliku EXE i jego uruchomienie.

Warto dodać, że dawniej częściej spotykałyśmy link: %SystemRoot%\System32\cmd.exe /c TU\_KOMENDA.

Jeżeli atakujący ma już przygotowany link, to kolejnym krokiem jest sposób jego dostarczenia:

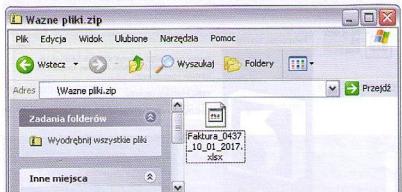
1. LNK + dokument Microsoft Office

Jedną z opcji jest dokument Word:



2. .LNK + archiwum

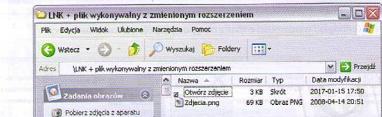
Oczywiście tak przygotowany link (plik .LNK) może zostać umieszczony w archiwum, np. ZIP:



3. .LNK + plik wykonywalny ze zmienionym rozszerzeniem

Kolejną techniką jest zmiana rozszerzenia pliku wykonywalnego, np. EXE na PNG. Następnie atakujący musi już tylko stworzyć skrót, z komendą podobną do poniższej:

Polecenie: %SystemRoot%\System32\cmd.exe /c Zdjeca.png.



### Pozostałe techniki

Spójrzmy na jeszcze jedną, mało znawaną technikę, która ma wiele wad (m.in. nie podmienia pełnego paska, tylko ścieżkę) – wielokrotnie spotykając się z jej implementacją. Mam na myśli wykorzystanie metody „replaceState” w JavaScript do manipulacji paskiem adresu.

Przygotowałem prosty przykład zastosowania tej techniki:  
» [https://sekurak.pl/przykłady/replacestate](https://sekurak.pl/przykladы/replacestate).

Po kliknięciu w powyższy link powinniśmy zobaczyć automatycznie „zaktualizowany” pasek adresu.

## PODSUMOWANIE

Mam nadzieję, że artykuł pomoże czytelnikom lepiej rozpoznawać phishing. Oczywiście nie zapomnijmy również o socjotechnice, a także złożliwym oprogramowaniu, które dość często dołączane jest m.in. do wiadomości e-mail. Należy mieć na uwadze, że artykuł nie wyczerpa listy wszystkich ataków – wspomniane zostały jedynie te „najciekawsze”.

Polecam lekturę tego artykułu wszystkim osobom, które są zainteresowane zwiększeniem swojej świadomości na temat możliwych ataków.

### Źródła i dodatkowe linki:

- » <https://github.com/securestate/king-phisher>
- » <https://github.com/Ralkia/FiercePhish>
- » <https://github.com/ryhanson/phishery>
- » <https://getgophish.com/>
- » <https://www.phishtank.com/>



**ARTUR CZYŻ**

Konsultant ds. bezpieczeństwa IT realizujący projekty z zakresu bezpieczeństwa teleinformatycznego oraz prowadzący szkolenia w firmie Securitum. Zainteresowany tematyką bezpieczeństwa cyfrowego i nowymi technologiami, a także nowymi zagrożeniami. Interesuje mnie rozwijanie się bezpieczeństwa IT i nowe technologie, których zastosowanie może być skutecznym narzędziem do walki z cyberzagrożeniami.

# Zastosowanie grafów w projektowaniu przypadków testowych. Segmentacja grafów

Podczas projektowania systemów informatycznych pojawiają się na ogół różne propozycje implementowania przepływu danych, statusów obiektów oraz GUI. Sposób implementacji systemu lub jego elementów składowych ma wpływ na późniejsze projektowanie przypadków testowych w okresie wytwarzania oprogramowania, w którym jest to możliwe na podstawie specyfikacji.

**W**ematycznie wprowadzona jest teoria grafów. Okazuje się, że wykorzystanie własności grafów oraz ich zastosowanie w systemach informatycznych może ułatwić projektowanie testów w pewnych przypadkach. Celem artykułu jest pokazanie przypadków zastosowania grafów podczas projektowania testów. Pisząc niniejszą pracę, zakładam, że czytelnik zna podstawowe wiadomości na temat grafów skierowanych oraz sposobu ich prezentacji.

## SFORMUŁOWANIE PROBLEMU

Istnieje sporo systemów informatycznych, które umożliwiają prezentację następujących procesów:

- Obieg dokumentów (logistyka, gospodarka magazynowa, dokumenty finansowe, dokumenty kasowe, dokumenty zamówienia itd.),
- Zmiana statusu (stanu) dokumentów,
- Zmiana statusu (stanu) obiektów,
- Proces tworzenia nowego obiektu z innego istniejącego w systemie obiektu,
- Przejście pewnej ścieżki mającej punkty (np. punkty kontrolne), które wymagają przetestowania przed wdrożeniem oprogramowania,
- Prezentacja danych zależnych od siebie hierarchicznie (np. obiekt TreeList),
- Sterowanie lub symulacja sterowania w fabryce.

Wyżej przedstawione procesy mają jedną ważną część wspólną: **można je pokazać za pomocą grafów**. Grafy mają taką własność, że łączą one z sobą zrozumienie zachowania się pewnych zjawisk, czy procesów w systemie informatycznym, a to z kolei może pozytywnie wpłynąć na optymalizację przypadków testowych. Poza przepływem danych czy procesów graf ukazuje również ścieżki niezdolowane w systemie informatycznym, co również może być pomocne podczas projektowania testów. Publikacja przedstawiona w pozycji [3] prezentuje przykład zmiany statusów przetargów budowlanych w Polsce, które zobrazowano są za pomocą grafu.

Inny problem, jaki można napotkać, projektując przypadki testowe za pomocą dużych grafów (drzew danych), to napisanie optymalnego zestawu testów, który sprawdzi wszystkie wymagane funkcjonalności wykonywane na drzewie w systemie informatycznym. Odpowiedź na to pytanie jest próbą znalezienia podobieństwa pomiędzy fragmentami grafu.

## WYBRANE WŁAŚNOŚCI GRAFÓW EULERA I GRAFÓW HAMILTONA

W tym rozdziale zaprezentowane zostaną podstawowe wiadomości oraz własności dotyczące grafów. O tym, jak wygląda graf skierowany lub nieskierowany, wie większość osób związanych z branżą IT, dlatego nie będę opisywać grafów od podstaw. Przymijemy założenie, że cykl prosty to droga zamknięta, gdzie ostatni wierzchołek jest identyczny z pierwszym wierzchołkiem. Pierwsza definicja dotyczy bardzo ważnych w matematyce **grafów Hamiltona**.

**Definicja 1.** [2] Cykl Hamiltona to taki cykl w grafie, w którym każdy wierzchołek grafu przechodzony jest tylko jeden raz (oprócz pierwszego wierzchołka). Ścieżka Hamiltona (ang. *Hamiltonian path*) jest taka ścieżka w grafie, która odwiedza każdy jego wierzchołek dokładnie jeden raz. Grafy zawierające cykl Hamiltona nazywane są **grafami hamiltonowskimi**. Istnieją algorytmy, którymi potrafią odnaleźć cykl Hamiltona w grafie, jeśli taki cykl istnieje. Można też sprawdzić formalnie, czy dany graf jest hamiltonowski np. na podstawie niżej przedstawionych twierdzeń:

**Twierdzenie 1. (O wierzchołkach w grafie)** [3] Jeżeli graf prosty o  $n$  wierzchołkach ma co najmniej  $m$  krawędzi, gdzie:

$$m = \frac{1}{2} \cdot (n - 1) \cdot (n - 2) + 2$$

to jest hamiltonowski.

**Twierdzenie 2. (Ore)** [3] Jeżeli w grafie prostym  $G$  posiadającym  $n$  wierzchołków,  $n > 2$  zachodzi następująca nierówność:

$$\deg(v) + \deg(u) \geq n - 1,$$

dla każdej pary niepołączonych bezpośrednio krawędzią wierzchołków  $u$  i  $v$ , to graf  $G$  posiada cykl Hamiltona.

**Twierdzenie 3. (O grafie nadzdnym)** [3] Niech  $G$  będzie grafem o  $n$  wierzchołkach, a  $C(G)$  oznacza jego nadgraf zbudowany według reguły mówiącej, że dla każdej pary  $\{u, v\}$  nie połączonych bezpośrednio krawędzią wierzchołków takich, że:

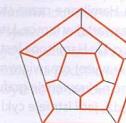
$$\deg(u) + \deg(v) \geq n$$

dodaje się krawędź  $\{u, v\}$ . Graf  $G$  jest hamiltonowski wtedy, i tylko wtedy, gdy  $C(G)$  jest hamiltonowski, oraz:

**Twierdzenie 4.** [2] Jeżeli w prostym  $n$ -wierzchołkowym grafie  $G$ ,  $n > 2$ , stopień każdego wierzchołka jest większy niż  $n/2$ , to graf jest grafem Hamiltona.

## ZASTOSOWANIE GRAFÓW W PROJEKTOWANIU PRZYPADKÓW TESTOWYCH

Pojęcia takie jak stopień wierzchołka, składowe grafu, graf spójny, graf prosty są opisane dokładnie w pozycji [4]. Twierdzenia 1, 2, 4 są podobne, a ich idea jest taka, że „graf jest hamiltonowski, jeżeli ma odpowiednio dużą liczbę krawędzi”. Twierdzenie ostatnie jest nieco inne od trzech pierwszych, ponieważ przedstawia bardzo ciekawą własność związaną z rozszerzaniem grafów hamiltonowskich. Mówią ono, że jeśli wewnątrz grafu znajdziemy cykl Hamiltona, a pozostałe wierzchołki i krawędzie będą spełniały założenia twierdzenia 3, to nasz rozbudowywany graf też jest grafem Hamiltona [4]. Istnieje w teorii grafów takie pojęcie jak graf **półhamiltonowski**. Różni się on tem, że w grafie istnieje jedna droga, która łączy wszystkie wierzchołki, ale nie łączy pierwszego i ostatniego wierzchołka. Przykładowy graf Hamiltona pokazany jest na poniższym rysunku:



Rysunek 1. Przykładowy cykl Hamiltona w grafie dwunastościanu foremnego (źródło: [http://pl.wikipedia.org/wiki/Pl:k&lt;math display="block">Hamiltonian\\_graph\\_example.svg](http://pl.wikipedia.org/wiki/Pl:k%26lt;math%20display%3D%22block%22%3EHamiltonian\_graph\_example.svg%3C/math%3E)

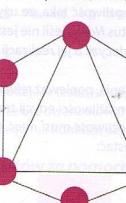
Istnieje inna klasa grafów, tzw. grafy Eulera. Ich nazwa wywodzi się od nazwiska wybitnego matematyka Leonarda Eulera. L. Euler stworzył ogólną teorię grafów, w których istnieje cykl prosty (ciąg czynności), zwany **cyklem Eulera**, zawiązujący wszystkie krawędzie grafu:

**Definicja 2.** [2] Graf posiadający cykl Eulera nazywamy grafem eulerowskim.

Analogicznie do grafów Hamiltona istnieją algorytmy, które potrafią sprawdzić, czy dany graf jest grafem eulerowskim. Z metod formalnych, w tym celu można zastosować twierdzenia:

**Twierdzenie 5.** [2] Spójny graf jest eulerowski wtedy, i tylko wtedy, gdy stopień każdego wierzchołka jest parzysty.

Przykładowy graf Eulera pokazano na poniższym rysunku:



Rysunek 2. Graf Eulera (źródło: <http://student.krk.p.lodz.pl/~Olkusz/index.htm>)

Znalezienie cyklu Eulera w grafie eulerowskim o dużej liczbie krawędzi nie jest łatwe. Aby tego dokonać, można skorzystać z twierdzenia:

**Twierdzenie 6. (Algorytm Fleury'ego)** [2] Niech graf będzie grafem eulerowskim. Wówczas następująca konstrukcja jest wykonalna i daje cykl Eulera w grafie:

- Zaczynamy cykl w dowolnym wierzchołku oraz przechodzimy krawędzie, dbamy o zachowanie reguł 2 i 3,
- Usuwamy z grafu przechodzone krawędzie i wierzchołki izolowane powstałe w wyniku usuwania krawędzi,

3. Przechodzimy przez wcześniej przebyty wierzchołek wtedy i tylko wtedy, gdy nie ma innej możliwości.

Istnieje jedno ciekawe twierdzenie pozwalające stwierdzić, czy graf jest grafem Eulera:

**Twierdzenie 7. (Twierdzenie Eulera o drodze Eulera)** [4] Graf spójny, mając dokładnie dwa wierzchołki stopnia nieparzystego ma drogę Eulera (taką ścieżką w grafie, która przechodzi przez każdą jego krawędź dokładnie raz).

Analogicznie do grafów półhamiltonowskich istnieją grafy **półeulerowskie**. Ich cechą jest to, że nie są grafami eulerowskimi, ale zawsze mają ścieżkę łączącą wszystkie krawędzie, jednak pierwszy wierzchołek jest inny niż ostatni. Na podstawie niżej przedstawionego twierdzenia, które może być również wnioskiem z twierdzenia 7, można sprawdzić, czy dany graf jest półeulerowski:

**Twierdzenie 8.** [2] Spójny graf jest półeulerowski wtedy, i tylko wtedy, gdy ma dokładnie dwa wierzchołki stopnia nieparzystego.

Z twierdzenia 7 wynika, że w grafie istnieje jedna droga, która łączy wszystkie wierzchołki, ale nie łączy pierwszego i ostatniego wierzchołka. Na koniec w drugim takim wierzchołku. Należy także zwrócić uwagę na następujące wnioski:

- Graf Hamiltona nie musi być grafem Eulera,
- Graf Eulera nie musi być grafem Hamiltona,
- Może wystąpić graf, który posiada obie własności, ale jedna własność nie wynika z drugiej,
- Graf może być grafem Eulera lub grafem Hamiltona, pomimo że nie są spełnione założenia wcześniejszych twierdzeń. Spełnione założenia potwierdzają własność grafu, ale może się okazać, że pomimo braku spełnienia założen zawartych w twierdzeniach graf jest grafem Eulera lub grafem Hamiltona.

## PROJEKTOWANIE PRZYPADKÓW TESTOWYCH ZA POMOCĄ GRAFÓW

W tej części artykułu pokażemy, jak należy wykorzystać wyżej przedstawione wiadomości w celu badania własności grafów, zanim zaproektowane zostaną testy.

### Zastosowanie cyklu Hamiltona w projektowaniu testów stanów obiektów w systemie informatycznym

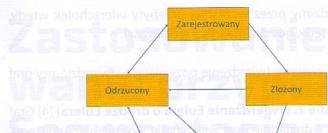
Testowanie statusów w grafie Hamiltona pokażemy na następującym przykładzie:

**Przykład 1.** Pewna firma ma otrzymać oprogramowanie, które umożliwia tworzenie zleceń na transport. Status zlecenia zmienia się zgodnie z zasadami:

- Użytkownik portalu rejestruje dokument zlecenia transportu.
- Zarejestrowany dokument można odrzucić lub złożyć w celu ustalenia szczegółów zlecenia.
- Złożony dokument może zostać odrzucony przez administratora systemu lub może zostać poddany przeglądowi przez osobę odpowiedzialną za wykonanie zlecenia.
- Dokument przeglądany przez administratora może zostać zaakceptowany przez realizację lub odrzucony.
- Z kolei dokument odrzucony można jeszcze raz zarejestrować.
- System dopuszcza możliwość odrzucenia dokumentu zaakceptowanego z uwagi na nieprzewidziane czynności losowe.

Graf, który prezentuje zmianę statusów dokumentu zlecenia, ma postać:

## TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ



Rysunek 3. Graf zmiany statusów dokumentów zlecenia (opracowanie własne)

Wyzłokowany graf nie zawiera dużej ilości węzłów, dlatego nie ma sensu szukać cyklu Hamiltona za pomocą algorytmu. Korzystając z twierdzenia 1, 2 lub 4, łatwo zauważyc, że graf jest hamiltonowski (ale nie półhamiltonowski), a więc można przejść wszystkie wierzchołki za pomocą jednej zamkniętej ścieżki. Krawędź grafu w takim przypadku można utożsamiać z czynnościami, które należy wykonać, aby status zlecenia mógł zostać zmieniony. Cykl Hamiltona w danym grafie stanowi ścieżkę:

**Zarejestrowany → Złożony → Przeglądany → Zaakceptowany → Odrzucony → Zarejestrowany (C1)**

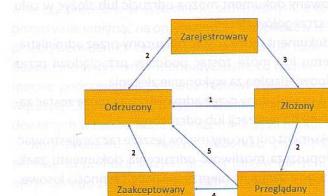
lub:

**Przeglądany → Zaakceptowany → Odrzucony → Zarejestrowany → Złożony → Przeglądany (C2)**

Jak wobec tego wykorzystać tę własność grafu w projektowaniu testów? Odpowiedź na to pytanie nie jest jednoznaczna i tak naprawdę zależy od tego, w jaki sposób podchodzi do testów osoba pisząca przypadki testowe. Mimo to nasuwają się na myśl następujące wnioski:

- Można zaprojektować jeden dłuższy test (może to być test automatyczny), który sprawdzi poprawność **zmiany statusu dokumentu** zlecenia.
- Można zaprojektować liczbę testów równą liczbie przejść pomiędzy stanami, w tym przypadku pięć testów.
- Ponieważ istnieje więcej niż jeden cykl Hamiltona, można wprowadzić wagę do krawędzi. Wagi mogą być zyskiem lub kosztem. Podczas przechodzenia cyklu Hamiltona liczymy sumę wag i w zależności, czy są to koszty lub zysk, bierzemy tę ścieżkę, która jest mniejsza lub większa (dajemy do minimalizacji kosztów lub maksymalizacji przychodów).
- Dopuszczona jest możliwość taka, że użytkownik może ustawić w reklamacji status *Nowy*, jeśli nie jest zadolowany z przynajmniej negatywnej decyzji o jej realizacji.

Przykładowy ważony graf Hamiltona:



Rysunek 4. Ważony graf Hamiltona (opracowanie własne)

W tym przypadku suma wag krawędzi w cyklach (C1) oraz (C2) jest identyczna, wobec tego można do projektowania testów wybrać dowolny z nich.

Istnienie cyklu Hamiltona w oprogramowaniu pozwoli z pewnością zaoszczęścić czas wykonania testów wszelkiego rodzaju zmiany statusów obiektów zdefiniowanych w systemie informacyjnym. Wiele testów wykonuje się manualnie, więc oszczędność czasu jest niezmiernie ważna rzeczą. Zmniejszenie czasu wykonywania czy projektowania testów oraz zredukowanie ilości tych testów jest cenione w firmach z branży IT.

Twierdzenie 3 można zastosować przy dokладaniu nowych funkcjonalności do systemu lub oprogramowania. W przypadku, gdy do aplikacji zostanie wprowadzony nowy status lub kilka nowych statusów dokumentu zlecenia, również warto sprawdzić, czy nie da się przejść cyklem Hamiltona przez statusy. Taki przykład opisany jest w pozycji [3] zawartej w ramce „Literatura”. Testowanie za pomocą wykorzystania cyklu Hamiltona jest zorientowane tylko na statusy dokumentów w wyżej omawianym przykładzie, natomiast nie jest zorientowane na krawędzie grafu, co nie oznacza, że nie trzeba testować krawędzi, jeśli istnieje cykl Hamiltona.

### Zastosowanie grafów Eulera w testowaniu przejść pomiędzy stanami testowanych obiektów

Załóżmy, że istnieje aplikacja, która zawiera obsługę dokumentów reklamacji, zgodnie z następującymi założeniami:

- Zgłoszony dokument reklamacji posiada status *Nowy*.
- Zgłoszona reklamacja może być od razu rozpatrzona albo może zostać przekazana do rozpatrzenia.
- Przekazana do rozpatrzenia reklamacja może zostać zamknięta z powodu jej nieuznania albo może zostać rozpatrzona pozytywnie (prijęta) lub negatywnie (zostać zamknięta).
- Zamknięta reklamacja może z powrotem zostać przekazana do rozpatrzenia, jeśli klient twierdzi, że powinien mieć rozpatrzoną pozytywnie reklamację.
- Pozitynie rozpatrzona reklamacja po realizacji zostaje zamknięta.
- Dopuszczona jest możliwość taka, że użytkownik może ustawić w reklamacji status *Nowy*, jeśli nie jest zadolowany z przynajmniej negatywnej decyzji o jej realizacji.

Podpunkt f. ma sens istnienia, ponieważ reklamacja przekazana do realizacji może nie mieć możliwości edycji treści skargi, natomiast nowa reklamacja taką możliwość musi mieć. Graf przedstawiający statusy reklamacji ma postać:



Z twierdzenia 8 wynika, że graf jest na pewno półeulerowski, ponieważ ma dokładnie dwa wierzchołki stopnia nieparzystego (*Nowy* oraz *Przekażony do rozpatrzenia*). Graf, który ma taką własność,

## ZASTOSOWANIE GRAFÓW W PROJEKTOWANIU PRZYPADKÓW TESTOWYCH

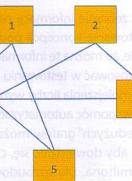
tak naprawdę musi zawierać ścieżkę, która ma w sobie wszystkie krawędzie, co już jest dobrym znakiem dla osoby projektującej testy w oparciu o teorię grafów. Jeśli jednak zastosujemy algorytm Fleury'ego (twierdzenie 4), to okazuje się, że graf prezentujący dokumenty reklamacji zawiera cykl Eulera, który składa się z kroków:

**Rozpatrzony pozytywnie → Zaakceptowany → Przekażany do rozpatrzenia → Rozpatrzony pozytywnie → Przygotyty do realizacji → Zaakceptowany → Nowy → Przekażany do rozpatrzenia**

Wykorzystanie grafów Eulera może przynieść następujące korzyści podczas projektowania testów:

- Można zaprojektować jeden dłuższy test (może to być test automatyczny), który sprawdzi **poprawność przedchedzenia pomiędzy statusami dokumentów** (ogólnie pomiędzy stanem obiektu w systemie informacyjnym).
- W przypadku, gdy będzie istniał więcej niż jeden cykl Eulera, opracować metodę, na podstawie której ktoś będzie decydował, który cykl wybierać do testów.

Testowanie krawędzi stawia główny nacisk na sprawdzenie wszystkich możliwości zmian statusów dokumentów, natomiast niekiedy na samą prezentację stanu dokumentu. Niżej przedstawiony jest graf z dwoma cyklami Eulera:



**1 → 5 → 4 → 1 → 2 → 3 → 4 → 2 → 1 → 3 → 4 → 1 → 5 → 4 → 2 → 1**

Jeśli taka sytuacja zaistnieje w testowanej aplikacji, to decyzja o tym, który cykl wybrać, może być uwarunkowana różnymi kryteriami znającymi z literatury (szczególnie kontekst użycia testowanej aplikacji).

### Optymalizacja testów za pomocą segmentacji grafu

Ta część artykułu poświęcona została systemom informatycznym, w których istnieją procesy dające przedstawić się za pomocą „dużych grafów” albo za pomocą drzew. Słowo „dużych” oznacza tutaj grafy lub drzewa, mające bardzo dużą liczbę węzłów, np. 1000. Istnieją w językach programowania struktury danych przypominające drzewa, np. TreeList w technologii .NET. Wspomniane struktury mogą mieć bardzo dużą liczbę węzłów, a przez co również mogą wymagać przeprowadzenia dużej ilości testów. Przykładowa struktura TreeList przedstawiona jest na Rysunku 7.

Załóżmy, że w widocznym drzewie w pewnym programie zaimplementowane są takie funkcje, jak:

- prezentowanie danych dotyczących węzłów,
- dodawanie węzłów,
- edykcja węzłów,
- usuwanie węzłów,

Numer	Nazwa pozycji	Wartość pozycji
1	Wypłaty dla pracowników	123 300,00 zł
1.1	wynagrodzenie pracowników	108 000,00 zł
1.2	zakładane premie	15 300,00 zł
2	Zakup sadzonek	14 735,00 zł
2.1	drzewa liściaste	1 134,00 zł
2.2	drzewa iglaste	10 612,00 zł
2.3	krzewy	1 035,00 zł
2.4	rośny	1 792,00 zł
3	Usługi zlecone innym firmom	13 060,00 zł
3.1	fontanna	4 800,00 zł
3.2	pomniki	5 460,00 zł
3.3	lawni	2 700,00 zł
4	Koszty utrzymania sprzętu	8 900,00 zł
4.1	paliwo	3 400,00 zł
4.2	konserwacja	5 500,00 zł

Rysunek 8. Przykładowa struktura TreeList (opracowanie własne)

sumą węzłów 1, 1, 2. Analogiczna sytuacja ma miejsce dla węzłów 2, 3, 4. Jeśli natomiast dojdzie do sytuacji, że liście w drzewie (węzły na ostatnim poziomie) osiągną numerację np. 1.1.1.1, a na drzewie są zaprogramowane różne operacje arytmetyczne oraz różne funkcjonalności, to ilość testów potrzebnych do potwierdzenia poprawności działania aplikacji może być bardzo duża, a nawet nie realna do osiągnięcia w czasie. Pojawia się więc wobec tego pytanie: jak optymalnie przetestować poprawność działania funkcjonalności na takim drzewie?

Odpowiedzią na to pytanie może być próba znalezienia w gracie (również w drzewie) takich podgrafów (ogólnie fragmentów drzew), które mają analogiczny wzgldem siebie:

- sposób prezentowania informacji,
- sposób wykonywania pewnej funkcjonalności,
- sposób dziedziczenia własności względem węzłów-dzieci w drzewie,
- sposób przepływu danych (ang. workflow),
- sposób usuwania, dodawania, edycji węzła oraz definiowania połączeń w grafie,
- schemat działania dający się opisać matematycznie (ale nie zawsze jest to łatwa i szybka droga postępowania).

Popatrzmy na drzewo przedstawione na poniższym rysunku:

L.P.	Nazwa pozycji	Ilość (szt.)	Cena	Wartość
1	Zbiorcze koszty	5 984,38 zł		
1.1	Materiały	590,58 zł		
1.1.1	Materiały - A	1.2345	55,00 zł	67,90 zł
1.1.2	Materiały - B	5,7656	45,00 zł	225,45 zł
1.1.3	Materiały - C	4,6540	56,56 zł	263,23 zł
1.2	Prace	5 390,56 zł		
1.2.1	Sprzet - 1	45,0034	75,34 zł	3 390,56 zł
1.2.2	Sprzet - 2	5,6534	34,00 zł	192,22 zł
1.2.3	Sprzet - 3	5,3454	31,00 zł	165,71 zł
1.3	Inne koszty - X	1 645,31 zł		
1.3.1	Inne koszty - Y	45,2356	23,00 zł	1 040,42 zł
1.3.2	Inne koszty - Z	23,2560	11,00 zł	255,82 zł
1.3.3	Inne koszty - T	4,4545	34,00 zł	151,45 zł
1.3.4	Inne koszty - S	2,2896	54,00 zł	123,64 zł
1.3.5	Inne koszty - P	2,3866	31,00 zł	73,98 zł

Rysunek 8. Drzewo prezentujące zestawienie kosztów (opracowanie własne)

Załóżmy, że w widocznym drzewie w pewnym programie zaimplementowane są takie funkcje, jak:

- prezentowanie danych dotyczących węzłów,
- dodawanie węzłów,
- edykcja węzłów,
- usuwanie węzłów,

## TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

- e. sumowanie węzłów-dzieci mających jeden wspólny węzeł – rodzic w kolumnie „Wartość”.
- f. przeliczanie wartości z zaokrągleniem do dwóch miejsc po przecinku.

Załóżmy, że każdy podpunkt od a. do f. jest pewnym warunkiem testowym oraz podczas projektowania testów przyjęte jest takie kryterium, że każdy warunek testowy musi być pokryty co najmniej jednym przypadkiem testowym oraz każdy sposób przetwarzania danych musi być pokryty również jednym przypadkiem testowym. Takie kryterium wbrew pozorom wcale nie jest prosté. Jeśli dwie węzły, które należy przetestować, osiągnie duży rozmiar, to liczba wymaganych przypadków testowych będzie dość duża. Spróbujmy teraz podzielić graf na pewne analogiczne względem siebie grupy segmentów (wspomniana w tytule rozdziału segmentacja grafu):

- » Grupa 1: Węzeł „Zbiorce koszy” ze wszystkimi węzłami.
- » Grupa 2: Węzły „Materiały”, „Sprzęt”, „Inne koszy – X” z węzłami podlegającymi:

1.1 Materiały	590,58 zł
1.1.1 Materiały - A	1.2345 55,00 zł 67,90 zł
1.1.2 Materiały - B	5.7656 45,00 zł 259,45 zł
1.1.3 Materiały - C	4.6540 56,56 zł 263,23 zł

Rysunek 9. Segment z węzłem Materiały (opracowanie własne)

1.2 Sprzęt	3 748,49 zł
1.2.1 Sprzęt - 1	45.0034 75,34 zł 3.390,56 zł
1.2.2 Sprzęt - 2	5.6534 34,00 zł 192,22 zł
1.2.3 Sprzęt - 3	5.3454 31,00 zł 165,71 zł

Rysunek 10. Segment z węzłem Sprzęt (opracowanie własne)

1.3 Inne koszy - X	1 645,31 zł
1.3.1 Inne koszy - Y	45.2356 23,00 zł 1 040,42 zł
1.3.2 Inne koszy - Z	23.2560 11,00 zł 255,82 zł
1.3.3 Inne koszy - T	4.4545 34,00 zł 151,45 zł
1.3.4 Inne koszy - S	2.2896 54,00 zł 123,64 zł
1.3.5 Inne koszy - P	2.3866 31,00 zł 73,98 zł

Rysunek 11. Segment z węzłem Inne koszy - X (opracowanie własne)

- » Grupa 3: wszystkie węzły na najniższym poziomie (11 węzłów).
- Sam fakt, że ostatni segment z grupy 2 ma więcej węzłów, nie wpływa na sposób przetwarzania danych, więc nie ma powodu, żeby zaliczyć go do innej grupy segmentów. Popatrzmy teraz, jak zmniejszy się liczba wymaganych testów w wyniku wskazanego postępowania:

Warunek testowy	Ilość testów bez podziału na segmenty	Ilość testów z podziałem na segmenty
Prezentowanie danych dotyczących węzłów	15	3 (dla każdego poziomu gałęzi)
Dodawanie węzłów	Trudno powiedzieć, ale w celu utworzenia wykresu przyjmijmy po 2 węzły na każdym segment, czyli 10	3 (dla każdego poziomu gałęzi)
Edycja węzłów	15	3 (dla każdego poziomu gałęzi)
Usuwanie węzłów	15	3 (dla każdego poziomu gałęzi)
Sumowanie węzłów dzieci	4	2 (dla poziomu węza 1 oraz poziomu np. węza 1.1)
Przeliczanie wartości z zaokrągleniem	4	2 (dla poziomu węza 1 oraz poziomu np. węza 1.1)

Tabela 1. Tabela prezentująca ilość wymaganych testów bez podziału oraz z podziałem na segmenty

Dane z tabeli bardziej obrazowo zaprezentowano na wykresie:



Rysunek 12. Ilość wymaganych testów bez podziału oraz z podziałem na segmenty

Jak wynika z wykresu, już przy niewielkim drzewie (grafie) liczba testów spadła w wyniku podziału na segmenty, co jest bardzo dobrym rezultatem w procesie optymalizacji projektowania testów oprogramowania.

## PODSUMOWANIE I WNIOSKI

Celem artykułu było zebranie informacji na temat cykli Eulera, cykli Hamiltona, zaprezentowanie koncepcji podziału grafu na segmenty, a następnie pokazanie, że można te informacje oraz wynikające z nich pewne własności zastosować w testowaniu oprogramowania w praktyce, ale pojątkiem zmniejszenia liczby wymaganych testów, lub jako element, który może wspomóc automatyzację wykonanie testów.

W przypadku „niedużych” grafów można skorzystać z matematycznych twierdzeń, aby dowiedzieć się, czy szukając w grafie cyklu Eulera lub cyklu Hamiltona. Dla rozbudowanych grafów bardziej przydatne będą algorytmy, którzy potrafią znaleźć i zaprezentować szukany cykl. Podział na segmenty oraz dostrzeganie podobieństwa w przetwarzaniu informacji, które można opisać na grafie, były trudny, jeśli chciałbym pisać do tego matematyczne modele czy relacje, ponieważ dla każdej funkcjonalności i każdego grafu musiałaby zostać utworzony model matematyczny, a to wymaga sporego nakładu pracy w wielu przypadkach. Dlatego w takię sytuację pomocna będzie testerska logika i własne myślenie.

## Literatura

[1] Roman A., *Testowanie i Jakość oprogramowania*, PWN W-wa, 2015.

[2] Włoch, I. Włoch A.: *Matematyka dyskretna*, Oficyna wydawnicza Politechniki Rzeszowskiej 2008.

[3] Zukowicz M. *Edukacyjne i ekonomiczne aspekty zastosowania cyklu hamiltona w projektowaniu i testowaniu oprogramowania*, General and Professional Education, Wyd. Adsepo, AM Szczecin, 4/2014.

[4] [http://marmaj.math.uni.lodz.pl/TG15\\_W/tg15a.pdf](http://marmaj.math.uni.lodz.pl/TG15_W/tg15a.pdf)

[5] [http://eduminf.waw.pl/inf/alg/001\\_search/0134.php](http://eduminf.waw.pl/inf/alg/001_search/0134.php)

[6] <http://www.scrum.org/Scrum-Guide.html>

[7] <http://www.scrum.org/The-Scrum-Guide.html>

[8] <http://www.scrum.org/The-Scrum-Guide-2012.html>

[9] <http://www.scrum.org/The-Scrum-Guide-2013.html>

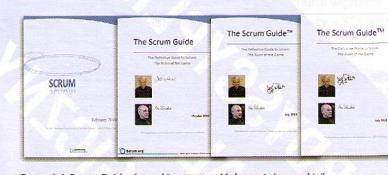
**MAREK ŻUKOWICZ**  
bobmarek@o2.pl

Absolwent matematyki na Uniwersytecie Rzeszowskim, pracuje jako tester oprogramowania. Jego zainteresowania skupiają się wokół obszarów testowania oraz jakości oprogramowania. Interesuje się również wybranymi zagadnieniami matematyki oraz zastosowaniami modeli matematycznych w procesie testowania oprogramowania.

## ZARZĄDZANIE PROJEKTAMI

# Jak zmieniał się Scrum – ewolucja Scrum Guides

Nowy rok jest idealną okazją do przypomnienia sobie zasad Scruma i ponownej lektury Scrum Guide'a. O ile trzymanie się najbardziej aktualnej wersji przewodnika jest poprawne i zalecone, o tyle warto przynajmniej raz wrócić do „korzeni” i samemu przekonać się, w jakim kierunku zmierza ten framework. Celem artykułu było zebranie w jednym miejscu dotychczasowych zmian, jakie nastąpiły w przewodniku na przełomie ostatnich kilku lat, i próba wyjaśnienia, jakie niesie to za sobą konsekwencje.



Rysunek 1. Scrum Guides (rysunki autorstwa Małgorzaty Lamorskiej)

## 2010 – 2011

W 2011r. Ken Schwaber i Jeff Sutherland po długich dyskusjach po stanowili, że formuła Scrum Guide'a zostanie całkowicie zmieniona.

Z pewnego rodzaju poradnika staje się on dokumentem, który zawiera spis reguł, jakie panują w tym framework'u. Twórcy przewodnika w dokumencie *Scrum Update 2011* na przykładzie gier (np. szachów) tłumaczą, że dobrą praktyką jest oddzielenie zasad gry od strategii, jaką posługiujemy się, grając wnią. Tak też się stało, Scrum Guide z 2011 roku jest przede wszystkim książką, która przedstawia nam reguły Scruma. Rozumowanie autorów wydaje się być słusze, istnieje przecież wiele innych książek, artykułów czy blogów, które przedstawiają nam różne strategie na to, jak poruszać się w Scrumie, możliwe, że różnią się one od siebie, ale nawet jeśli tak, to pytanie brzmi – czy to złe?

## Co zniknęło?

Zniknęły podpowiedzi (tzw. tipsy) i dobre praktyki dotyczące różnych zachowań, np. jak zachować się, kiedy mamy dodatkową pracę, synchronizacji sprintu, estymacji, historyjk użytkownika czy łączenia poszczególnych ról. Zniknęła także charakterystyczna dla Scruma historyjka o „świnkach i kurczakach”.

## Co się zmieniło?

Pracę, jaką wykonuje cały Zespół Developerski i dostarcza podczas sprintu, nazywamy teraz przyrostem (ang. increment), a pojęcie commitowania zostaje zamienione na prognozowanie wielkości przyrostu, jaki zostanie dostrzegony. Wykres spalanía nie jest już wymagany, ale dalej jest zalecane. Dodatkowo podział tego spotkania na dwie formalne części nie jest już wymagany, nie zmienia to dalej faktu, że kiedy się ono zakończy, musimy być w stanie odpowiedzieć na dwa pytania: „co może zostać wykonane w Sprintie?” i „w jaki sposób zostanie to zrobione?”. Podkreślono także rolę celu sprintu, który powinien być sformułowany na koncu Sprint Planningu. Bez wspólnego celu sprintu zespołowi trudno skupić się na wybraniu odpowiedniego kierunku prac.

W przypadku Daily Scruma wyraźnie zostaje zaznaczony jego charakter planningowy, planujemy na nim, co jako zespół robimy



Rysunek 2. Zmiany w Scrum Guide 2011r.

## 2011–2013

Czerwiec 2013 roku przynosi nam nową wersję przewodnika. Tym razem jego forma pozostała taka sama, podkreślono jednak rolę planowania, przygotowania się do sprintu i jego celu.

Na ciekawe zmiany natrafiamy już w paragrafie dotyczącym Planowania Sprintu, które od tej pory trwa do 8 godzin i jest zwykle krótsze dla krótszych sprintów niż miesiąc, co oznacza, że skrócenie czasu tego spotkania proporcjonalnie do długości sprintu nie jest już wymagane, ale dalej jest zalecane. Dodatkowo podział tego spotkania na dwie formalne części nie jest już wymagany, nie zmienia to dalej faktu, że kiedy się ono zakończy, musimy być w stanie odpowiedzieć na dwa pytania: „co może zostać wykonane w Sprintie?” i „w jaki sposób zostanie to zrobione?”. Podkreślono także rolę celu sprintu, który powinien być sformułowany na końcu Sprint Planningu. Bez wspólnego celu sprintu zespołowi trudno skupić się na wybraniu odpowiedniego kierunku prac.

W przypadku Daily Scruma wyraźnie zostaje zaznaczony jego charakter planningowy, planujemy na nim, co jako zespół robimy w ciągu najbliższych 24 godzin, żeby osiągnąć cel sprintu. Jeszcze raz podkreślona została rola celu sprintu, za pomocą rozwinięcia pytania o to, jak nasze działania przybliżają nas do osiągnięcia celu sprintu, i o to, gdzie obecnie jesteśmy w odniesieniu do celu sprintu. Zmiek-

czony zostaje zapis o osobach obecnych na codziennym Scrumie, nie ma już wyraźnego nakazu, aby Scrum Master pilnował, że w spotkaniu udział bierze tylko Zespół Developerski. Teraz jego rola polega na załatwieniu, że zespół ten spotka się w ramach codziennego spotkania.

Czas Sprint Review, a także Retrospekcji, podobnie jak Planningu, zostaje taki jak dla sprintu miesięcznego i jest czasem maksymalnym, co znaczy, że możemy go skrócić, jeśli uważamy, że cel spotkania został osiągnięty.

Product Backlog nie jest już „groomowany”, udoskonalamy go w ramach Product Backlog Refinement, który odbywa się do czasu, aż zadania w Product Backlog będą gotowe do zaplanowania.

Dodana została cała sekcja dotycząca przejrzystości, podkreślająca, że przejrzystość jest konsekwencją zarządzania rzykiem.

Mówiąc krótko, ta zmiana przyniosła nam kolejne wyjaśnienia, ale także zwraca szczególną uwagę na przejrzystość w codziennej pracy oraz maksymalizuje znaczenie celu sprintu i pracy zespołowej. Scrum staje się „lejszy”, ale przez to bardziej wymagający, stąd może następne zmiany?



Rysunek 3. Przejrzystość i skupienie na celu sprintu

## 2013-2016

Po przeczytaniu tej wersji przewodnika stwierdzić możemy, że jedną zmianą, jaka została dokonana, jest dodanie jednego rozdziału: *Scrum Values*, czyli Wartości Scruma. O tym temacie mówiło się już dość długo, dlatego wprowadzenie wartości do oficjalnego przewodnika nie było czymś niespodziewanym.

## Wartości Scruma

Nowy Scrum Guide wprowadza pięć wartości:

**Courage – odwaga, Focus – skupienie się, Commitment – zaangażowanie, Respect – szacunek i Openness – otwartość**

Twórcy poradnika zwracają uwagę na to, że tylko zespoły, których członkowie cechują się tymi pięcioma wartościami, będą w stanie wspierać trzy filary, na których opiera się Scrum – przejrzystość, inspekcję i adaptację, dzięki czemu także będą w stanie osiągnąć stawiane im cele sprintu. Dlatego autorzy podkreślają ich znaczenie, poświęcając im cały rozdział poradnika.

## Odwaga (courage)

Agile wymaga odwagi, odwagi do tego, aby wziąć na siebie odpowiedzialność za osiągnięcie celu sprintu. Zdarzają się sytuacje, że dochodzimy do wniosku, że to, co robiliśmy przez ostatni czas, nie jest najlepszym rozwiązaniem, trzeba mieć dużo odwagi, żeby przyznać się do złego wyboru.

## Skupienie (focus)

Skupienie się na najważniejszym, czyli na tym, co jest teraz i pomoże nam osiągnąć cel sprintu. Bez wykluczenia zbędnych „rozpraszających” nie będziemy w stanie osiągnąć zamierzzonego celu.

## Zaangażowanie (commitment)

Commitment jest często kojarzony ze słowem „zobowiązanie się”, zobowiązujemy się do wykonania wszystkiego, co obiecaliśmy PO, jednak nie bardziej mylnego. Angażujemy się i dokładamy wszelkich starzeń, aby osiągnąć cel sprintu, to jest prawidłowe zrozumienie.

## Szacunek (respect)

Bez szacunku do siebie nie jesteśmy w stanie stworzyć niczego na dłuższą metę, jest on podstawową wartością w zespole. Musimy respektować zdanie swojego kolegi czy koleżanki, nawet jeśli jest odmienne od naszego. Marnowanie środków klienta na działania, które nie prowadzą do stworzenia dobrego produktu, też jest brakiem szacunku, podobnie jak brak transparencji przed PO.

## Otwartość (openness)

Otwartość to nie opowiadanie wokół o wszystkim, ale gotowość na podejmowanie nowych wyzwań czy trudnych rozwiązań, które sprawią, że nasz produkt będzie jeszcze lepszy.

Nie da się ukryć, że zmiana w Scrum Guide jest niewielka, ale tylko jeśli mierzymy ją w centymetrach. Jeśli zastosujemy jednak inną miarę, dostrzeżemy, że podpowiada nam, czego możemy oczekować od naszego zespołu, jakich ludzi zatrudnia do organizacji, jak rozmawiać z jego członkami/członkiniami i jakie relacje powinny między nimi być.



Rysunek 4. Wartości Scruma

## PODSUMOWANIE

Odpowiadając na pytanie: „Jak zmienił się Scrum?” – mam dobre wiadomości. Scrum się nie zmienił, filary, zasady i podstawa, na jakich był budowany od początku, pozostały takie same. Dzięki kolejnym modyfikacjom w przewodniku staje się on czytelniczy i daje więcej swobody w implementacji, co pociąga za sobą większe wymagania w stosunku do ludzi, którzy Scruma wdrażają i używają.

## Źródła

- ▶ Scrum Guide™ 2010-2016 r., Ken Schwaber, Jeff Sutherland
- ▶ Scrum Update 2011-2013 r., Ken Schwaber, Jeff Sutherland
- ▶ <https://goo.gl/8ccF1L>



MICHał CICHY

Absolwent Elektrotechniki i Telekomunikacji oraz Zarządzania Projektami na Politechnice Wrocławskiej. Pasjonat metodik zwiniętych i pracy z ludźmi. Obecnie Scrum Master w Nokia Wrocław.

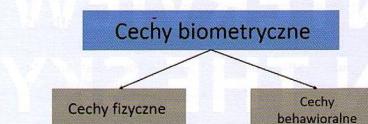
# Rozwiązania biometryczne w identyfikacji użytkowników

W dawnych filmach opowiadających o futurystycznej przyszłości można było niejednokrotnie napotkać na sceny, w których bohaterowie otwierali pomieszczenia poprzez przyłożenie dłoni czy też przykładając oko do skanera. W tamtych czasach takie rozwiązania były określane mianem bardzo odległej przyszłości, która szybko nie nadjeździe. Jednakże tego typu technologie są już dostępne i powoli stają się coraz częściej wykorzystywane. Co więcej, możemy je spotkać nawet w codziennym życiu, na przykład istnieją już bankomaty, które rozpoznają użytkownika poprzez układ żył w jego dloni czy odcisk palca. W niedalekiej przyszłości staną się one swoistym standardem zabezpieczeń, gdyż gwarantują bardzo wysoki procent poprawnych rozpoznań, jak również są proste w użyciu. Takie procedury są w stanie zagwarantować znaczco większy stopień bezpieczeństwa. Jest to bowiem powodowane faktem, że jako użytkownicy systemów biometrycznych nie musimy głosić się o przygotowaniu i zapamiętaniu hasła, jak również nikt nie jest w stanie ukraść naszego hasła, albowiem to sam człowiek i jego cechy są w tym przypadku kluczem.

**B**iometria – to słowo przeciwtemu człowiekowi mówi niewiele, niejednokrotnie na pytanie, czym jest Biometria, słyszę odpowiedź, że jest to jedna z dziedzin Biologii. W pewnym sensie taka odpowiedź nie byłaby pozbawiona racji bytu, jednakże ze względu na to, że Biometria w gruncie rzeczy należy do domeny informatyki i znanego obrazów, należy ją uznać za błędą. Zauważmy bowiem, że etymologia słowa „biometria” wskazuje na języku greckim. Mianowicie rzeczony wyraz możemy podzielić na dwie części – Bios (życie – z j. greckiego) oraz metricos (mierzyć – z j. greckiego). Zatem słowo „Biometria” w dosłownym tłumaczeniu oznaczałoby „mierzyć życie”, natomiast niedosłownie możemy je przetłumaczyć jako „mierzenie ludzkich cech”. Biometria jest uważana za jedną z młodszych gałęzi nauki. Jednakże jest to nieprawda. Kiedy bowiem sięgnęliśmy do źródeł historycznych, okaże się, że takowa nauka (oczywiście nie pod dobrze znaną nam wspólnie nazwą) była używana już w starożytności. Pierwsze źródła pochodzą bowiem sprzed 31 tysięcy lat przed Chrystusem, są to odciski dloni umieszczone przy malowidłach naskalnych. Według badaczy takie ślady miały bowiem za zadanie odróżnić autorytów poszczególnych dylegancji. Kolejnym interesującym śladem Biometrii są gliniane tabliczki pochodzące z Starożytnego Babilonu, które to są datowane na 500 lat przed Chrystusem. Na tychże tabliczkach znajdują się bowiem odciski ludzkich palców. Według historyków miały one służyć za potwierdzenia dokonywanych transakcji handlowych. Analogiczne zastosowanie odcisków palca mogliśmy również zaobserwować w starożytnych Chinach, ponadto rodzice wielodzietnych rodzin wykorzystywali odciski dloni oraz stop do odróżnienia swoich dzieci. Biometria miała również swoje zastosowanie w starożytnym Egipcie. W kraju faraonów opis fizycznych cech pozwalał na odróżnienie zaufanych sprzedawców na targu od tych, którzy byli nowymi handlarzami. Reasumując ten fragment historii, możemy skonkludować, że w gruncie rzeczy Biometria

była wykorzystywana w przypadku transakcji handlowych oraz w procesach identyfikacyjnych. Pierwszy prawdziwy system biometryczny jest datowany na 1870 rok. Jego autorem był francuski antropolog Alphonse Bertillon. Jego system był bowiem oparty o dokładny opis ciała ludzkiego bazujący na fizycznych deskryptorach i zdjęciach. Zaproponowana przez niego metoda została jednakże szybko uznana za zbyt trudną do użycia, jak również poddana pod wątpliwość jej skuteczności. W kolejnych latach, aż do czasów współczesnych, zostało zaproponowanych wiele zróżnicowanych systemów biometrycznych. Oczywiście niektóre z nich cechują się wysoką rozpoznawalnością człowieka, a inne są praktycznie nieużywalne. Współcześnie interesującym podejściem są systemy fuzyjne (nazywane również multimodalnymi czy hybrydowymi), które to łączą zróżnicowane cechy biometryczne. Przejedźmy zatem do krótkiego omówienia klasyfikacji cech biometrycznych oraz przedstawmy kilka przykładów dla każdej z grup.

## CECHY BIOMETRYCZNE



Główny podział cech biometrycznych opiera się o podział na cechy fizyczne oraz cechy behawioralne. Pierwszą z wyżej wymienionych grup jest opisywana jako zestaw zawierający te cechy, które to

## ROZWIĄZANIA BIOMETRYCZNE W IDENTYFIKACJI UŻYTKOWNIKÓW

człowiek maływa po urodzeniu i są one kształtowane samoistnie przez cały okres jego życia (albo tylko przez pewien okres, na przykład tęczówka oka kształtuje się przez pierwsze dwa lata życia, po tym czasie jest praktycznie niezmienna), chyba że przez pewne względy chorobowe. Mianowicie możemy tutaj zakwalifikować takie cechy jak: odciśn palca, tęczówka, siatkówka czy geometria dłoni i układ żył w dłoni. Z kolei dla drugiej grupy, czyli cech behawioralnych, możemy zaliczyć takie cechy jak: szybkość pisania na klawiaturze, chód czy sposób, w jaki mrużymy. Należy bowiem zauważyć, że ta grupa może zostać scharakteryzowana jako zespół cech, które to mogą być rozwiązywane przez całe życie, jednakże nie rozwiązywać się one w sposób samoistny, a poprzez bezpośrednią interwencję człowieka. Innymi słowy są to cechy, których możemy się nauczyć i doskonalić je przez cały okres naszego życia. Koronnym przykładem może być podpis. Każdy z nas wykonuje swoją własną sygnaturę w inny sposób i nawet pomimo upływu lat nadal podpis nosi pewne znaczenia, które pozwalają na odróżnienie danego autora od innej osoby. Niejednokrotnie spotkać się można na pytaniu, które cechy powinno się wykorzystać w systemie biometrycznym w celu uzyskania dużego bezpieczeństwa. Należy zauważyć, że dobor cech powinien zależeć od pewnych warunków. Mianowicie jedna z elementarnych kwestii, którą powinniśmy rozważyć, jest prosta użycia systemu przez potencjalnych użytkowników. Rozwijamy bowiem przykład aplikacji internetowej, która wykorzystywałaby pewne cechy biometryczne do identyfikacji użytkowników. Czy dobrym pomysłem byłoby zapewnienie bezpieczeństwo poprzez wykorzystanie układu żył w dłoni? Oczywiście, że tak pomysł bardzo szybko zostałby skrytykowany, a użytkownicy takiej aplikacji zapewne przestaliby ją użytkować. Nie możemy bowiem wymagać od użytkownika, aby nabywał dodatkowe urządzenie (które to ponadto musiałoby być kompatybilne z naszym rozpoznaniem), z użyciem którego mógłby być prawidłowo rozpoznawany w naszym systemie. Znaczco lepszym rozpoznaniem w przypadku takiej aplikacji byłoby użycie chociażby obrazu twarzy. Takie podejście byłoby bardziej efektywne ze względu na to, że praktycznie każdy użytkownik dysponuje kamerą internetową (czy to w laptopie, czy też jako oddzielnego modułu) i wykonanie zdjęcia byłoby bardzo proste. Jednocześnie istnieje możliwość zapewnienia znaczco większego bezpieczeństwa użytkownikowi chociażby poprzez sprawdzenie zgodności twarzy – w takim przypadku oszustwo z wykorzystaniem zdjęcia byłoby niemożliwe. Kolejnym ważnym elementem, który powinien

## LUDZKA TWARZ JAKO ELEMENT SYSTEMU IDENTYFIKACJI

Ludzka twarz jest jedną z najczęściej wykorzystywanych cech biometrycznych. Mianowicie charakteryzuje ją prostota pobrania, szybkość przetwarzania oraz niski współczynnik fałszywych akceptacji. Jest to powodowane tym, że istnieje niski prawdopodobieństwo (oczywiście wykluczając casus bliżniaczy jednojednorodowych), że istnieje druga osoba o idealnie tych samych deskryptorach fizycznych twarzy. W tej materii zostały poczynione znaczące badania, które zaowocowały tak popularnymi rozpoznaniami jak Eigenface czy Fisherface. W ramach tego artykułu, odwołując się do materiałów źródłowych, omówimy każdą z tych technik. Następnie postaramy się przedstawić fragmenty kodu, które pozwolą czytelnikowi na implementację takich algorytmów (z wykorzysta-

## FELIETON

niem biblioteki Java CV), i zwrócimy uwagę na pewne fakty związane z implementacją. Pierwszym algorytmem, który omówimy, będzie algorytm Eigenface.

### Algorytm Eigenface

Algorytm Eigenface w kontekście rozpoznawania człowieka został zaproponowany przez M. Kirby'ego i L. Sirovicha w 1987 roku, jednakże pełną implementację tego rozwiązania (oczywiście w ramach wspomnianego wcześniej kontekstu) zaproponowali Matthew Turk i Alex Paul Pentland w artykule [1]. W głównej mierze ten algorytm opiera się o analizę głównych składowych (PCA). Jako pierwszy krok tego rozwiązania wymieniana jest konstrukcja macierzy, w której wektorem pionowym są wszystkie obrazy. Oznacza to, że każdy wektor składa się z wartości wszystkich pikseli obrazu. Już w tym momencie warto zwrócić uwagę na to, że wszystkie obrazy (zarówno znajdujące się w bazie danych, jak i wejściowe) powinny mieć dokładnie te same wymiary. Następnym krokiem jest wyznaczenie wartości uśrednionych dla każdego wektora poziomego macierzy i ustalenie w ten sposób pionowego wektora średniego. Jako trzeci krok wymienia się jego odejmowanie od każdego z wektorów pionowych macierzy wektora średniego. Następnie na podstawie wyznaczonej macierzy wektorów pionowych  $M$ , symbolizujących wszelkie obrazy w bazie danych, wyznaczana jest macierz kowariancji  $C$ . Ten krok jest wykonywany zgodnie ze wzorem (1).

$$C = M \cdot M^T$$

Gdzie  $M^T$  jest transpozycją macierzy  $M$ . W kolejnym etapie wyznaczane są wartości własne dla każdego z wektorów pionowych macierzy. Aby uzyskać odpowiednio wartości, należy skorzystać ze wzoru (2).

$$C \cdot v_i = \lambda_i \cdot v_i$$

Gdzie  $v_i$  jest wartością własną macierzy  $C$ , natomiast  $\lambda_i$  jest wektorem własnym macierzy. W przypadku klasyfikacji, która to jest głównym celem użycia tegoż algorytmu, odbywa się ona praktycznie w dwóch krokach. Pierwszym z nich jest obliczenie wektora wartości własnych dla macierzy stanowiącej reprezentację wszystkich obrazów z bazy danych, natomiast drugim krokiem jest wyznaczenie wartości własnych dla nowego obrazu i dokonanie klasyfikacji. Najczęściej stosowanym algorytmem klasyfikacyjnym (jednocześnie najprostszym) jest algorytm k-Najbliższych Sąsiadów.

Innymi słowy algorytm zaproponowany przez Turka i Pentlanda opiera się o fakt, że każda twarz może być opisana poprzez pewien niezerowy zestaw charakterystycznych widoków dwuwymiarowych. Obrazy twarzy są w ramach tego rozpoznania rzutowane na pewną przestrzeń cech, która jest nazywana przestrzenią twarzy. Z kolei na rzeczną przestrzeń składają się „twarze własne”, które są definiowane przez wektory własne zestawu twarzy. Oczywiście zadane wektory nie muszą odpowiadać wyizolowanym cechom ludzkim – na przykład oczom czy uszom. Dane podejście wykazuje pewne zalety w stosunku do innych rozwiązań – na przykład szybkość porównania czy też niewielki wpływ na wydajność systemu. Metoda ta jest głównie wykorzystywana ze względu na prostotę przetwarzania obrazów twarzy oraz z uwagi na fakt, że porównanie pomiędzy wektorami cech jest bardzo proste i szybkie.

Przede wszystkim jednakże do najważniejszego elementu, czyli kwestii implementacyjnej, która to w tym przypadku jest trywialna. Skorzystamy bowiem z dostępnej nam biblioteki Java CV.

Listing 1. Przykładowa konfiguracja face recognizera i klasyfikacja przykładowej próbki

```
getImages();  
FaceRecognizer faceModel = createEigenFaceRecognizer(14,Double.  
POSITIVE_INFINITY);  
Mat[] imagesArray = new Mat[this.images.size()];  
imagesArray = this.images.toArray(imagesArray);  
int[] labelsArray = toArray(this.labels);  
Mat matLabels = new Mat(new IntPointer(labelsArray));  
MatVector imagesVector = new MatVector(imagesArray);  
faceModel.train(imagesVector, matLabels);  
int predictedLabel = faceModel.predict(testImage);  
return predictedLabel;
```

W ramach pierwszej linijki kodu przedstawionego w Listingu 1 możemy zaobserwować metodę `getImages`. Dана metoda jest wykorzystywana do pobrania obrazów z bazy danych – mianowicie opiera się ona o odczytanie informacji o ścieżce do pliku i klasie użytkownika (w tym przypadku loginu) z pliku XMLowego. Odczytane obrazy są dodawane do listy o nazwie `images`, natomiast informacje o klasie użytkownika trafiają na listę `labels`. Kolejność dodawania rekordów do obydwiu list jest bardzo ważna, gdyż jej zamiana może spowodować błąd w identyfikacji użytkownika. W kolejnym kroku tworzymy nowy face recognizer, ustawiając liczbę twarzy własnych na 14 oraz ustawiając wartość progu na nieskończoność. Następnie przygotowujemy odpowiednią strukturę zawierającą zdjęcia i informacje o klasie każdego ze zdjęć i dopiero po zweryfikowaniu poważności utworzonej takich struktur wykonujemy trening face recognizera. W kolejnym kroku podajemy nowy obraz do zaklasyfikowania. Poprzedź użycie metody `predict` jesteśmy w stanie uzyskać informację o klasie przedstawionej próbki.

### Algorytm Fisherface

Inne podejście jest prezentowane w ramach algorytmu Fisherface [2]. Jak mogliśmy zaobserwować w ramach algorytmu Eigenface, jego podstawowym celem było znaksmalizowanie wartości wariancji w ramach naszej bazy danych (tak by próbki były znacznie różne od innych), natomiast w przypadku podejścia zaproponowanego w ramach algorytmu Fisherface za cel stawiana jest maksymalizacja średniego dystansu pomiędzy różnymi klasami oraz zmniejszanie wariancji wewnętrzklasowej. Sposobem, który doprowadza do uzyskania takich warunków, jest użycie liniowej metody znanej pod nazwą liniowej analizy dyskryminacyjnej (Fisher's Linear Discriminant – FLD). W przypadku algorytmu Fisherface możemy zauważać, że jego implementacja byłaby w gruncie rzeczy analogiczna do tej, która została zaprezentowana w ramach Listingu 1, wymagałaby ona jednak jednej drobnej modyfikacji, którą zaprezentowano w Listingu 2.

Listing 2. Klasyfikacja z wykorzystaniem algorytmu Fisherface

```
getImages();  
FaceRecognizer faceModel = createFisherFaceRecognizer(14,Doub  
le.POSITIVE_INFINITY);  
Mat[] imagesArray = new Mat[this.images.size()];  
imagesArray = this.images.toArray(imagesArray);  
int[] labelsArray = toArray(this.labels);  
Mat matLabels = new Mat(new IntPointer(labelsArray));  
MatVector imagesVector = new MatVector(imagesArray);  
faceModel.train(imagesVector, matLabels);  
int predictedLabel = faceModel.predict(testImage);  
return predictedLabel;
```

Szkolenie dla Ciebie lub Twojego zespołu

## Inżynieria odwrotna i techniki zabezpieczania aplikacji na platformie Android

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages włącznie przy zamówieniach złożonych do końca marca 2017 r.  
Hasło: PROGRAMISTAMAG



www.sages.com.pl

## ROZWIĄZANIA BIOMETRYCZNE W IDENTYFIKACJI UŻYTKOWNIKÓW

Jedyną różnicą, jaką wprowadziliśmy w stosunku do ówcześnie zaprezentowanej implementacji algorytmu Eigenface, była zmiana recognizera poprzez zastosowanie Fisherface recognizera, który jest przypisywany do zmiennej faceModel1.

Niejednokrotnie można spotkać się z pytaniem, który z tych dwóch algorytmów jest lepszy. Oczywiście tak jak w przypadku wielu różnych rozwiązań biometrycznych, najczęściej nie istnieje dokładna odpowiedź. Zależy to bowiem od wielu czynników, które zostały wymienione wcześniej. Zauważmy bowiem, że również może być to także zależne od ilości użytkowników czy też miejsca zastosowania. Oczywiście obydwie metodyki opisane w ramach tego artykułu mogą czynnikom posłużyć jedynie do przeprowadzania prostych, wstępnych eksperymentów dotyczących rozpoznawania użytkownika z wykorzystaniem obrazu twarzy. Aby móc stosować je w ramach systemów wymagających bardzo dokładnych procedur bezpieczeństwa, należałoby wprowadzić pewne modyfikacje w obydwu rozwiązaniach.

## IDENTYFIKACJA Z WYKORZYSTANIEM SZYBKOSCI PISANIA NA KLAWIATURZE

Wiełokrotnie, kiedy wspomniałem – czy to swoim kolegom czy też na spotkaniach Koła Naukowego Biometrii (działającego na Politechnice Białostockiej) – o tym, że taka cecha jak szybkość pisania na klawiaturze jest w stanie zapewnić nam jako programistom możliwość adekwatnego zabezpieczenia naszych aplikacji, często spotykałem się z niedowierzeniem i pytaniem, czy nie żartuję. Otóż nie jest to żart, a rzeczywisty fakt. Sam miałem okazję uczestniczyć w badaniach [3], które to potwierdzają skuteczność klasyfikacji z wykorzystaniem takiej cechy. Jak zatem wykonać tego typu zabezpieczenie oraz na co zwrócić uwagę? Omówimy to w ramach części artykułu.

Analizę takiego podejścia powinniśmy rozpoczęć od zastawienia się nad postacią wektora cech. Wedle wszelkich standardów musi stanowić on unikalowy wzorzec, który to w sposób jednoznaczny będzie identyfikował użytkownika. Oczywiście możemy wziąć pod uwagę cały zakres alfabetu (najlepiej angielskiego – gdyż jest najbardziej uniwersalny i nie występują w nim żadne dodatkowe znaki lokalne), aczkolwiek aby móc to wykorzystać

cech? Jak łatwo możemy skonkludować, może to być niedostateczne rozwiązanie. Dlaczego? Otóż jeżeli pozostawimy dowolność użytkowniku w kwestii wyboru wpisywanego zdania, może on nie pokryć całego alfabetu, co z kolei może skutkować błędym rozpoznaniem. W związku z tym powiniśmy skoncentrować się na pewnej zamkniętej puli znaków albo jednoznacznie określić zdanie, które użytkownik powinien wpisać podczas logowania (najlepiej aby takie zdanie pokryło całkowicie wybraną część alfabetu albo cały alfabet). Kolejnym elementem, który należałoby rozważyć, są tzw. zrosty – w przypadku polskiego alfabetu jest ich kilka, np. sz, cz, ch itd. Jak wykazały badania, takie zrosty również pozwalają na określenie, z pewnym stopniem pewności, tożsamości użytkownika. Powinny zatem one stanowić kolejną część naszego wektora cech. Ponadto należy się zastanowić, czy nie warto by było wziąć pod uwagę całkowitego czasu pisania danego tekstu czy też odstępów pomiędzy kolejnymi wpisymi znakami. Ważną kwestią, która programista takiego rozwiązania musi uwzględnić, jest czas wcisnięcia klawisza przez długi czas oraz jednocześnie wcisnięcie drugiego klawisza w trakcie trwania wcisnięcia klawisza pierwszego. Reasumując kwestię wektora cech, należy skonkludować, że musi być on skonstruowany na tyle poprawnie, aby móc na jego podstawie w sposób jednoznaczny zidentyfikować użytkownika. Jak wykazały zróżnicowane badania, szybkość pisania na klawiaturze przy niepoprawnie przygotowanym wektorze cech charakteryzuje się wysokim współczynnikiem FAR.

Równie ważnym elementem jak budowa wektora cech jest sposób, w jaki będziemy mierzyć odległość pomiędzy nową próbką a wszystkimi próbками należącymi do bazy danych. W literaturze najczęściej wymienia się dwa najprostsze sposoby, czyli wykorzystanie odległości Euklidesowej przedstawionej we wzorze (3) lub odległości Manhattan zaprezentowanej w ramach wzoru (4).

$$distance = \sqrt{\sum_{i=0}^N ((v_i - x_i)^2)}$$

$$distance = \sum_{i=0}^N |v_i - x_i|$$

## FELIETON

W przypadku wzorów (3) i (4) wartości  $v_i$  oraz  $x_i$  symbolizują i-te pozyycje w ramach wektorów cech, dla uproszczenia przyjmujemy, że w odpowiadającej nowej próbce, natomiast  $x$  symbolizuje próbkę z bazy danych. W celach klasyfikacyjnych możemy skorzystać ze zróżnicowanych technik, na przykład możemy użyć techniki programowania, czyli takiej, w której pierwsza próbka poniżej pewnego progu będzie uznana za prawidłową i klasyfikacja zostanie zakończona. Inną techniką, która daje znacząco lepsze efekty, jest wspominany już w ramach tego artykułu algorytm k-Najbliższych Sąsiadów.

Jednocześnie biorąc pod uwagę cechy, jaką jest szybkość pisania na klawiaturze, należy zauważać, że w przypadku dużych baz danych, obejmujących znaczącą liczbę użytkowników, taka cecha może być brana pod uwagę jedynie jako cecha pomocnicza. Mianowicie inne badania, które również zostały przeprowadzone, wskazały, że u osób, które mają codzienny kontakt z komputerem (inny sposób duzo korzystają z komputera, a co za tym idzie – również z klawiatury), czas pisania zróżnicowanych tekstów mogą różnić się w sposób nieznaczny.

Niewątpliwym plusem tego rodzaju identyfikacji jest to, że może być on szybko i efektywnie zaimplementowany.

## CECHY NIESTANDARDOWE

Do cech niestandardowych z całkowitą pewnością możemy zaliczyć chociażby rytm bicia serca. Choć może się to wydać dosyć dziwne, jednakże przeprowadzone badania oraz wyniki uzyskane przez ich autorów wskazują na to, że niezależnie od tego, czy mówimy o EKG spoczynkowym czy też EKG wysiłkowym, pewne cechy wskazujące na daną osobę mogą zostać zaobserwowane. Niewątpliwe może być to przyszłość identyfikacji biometrycznej, jednakże tylko i wyłącznie w sytuacji, w której zostanie dostarczony sprzęt, który to będzie małoawaryjny oraz będzie działał na tyle efektywnie, że nie będzie wymagały zbyt długiego czasu na pobranie próbki. Aktualnie ta cecha w kontekście rozpoznawania biometrycznego jest wciąż w fazie badań.

Inną interesującą cechą jest chociażby układ żył w dłoni. Udomioniono bowiem, że każdy człowiek ma inne ułożenie żył w dloni.

ni. Taka cecha jest prosta do pobrania, jednakże jej przetworzenie wymaga pewnych, nierazdrożnych, procedur. Należy tutaj również zauważać, że niestety, ale taka cecha może łatwo zostać sfalszowana. Jak zostało wykazane, do jej spreparowania wystarcza chociażby sztuczny śnieg. Aktualnie możemy znaleźć również bankomaty, które opierają się o wykorzystanie tej cechy.

Ostatnią cechą, na którą chciałbym zwrócić uwagę, jest sposób, w jaki zginamy palec wskazujący. Mianowicie każdy z nas robi to w inny sposób. Jedno z rozwiązań opisane w [7] proponuje wykorzystanie linii zakończonych poprzez ruch palca do identyfikacji człowieka. Jednocześnie brane są pod uwagę trajektorie zakończenia przez trzy różne palce. W tym artykule został również zaproponowany sposób pobierania danych. Co ciekawe, pozwala to również na przeprowadzenie identyfikacji.

Abstrahując od cech biometrycznych, należy również zauważać, że interesując jest połączenie Biometrii i analizy obrazów medycznych. Zauważmy bowiem, że nierzadko w sytuacji, w której nie możemy pobrać pewnej cechy, a byliśmy z jej użyciem zarejestrowani, powinniśmy rozważyć wizytę u lekarza. Szczególnie ważne jest to w przypadku chociażby siatkówki oka, kiedy bowiem siatkówka jest przesłonięta – co może być powodowane pewnymi symptomami chorobowymi – będzie ona niemożliwa do pobrania. Oczywiście wizyta u specjalisty byłaby wskazana tylko i wyłącznie w przypadku cech, których nie możemy naocznie sprawdzić – problem z pobraniem odcisków palca nie musi wskazywać na chorobę, a na przykład być powodowany pewnymi uszkodzeniami mechanicznymi.

## PODSUMOWANIE

W ramach tego artykułu przedstawiłem pokrótkie, czym jest Biometria, oraz omówiłem wybrane cechy biometryczne, jak również wskazałem pewne algorytm, które mogą być użyte do identyfikacji użytkownika. Oczywiście zachęcam czytelników do samodzielnego oglądania meandrów Biometrii. W razie jakichkolwiek pytań zapraszam również do kontaktu.

## Bibliografia

- [1] M. Turk, A. Pentland – „Face recognition using eigenfaces” – Computer Vision and Pattern Recognition, 1991, Proceedings CVPR 1991, IEEE Computer Society Conference on 1991.
- [2] V. More, A. Wagh – „Improved Fisher Face Approach for Human Recognition System using Facial Biometrics”, International Journal of Information and Communication Technology Research, Volume 2 – No. 2, 2012, p. 135 - 139
- [3] P. Panasuk, M. Szymkowski, M. Dąbrowski, K. Saeed – „A Multimodal Biometric User Identification System Based on Keystroke Dynamics and Mouse Movements”, K. Saeed, W. Homenda – „Computer Information Systems and Industrial Management, 15th IFIP TC8 International Conference, CISM 2016, Vilnius, Lithuania, September 14-16, 2016, Proceedings”.
- [4] R. Bolle, J. Connell, S. Pankanti, N. Ratha, A. Senior – „Biometria”, Wydawnictwa Naukowo-Techniczne, Warszawa 2008, ISBN: 978-83-204-3332-6
- [5] <http://www.biometricupdate.com/201501/history-of-biometrics> (dostęp 31.01.2017)
- [6] <http://www.graileubiotmetrics.com/en-us/book/understanding-biometrics/introduction/history> (dostęp 31.01.2017)
- [7] M. Kamijo, Y. Shimizu – „Kansei Measurement in Cooperative Development Techniques”, K. Saeed, T. Nagashima – „Biometrics and Kansei Engineering”, Springer, New York 2012, ISBN: 978-1-4614-5607-0

MACIEJ SZYMKOWSKI

szymkowskimack@gmail.com

Inżynier informatyk, Junior Java Developer w Transition Technologies PSC, student Wydziału Informatyki Politechniki Białostockiej. Do swoich zainteresowań zalicza: Biometrię, Analizę Obrazów i Sygnałów oraz Elektronikę i Kryptografię. Uwielbia poszerzać swoją wiedzę poprzez uczestnictwo w konferencjach oraz lekturę książek i artykułów. Programuje w językach: Java, C#, C++ oraz C.

## Idealne dopasowanie

W gospodarce opartej na wiedzy głównym źródłem strategicznej przewagi jest zdolność do uczenia się szybciej niż konkurencja oraz umiejętność zarządzania wiedzą w sposób efektywny. W branży nowych technologii najważniejszym kapitałem są ludzie, których talenty mogą doprowadzić firmę do sukcesu lub do porażki w zależności od zespołów i organizacji, w których przyszło im pracować. Właśnie dlatego kluczowym elementem do odniesienia sukcesu jest znalezienie odpowiedniego zróżnicowania i dopasowania zarówno w kontekście zespołu, jak i organizacji. W tym artykule przedstawię ciekawy punkt widzenia na zespoły zadowalowe, drużyny akcyjne, omówię różne kolory organizacji oraz zwróci uwagę na kolor turkusowy, w szczególności pod kątem startupów.

### ZRÓŻNICOWANIE I DOPASOWANIE

Każdy z nas jest inny i ma inne predyspozycje, dlatego pierwszym elementem, na jaki chciałbym zwrócić uwagę, jest kwestia zróżnicowania i dopasowania. W książce „Po pierwsze: złam wszelkie zasady” [1] przedstawione zostały cztery klucze potrzebne do odniesienia sukcesu:

1. Wybierz pod kątem talentu,
2. Określ oczekiwane rezultaty,
3. Skoncentruj się na mocnych stronach,
4. Znajdź odpowiednie dopasowanie.

Try pierwste klucze mają charakter bardziej indywidualny, natomiast w celu znalezienia ostatniego klucza, czyli odpowiedniego dopasowania, należy wziąć pod uwagę odpowiednie dopasowanie pod kątem stanowiska, pod kątem zespołu oraz pod kątem organizacji. Jeżeli chodzi o dopasowanie pod kątem stanowiska, jest to kwestia w miarę prosta. Zazwyczaj wystarczy sprawdzić kwalifikacje i doświadczenie w stosunku do oczekiwania na danym stanowisku. Schody zaczynają się w momencie próby dopasowania do specyfiki zespołu czy w szerszym kontekście do kultury organizacji, co długoterminowo jest kluczowe do osiągnięcia sukcesu.

Zanim poruszę kwestię dopasowania do zespołu, chciałbym zwrócić uwagę na różnicę między zespołem zadaniowym a drużyną akcyjną. **Zespół zadaniowy** to zespół ludzi posiadający wspólny cel powołanych razem do wykonywania określonych zadań w dłuższej perspektywie. W sytuacjach, w których jakieś zadanie trzeba wykonać wyjątkowo sprawnie, a koszt popełnienia błędu jest wysoki, dany zespół zadaniowy przekształca się w **drużynę akcyjną**. Są to dość szczególnie sytuacje będące chwilą prawdy o zespole i jego prawdziwej wartości.

Aby zespół zadaniowy mógł efektywnie działać jako drużyna akcyjna, powinien opierać się na zasadzie zróżnicowania. W szczególności powinien zaważyć w swych szeregach osoby o predyspozycjami **analityka, wykonawcy, przyjaciela i pomysłodawcy**. Wszystkie te cztery style komunikacji i typy zachowania powinny być obecne, aby zespół zadaniowy mógł działać efektywnie jako drużyna akcyjna.

Analityk skrupulatnie zbiera informacje, prezentuje argumenty i dostępne opcje. Przywiązuje dużą wagę do faktów, poszukuje danych liczbowych, łączy fakty. Jest usystematyzowany, dobrze zor-

ganizowany we wszystkim, co robi. Płaszczyzna **racji** jest w jego przypadku dominująca.

Wykonawca zabiiera się bez chwili wahania do działania, liczy na szybkie rezultaty, najpierw działa, potem myśli. Jest praktyczny, zorientowany na „tu i teraz”. Przekształca abstrakcyjne pomysły w konkrete działania. Płaszczyzna **akcji** jest w jego przypadku dominująca.

Przyjaciel przywiązuje dużą wagę do relacji międzyludzkich. Pełen empatii dba o odpowiednią atmosferę w zespole. Chce, żeby wszyscy czuli się w nim dobrze. Zwraca uwagę na emocje i stara się rozładować napięcia. Płaszczyzna **relacji** jest w jego przypadku dominująca.

Pomysłodawca jest wizjonerem z wyobraźnią, który widzi świat pełen możliwości. Często stawia uznane prawdy pod znakiem zapytania. Generuje pomysły i dzięki temu wskazuje kierunek, w którym zespół powinien podążać. Płaszczyzna **inspiracji** jest w jego przypadku dominująca.

W Tabeli 1 przedstawiłem nieco żartobliwe przykłady dwóch skutecznie działających drużyn akcyjnych.

### KOLORY ORGANIZACJI

Istotą każdej organizacji jest łączenie się grupy ludzi w ramach ustalonej struktury w celu osiągania określonych celów. Analizując historię cywilizacji oraz rozwój organizacji na przestrzeni wieków, można wyodrębnić kilka etapów rozwoju ludzkiej świadomości i odpowiadające im modele organizacyjne. Każdemu takiemu etapowi czy też poziomowi rozwoju przypisany został odpowiadający mu kolor. Na podstawie książki „Pracować inaczej” [2] rozróżniamy pięć podstawowych kolorów organizacji:

- » czerwony
- » bursztynowy
- » pomarańczowy
- » zielony
- » turkusowy

Kolory organizacji oraz odpowiadające im metafore i przykłady są zestawione w Tabeli 2. W Tabeli 3 natomiast zebrane są przełomowe idee i charakterystyczne cechy dla danego koloru.

W dużej organizacji mogą występować obok siebie różne kolory, zazwyczaj jednak jeden kolor jest dominujący. Chciałbym przy tym zaznaczyć, że żaden z kolorów nie jest lepszy ani gorszy od pozostałych.

Styl komunikacji i typ zachowania	Pingwiny z Madagaskaru	Wojownicze Żółwie Ninja
Analityk	Kowalski – mózg zespołu, zawsze proponuje kilka opcji wyjścia z danej sytuacji, choć ma tendencję do przesadzania w swojej analizie, wynalazca różnych gadżetów, z których często jest wilej kłopotów niż pozytyku	Donatello – mózg zespołu, technologiczny geniusz, kieruje się inteligencją i rozumem, zawsze najpierw myśli, zanim coś zrobi, inżynier i wynalazca, preferuje rozwiązania przy użyciu intelektu niż sily mięśni
Wykonawca	Rico – specjalista od broni i materiałów wybuchowych, polityka i wypływa różne przedmioty, jest gotowy do działania jako pierwszy, zawsze chętny, aby cos wysadzić w powietrze	Raphael – impulsywny, waleczny, najsilniejszy z żółwi, preferuje działać i atakować jako pierwszy bez chwili zastanowienia, pomimo swojego trudnego charakteru jest silnie związany ze swoimi braćmi
Przyjaciel	Szeregowy – emocjonalnie wrażliwy młodszy rekrut, ma mniejsze doświadczenie od pozostałych, więc czasami popefnia błędę, przez co staje się tematem do żartów, dobrze relacje i pozytywne emocje w zespole	Michelangelo – żółw z wielkim poczuciem humoru, lekkochudły, wyluzowany, uwielbia pizzę, która dzieli się z pozostałymi żółwiami, często rozmiesza całą resztę, ale też nie raz wędruje wśród wszystkich w kłopoty
Pomysłodawca	Skipper – przywódca pingwinów, nie ma dla niego rzeczy niemożliwych, zawsze ma pomysł na nową akcję, w sytuacjach nietypowych pyta o dostępne opcje, sam jednak podejmuje decyzje i wydaje rozkazy	Leonardo – odważny i honorowy żółw, przywódca całej czwórki, nazwany na cześć wielkiego wizjonera Leonardo Da Vinci, często podejmuje decyzje i dźwiga brzemię odpowiedzialności za cały zespół

Tabela 1. Przykłady skutecznie działających drużyn akcyjnych

Kolor	Metafora	Przykłady
Czerwony	Watąha wilków	mafia, gangi uliczne, organizacje terrorystyczne
Bursztynowy	Armia	wojsko, kościoły, agencje rządowe, szkoły publiczne
Pomarańczowy	Maszyna	globalne korporacje, banki inwestycyjne, prywatne uniwersytety
Zielony	Rodzina	organizacje pozarządowe, ruchy społeczne, organizacje szczupłe i zwinne (ang. <i>lean and agile</i> )
Turkusowy	Żywy organizm	pionierskie organizacje, innowacyjne startupy

Tabela 2. Kolory organizacji

**devstyle.pl**

Blog dla każdego programisty

FELIETONY TESTY DEPENDENCY INJECTION GIT

...i wiele więcej

## KLUB LIDERA IT

Kolor	Przelomowe idee	Charakterystyczne cechy
Impulsywny Czerwony	dowodzenie i kontrola (ang. <i>command and control</i> ) podział pracy (ang. <i>division of labor</i> )	autorytet oparty na sile fizycznej, impulsowność przywódcy, bajeczne nagrody, drakonische kary, chaos, niepewność jutra, zarządzanie przez strach (ang. <i>management by fear</i> )
Konformistyczny Bursztynowy	formalne role i hierarchia (ang. <i>formal roles and hierarchy</i> ) powtarzalne procesy (ang. <i>repeatable processes</i> )	sztynna i skalowana hierarchia, stabilny porządek, wydawanie poleceń i kontrola ich wykonania, ślepe przestrzeganie zasad i wytycznych, konformizm, podział na to, co dobre i złe, moralność zasad jako podstawa do podejmowania decyzji, używanie „kija”
Osiągający Pomarańczowy	innowacja (ang. <i>innovation</i> ) merytokracja (ang. <i>meritocracy</i> ) odpowiedzialność (ang. <i>accountability</i> )	konkurencja, zysk, zarządzanie przez cele (ang. <i>management by objectives</i> ), ocena pracowników, systemy premiove, proces budżetowania, nagrody za jakość, nacisk na strategię i osiąganie wyników, skuteczność działania jako podstawa do podejmowania decyzji, używanie „marchewki”
Pluralistyczny Zielony	równowaga udziałowców (ang. <i>stakeholder balance</i> ) kultura oparta o wartości (ang. <i>value-driven culture</i> ) upelnomocnienie (ang. <i>empowerment</i> )	zadowolenie klienta, współdzielone wartości, zaangażowanie, pragmatyzm, podział na to, co działa, i na to, co nie działa, harmonijne relacje międzyludzkie często ważniejsze od osiągania celów, pełny egalitarystyczny prowadzący czasami do impasu decyzyjnego
Ewolucyjny Turkusowy	samozarządzanie (ang. <i>self-management</i> ) dążenie do spełnienia (ang. <i>wholeness</i> ) ewolucyjne przeznaczenie (ang. <i>evolutionary purpose</i> )	plaska struktura, samoorganizacja, nadzenna rola misji, inspirowająca wizja, poczucie wewnętrznej spójności, rozproszone podejmowanie decyzji, całoszczęśliwe podejście oparte w pełni na człowieku

Tabela 3. Przelomowe idee i charakterystyczne cechy dla danego koloru organizacji

tych. Wszystkie są korzystne w zależności od posiadanego perspektywy, aktualnych potrzeb i złożoności danego przedsięwzięcia.

Znajomość własnego preferowanego stylu pracy pozwala określić, jakie mamy szansę odnaleźć się i dopasować do kultury danej organizacji. Warto przy tym zauważać, że kolory mogą ulegać zmianie, jest to jednak proces długotrwały, wymagający ponadprzeciętnych zdolności przywódczych osób zarządzających takim organizacjami.

## TURKUSOWE ORGANIZACJE

Na koniec chciałbym zwrócić uwagę na ostatni z wymienionych kolorów organizacji – kolor turkusowy. W tego typu organizacjach dużą szansę na odnalezienie się i spełnienie swoich potrzeb mają osoby posiadające wysokie kwalifikacje i szerokie kompetencje, które cenią sobie niezależność, chcące być pionierami w swojej dziedzinie.

Decyzje w tego typu organizacjach podejmowane są według wyjątkowo prostej zasady: „Decydują ci, którzy wiedzą, a reszta ma do nich zaufanie”. W turkusie nikomu nie wydaje poleceń, a zakres odpowiedzialności każdego pracownika można opisać w czterech prostych zasadach:

1. Robisz to, co potrafisz,
2. Robisz to, co jest potrzebne,
3. Bierzesz odpowiedzialność za to, co robisz,
4. To, co robisz, zawsze możesz zmienić, ale z zachowaniem zasad: 1., 2. i 3.

W takiej organizacji nikt nie powie: „To nie należy do moich obowiązków”, a raczej zapyta: „Czym powinieneś się najpierw zajęć?”. Spełnianie oczekiwani klienta i dostarczanie produktów wysokiej jakości jest niejako efektem ubocznym podstawowego działania wynikającego z nadzornej misji i inspirującej wizji. Nie jest to ani proste, ani oczywiste, jakby się na pierwszy rzut oka wydawało. Wymaga to solidnych fundamentów, umiejętności zarządzania, świadomości oraz kompetencji wszystkich udziałowców w ramach całej turkusowej organizacji.

Dla niektórych taka koncepcja może wydawać się czystą utopią, jednak jest wiele przykładów właśnie tak działających organizacji. Osobom zainteresowanym nową doktryną jakości w turkusowym

wydaniu polecam książkę „Doktryna Jakości (wydanie II turkusowe)“ [3].

Model organizacji w kolorze turkusowym wydaje się być odpowiedni do zarządzania startupami, oczywiście pod warunkiem posiadania odpowiedniej załogi na pokładzie zarówno pod kątem zróżnicowania, jak i dopasowania. Warto przy tym dodać, że w celu odniesienia sukcesu w turkusowej organizacji przyda się odrobinę szczęścia, szczypta pomysłowości i garść wytrwałości.

## PODSUMOWANIE

W tym artykule opisalem dość ciekawy punkt widzenia na zespoły zadaniowe i drużyn aktywne, które osiągają lepsze wyniki, kiedy są odpowiednio zróżnicowane i dopasowane, biorąc pod uwagę styl komunikacji i typy zachowania. Przedstawiłem poza tym kolorystyczne i odpowiadające im modele, przelomowe idee oraz charakterystyczne cechy. Wraz z rozwojem ludzkiej świadomości, gdy pojawiają się nowe modele organizacyjne, coraz większego znaczenia nabiera praca zespołowa. Na przykład przy czerwieni liczy się głównie kult jednostki, przy turkusie liczy się przede wszystkim samoorganizujący się zespół. Dlatego też umiejętność budowania zespołów jest tak bardzo istotna do osiągnięcia sukcesu, w szczególności przy zawiązywaniu startupów.

## Literatura

- [1] Buckingham Marcus, Coffman Curt, „Po pierwsze: złam wszelkie zasady. Co najwięksi menedżerowie na świecie robią inaczej”, MT Biznes, Warszawa 2001
- [2] Laloux Frederic, „Pracować inaczej. Nowatorski model organizacji inspirowany kolejnym etapem rozwoju ludzkiej świadomości”, Studio Emka, Warszawa 2015
- [3] Billek Andrzej Jacek, „Doktryna jakości (wydanie II turkusowe) – rzecz o turkusowej samoorganizacji”, Helion Onepress, Warszawa 2016

## GRZEGORZ OLENDER

Pasjonat rozwoju oprogramowania i telekomunikacji z wieloletnim doświadczeniem w obszarze badań i rozwoju. Absolwent Wydziału Elektroniki i Telekomunikacji Politechniki Wrocławskiej. Obecnie pracuje w firmie Nokia przy rozwoju technologii LTE. Triathlonista w wolnych chwilach.