

Magazyn programistów i liderów zespołów IT

**programista**  
8/2017 (63)  
wrzesień/październik

Cena 23,90 zł (w tym VAT 5%)

**Praktyczne ataki  
na AESa i nie tylko****SZYFRY  
BLOKOWE****Na granicy światów  
– technologia bezpieczeństwa  
ARM TrustZone****Automatyczne przetwarzanie  
kodu C++ z Clangiem****Przetwarzanie obrazów  
w biometrii****Office UI Fabric**

II EDYCJA

**Security PWNING  
Conference  
2017**

WARSZAWA | 6-7 LISTOPADA 2017

ISSN 2084-9400

08





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

## Praktyczna kryptografia

Dzięki swojemu otwartemu charakterowi Internet codziennie przyciąga rzesze ludzi poszukujących wiadomości, śmiesznych obrazków albo miejsca na dyskusję z innymi użytkownikami. Relatywnie wysoki stopień anonimowości i opiezałości niektórych użytkowników często jest przyczyną powstawania fałszywych informacji. Naturalne jest, że rozumiejąc te problemy, z dystansem podchodzimy do wiadomości prezentowanych na portalach typu „ŁamiąceWiadomości.pl”<sup>1</sup>. Jednak czy podobną ostrożność trzeba zachowywać, jeżeli poruszamy się na portalu dla profesjonalistów? Gdzie spora część użytkowników podpisuje się swoim prawdziwym imieniem i nazwiskiem?<sup>2</sup>

W tytulowym artykule tego wydania „Programisty”, autor – Jarosław Jedynak – postara się udowodnić, że „wiedza powszechna” nie zawsze występuje w intersekcji z „wiedzą poprawną”... i nie stanowią tu wyjątku wypowiedzi bywałców GitHuba czy StackOverflow. Zapraszamy do zapoznania się z ww. tekstem prezentującym szereg popularnych nieprawidłowości i nieporozumień podczas wykorzystywania szyfrów blokowych w praktyce, które nierzadko rujnują całe bezpieczeństwo proponowanego rozwiązania.

Życzymy miłej lektury!  
Michał Leszczyński

PS. Autorem projektu okładki do tego numeru jest Sebastian Rosik.

1. Mam nadzieję, że taki portal nie istnieje, bo inaczej pojedę z torbami...
2. Definitywnie złamałem prawo nagłówków Betteridge'a.

## BIBLIOTEKI I NARZĘDZIA

<b>3 najlepsze Python IDE dla Data Scientistów</b>	4
Maciej Bartoszuk	

## JĘZYKI PROGRAMOWANIA

<b>Automatyczne przetwarzanie kodu C++ z Clangiem</b>	8
Tomasz Karwala	

## PROGRAMOWANIE SYSTEMOWE

<b>Na granicy światów – technologia bezpieczeństwa ARM TrustZone</b>	18
Jan Dąbrowski	

## PROGRAMOWANIE APLIKACJI WEBOWYCH

<b>Office UI Fabric</b>	24
Dawid Borycki	

<b>Co każdy webmaster o dostępności wiedzieć powinien</b>	30
Bartłomiej Krakowski	

## PROGRAMOWANIE GRAFIKI

<b>Być jak Sherlock Holmes. Przetwarzanie obrazów w biometrii</b>	34
Maciej Szymkowski	

## PROGRAMOWANIE BAZ DANYCH

<b>Data Lake – to robi różnice</b>	44
Dawid Benski, Michał Dura	

## TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

<b>Gitlab CI – od pomysłu do produkcji</b>	48
Michał Pawlik	

<b>Wyjdź poza schemat – testuj dostępność aplikacji</b>	54
Kinga Witko	

## BEZPIECZEŃSTWO

<b>Praktyczna kryptografia: Szyfry blokowe</b>	58
Jarosław Jedynak	

## PLANETA IT

<b>Kurs angielskiego dla programistów. Lekcja 8</b>	72
Łukasz Piwko	

## KLUB DOBREJ KSIĄŻKI

<b>Indie Games. Podręcznik niezależnego twórcy gier</b>	74
Mariusz „maryush” Witkowski	

Zamów prenumeratę magazynu Programista  
przez formularz na stronie:

<http://programistamag.pl/typy-prenumeraty/>

lub zrealizuj ją na podstawie faktury Pro-forma. W spawie faktur Pro-forma prosimy kontaktować się z nami drogą mailową:  
[redakcja@programistamag.pl](mailto:redakcja@programistamag.pl).

Prenumerata realizowana jest także przez RUCH S.A.

Zamówienia można składać bezpośrednio na stronie: [www.prenumerata.ruch.com.pl](http://www.prenumerata.ruch.com.pl)

Pytania prosimy kierować na adres e-mail: [prenumerata@ruch.com.pl](mailto:prenumerata@ruch.com.pl)

lub kontaktując się telefonicznie z numerem:

**801 800 803 lub 22 717 59 59**, godz. 7:00 – 18:00 (koszt połączenia wg taryfy operatora).

Magazyn Programista wydawany jest przez Dom Wydawniczy Anna Adamczyk

**Wydawca/Redaktor naczelny:** Anna Adamczyk ([annaadamczyk@programistamag.pl](mailto:annaadamczyk@programistamag.pl)).

**Redaktor prowadzący:** Michał Leszczyński ([mleszczyński@programistamag.pl](mailto:mleszczyński@programistamag.pl)).

**Korekta:** Tomasz Łopuszański. **Kierownik produkcji:** Havok. **DTP:** Havok.

**Dział reklamy:** [reklama@programistamag.pl](mailto:reklama@programistamag.pl), tel. +48 663 220 102, tel. +48 604 312 716.

**Prenumerata:** [prenumerata@programistamag.pl](http://programistamag.pl/typy-prenumeraty/).

**Współpraca:** Michał Bartylewski, Mariusz Sieraczewicz, Dawid Kaliszewski, Marek Sawerwain, Łukasz Mazur, Łukasz Łopuszański, Jacek Matulewski, Sławomir Sobótka, Dawid Borycki, Gynvael Coldwind, Bartosz Chrabski, Rafał Kocisz, Michał Sajdak, Michał Bentkowski, Mariusz „maryush” Witkowski, Paweł „KrzaQ” Zakrzewski.

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

**Druk:** Drukarnia Edit ul. Dworkowa 2, 05-462 Wiązowna, Nakład: 4500 egz.

Nota prawa

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

O ile nie zaznaczono inaczej, wszelkie prawa do materiałów i znaków towarowych/firmowych zamieszczanych 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.

# 3 najlepsze Python IDE dla Data Scientistów

Jak powszechnie wiadomo, praca Data Scientisty nie ogranicza się jedynie do wyprowadzania skomplikowanych równań na kartce papieru, ale jest to też praktyczna, programistyczna praca przy komputerze. Znajomość języków programowania, np. Pythona, a także bibliotek, takich jak Numpy, Pandas, Matplotlib, to jedno. Jednak wybór IDE, środowiska, w którym Data Scientist będzie pisał swój program lub przeprowadzał analizę eksploracyjną, jest także sprawą niesłychanie ważną.

Zanim przejdziemy do opisu najlepszych IDE dla Data Scientistów pracujących w Pythonie, zastanówmy się, czemu wybór środowiska jest taki ważny oraz czym dobre IDE powinno się charakteryzować. Przede wszystkim IDE powinno ułatwiać nam wykonywanie codziennych czynności, a przez to przyspieszać naszą pracę. Chodzi tu czasem o rzeczy naprawdę banalne, jak intuicyjna nawigacja po plikach w projekcie, możliwość posiadania paru plików otwartych w kilku zakładkach czy wyświetlanie zawartości paru plików naraz, obok siebie. Możliwość dostosowania wyglądu IDE pod swoje potrzeby jest także bardzo ważna: chodzi tu o możliwość dowolnego przeciągania poszczególnych okien środowiska, aby były tam, gdzie wydaje nam się to najwygodniejsze, a także dowolność w ich otwieraniu i zamykaniu.

## ANALIZA SKŁADNI I CO Z TEGO WYNIKA

Kolejną cechą, jaka zdecydowanie odróżnia środowisko z prawdziwego zdarzenia od zwykłego notatnika, jest kolorowanie składni. Osobne kolory otrzymują: słowa kluczowe danego języka, zmienne lokalne, pola klas, nazwy funkcji, metod, treść komentarzy czy stałych tekstowych. Dzięki temu jeden rzut oka w zupełności wystarczy, aby zorientować się, czy dany identyfikator jest argumentem metody, czy polem klasy, poza tym pozwala to naszym umysłom łatwiej oddzielać od siebie poszczególne części kodu.

Rozumienie języka przez IDE sprawia, że może nam ono pomóc na o wiele więcej sposobów, niż tylko wspomniane wcześniej kolorowanie składni. Jest to chociażby podkreślanie błędów składniowych. Dzięki temu, jeśli zdarzy nam się literówka, widzimy ją od razu, bez konieczności napisania całego fragmentu kodu, a następnie jego uruchamiania, by przekonać się, że nasza funkcja nie działa z powodu braku wcięcia, dwukropka czy średnika.

Jeśli chodzi o wykrywanie błędów, to poza poprawianiem składni dobre środowisko powinno także wspomagać proces tzw. debugowania. Mamy tu na myśli uruchomienie kodu, a następnie możliwość prześledzenia, jak wygląda przepływ sterowania w programie, a także stan zmiennych w poszczególnych momentach działania aplikacji. Zazwyczaj taką analizę zaczynamy od postawienia tzw. breakpointa, czyli oznaczenia pewnego wiersza kodu, tak aby zatrzymać wykonywanie programu, gdy przepływ sterowania dotrze właśnie do niego. W tym momencie w nowym okienku powinniśmy być w stanie podejrzeć wartości zmiennych (powinno być to też możliwe po najechaniu na którąś z nich kursem), a także nakazać przejście do kolejnej instrukcji. W przypadku, gdy docieramy do wywołania funkcji, standardem jest to, że możemy

wybrać, czy chcemy się w nią zagłębić, czy też potraktować ją jako pojedynczą instrukcję.

Oczywiście dobre środowisko programistyczne nie tylko pomaga nam w znajdowaniu błędów w istniejącym kodzie, ale też podpowiada, co napisać. Funkcję tę najczęściej nazywamy IntelliSense albo autouzupełnianiem. Oznacza to, że jeśli napiszemy nazwę obiektu, a następnie kropkę, wyświetli się nam okienko, w którym zobaczymy, co możemy po takiej kropce napisać. Najczęściej są to oczywiście pola i metody, które taki obiekt udostępnia. Co więcej, gdy już napiszemy wywołanie funkcji lub metody, dostaniemy informację, jakie argumenty ona przyjmuje. Taki sposób podpowiadania pozwala nam znaczco zaoszczędzić czas, który normalnie byśmy musieli spędzić, czytając dokumentację.

Powiada się, że kod źródłowy znacznie częściej się czyta, niż pisze. Oznacza to m.in., że częstym scenariuszem jest czytanie własnego (lub cudzego) kodu sprzed paru tygodni lub miesięcy, by go zrozumieć i użyć w nowym projekcie lub po prostu zmodyfikować. W tym przypadku nieraz chcemy podejrzeć kod funkcji, która jest wywoływana w analizowanym przez nas fragmencie. Jednak ciało takiej funkcji może znajdować się w zupełnie innym pliku. Środowisko programistyczne powinno udostępniać nam możliwość automatycznego przejścia do definicji, niezależnie od tego, gdzie takowa się znajduje, poprzez ustawienie kurSORA na nazwie i naciśnięciu odpowiedniego skrótu klawiszowego. Co więcej, wcale nie takim rzadkim przypadkiem jest możliwość późniejszego cofnięcia się z powrotem.

## PROFESJONALNE WYTWARZANIE OPROGRAMOWANIA: JAKOŚĆ KODU I JEGO WERSJONOWANIE

Podczas profesjonalnego wytwarzania oprogramowania szczególną uwagę zwraca się na jakość kodu. Kod powinien nadawać się do wielokrotnego wykorzystania, a gdy ktoś będzie go czytał albo chciał zmodyfikować, nie powinien mieć z tym większych problemów. Oczywiście nie zawsze od razu wszystko udaje nam się napisać idealnie, a koncepcja potrafi zmienić się w trakcie. Dlatego bardzo przydatne są wszelkie funkcje, które pozwalają nam na tzw. refactoring, czyli takie operacje na kodzie źródłowym, które nie zmieniają jego działania, ale wpływają na jego jakość. Podstawową czynnością jest zmiana nazwy zmiennej lub funkcji czy metody. Funkcja wykonywała dwie czynności, ale po zmianach wykonuje jedną? Na pewno powinniśmy to odwzorować w jej nazwie. Po wprowadzeniu nowego identyfikatora inteligentne IDE wykryje i zamieni

# Zintegruj własną aplikację z SMSAPI

Skorzystaj z gotowych bibliotek w językach:



Załącz 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:

BONUS:

**PR56 500 SMS-ÓW**



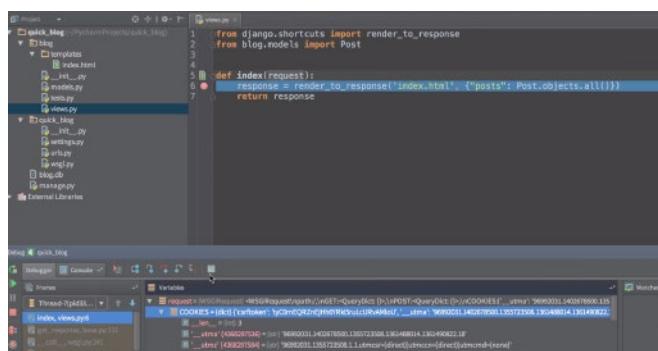
wszelkie wywołania takiej funkcji na nową nazwę w całym naszym projekcie. To samo tyczy się oczywiście zmiennych, tutaj także wszelkie jej wystąpienia zostaną zmienione.

Ostatnią cechą, jaką charakteryzuje powinno się dobre IDE, jest integracja z systemem wersji, np. Gitem. W dzisiejszych czasach coraz rzadziej tworzy się oprogramowanie samemu, na swoim komputerze, bez współpracy z innymi ludźmi. Zazwyczaj kod źródłowy projektu jest trzymany na zewnętrznym serwerze, a wszyscy programiści pobierają jego kopię, dokonują zmian, a następnie przesyłają nową wersję z powrotem do publicznego repozytorium. IDE pomoże nam zobaczyć, jakie pliki zmodyfikowaliśmy, wybrać, które z nich chcemy wysłać na serwer, a także obejrzeć historię zmian danego pliku czy wycofać swoje zmiany i pobrać z powrotem kod z serwera.

Teraz, gdy wiemy, czym kierować się przy wyborze IDE, a także jakie są ich możliwości, pozostaje odpowiedzieć na pytanie, które z nich wybrać, będąc Data Scientistem programującym w Pythonie?

## PYCHARM

PyCharm jest na pewno środowiskiem programistycznym z prawdziwego zdarzenia, używanym przez różnych programistów Pythona, nie tylko tych z obszaru Data Science. Znajdziemy tu wszystkie opisywane wcześniej zalety. Możemy otwierać wiele plików w zakładkach, dowolnie dzielić okno, aby zobaczyć zawartość paru naraz, otwierać i zamykać inne okna. Tak samo bez zarzutu działa tu kolorowanie składni, debugowanie, autouzupełnianie kodu, przejście do definicji, zmiana nazwy funkcji czy integracja z systemem wersji.



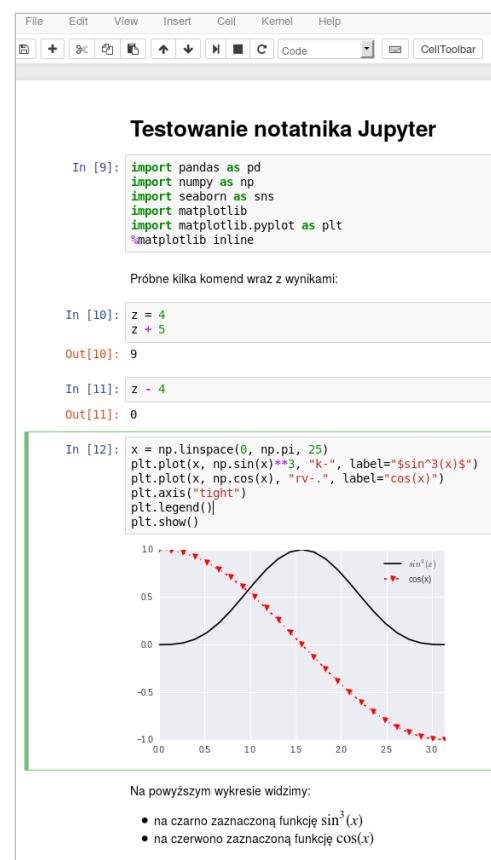
Rysunek 1. Wśród wielu zalet środowisko PyCharm umożliwia łatwą nawigację po plikach w projekcie, kolorowanie składni i debugowanie wadliwego kodu

PyCharm występuje zarówno w wersji darmowej (community), jak i płatnej, z dodatkowymi funkcjami. Na pewno warto najpierw zapoznać się z wersją bezpłatną, zanim zdecydujemy się wydać większą kwotę na płatną subskrypcję. Czy jednak oznacza to, że nie musimy już dalej szukać?

## Jupyter/IPython

Wszystko, co do tej pory napisaliśmy, tyczy się następującego trybu pracy: najpierw piszemy w całości kod, a następnie go uruchamiamy. Dane są przetwarzane w całości i tylko ostateczny wynik jest np. zapisywany do pliku lub bazy danych. O ile jest to typowy schemat postępowania dla programisty, o tyle nie jest on do końca wygodny dla analityka danych. Najczęściej zachodzi potrzeba działania małymi krokami: wczytania danych, podejrzenia ich, dokonania ich czyszczania, pogrupowania, znów podejrzenia, co

otrzymaliśmy, następnie policzenia prostych statystyk, aby przejść do budowania różnych modeli. Tak więc jest to na zasadzie interaktywnej: na jeden nasz niewielki fragment kodu chcemy otrzymać jakiś wynik (może to też być tabela lub rysunek) i dopiero na jego podstawie zdecydować o dalszych krokach. Dopiero gdy przejdziemy przez całą analizę, jesteśmy w stanie napisać pełen kod do naszego zagadnienia, np. w PyCharmie.



Rysunek 2. W interaktywnym notatniku Jupyter możemy przeplatać ładnie sformatowaną tekst, kod Pythona oraz jego wyniki

Dlatego taką popularnością wśród Data Scientistów cieszą się wszelkie edytory interaktywne. Jednym z najczęściej wymienianych jest Jupyter, który pozwala na tworzenie tego typu notatników w różnych językach programowania. W przypadku Pythona bazuje on na silniku IPython. Ma on spore ograniczenia w porównaniu do typowego IDE, takiego jak PyCharm, np. cały kod widzimy od góry do dołu, bez podziału na pliki (można oczywiście stworzyć parę notatników, ale to nie to samo, bo nie można np. zaimportować funkcji z jednego notatnika do drugiego). Tak samo nie pozwoli nam na zaawansowane debugowanie kodu, refactoring czy przejście do definicji. Jednak mamy tutaj dostęp do autouzupełniania kodu, podpowiedzi dotyczących argumentów funkcji czy szybkie zjazdzenie do plików dokumentacji, co czasami w zupełności wystarczy. Co więcej, mamy tu cechy, których normalnie nie spotkamy: możemy przeplatać kod na przemian z markdowniem, językiem pozwalającym na stworzenie schludnego tekstu, z pogrubieniem, kursywą czy wypunktowaniem. Zatem Jupyter, poza możliwością szybkiego tworzenia prototypów czy dokonania eksploracyjnej analizy danych, pozwala także w prosty sposób stworzyć raport, w którym mamy jednocześnie tekst, jak i wyniki naszych analiz, np. tabelę ze średnimi czy histogram rozkładu wieku klientów.

## RODEO

Jeśli chcielibyśmy otrzymać kompromis pomiędzy pełnoprawnym IDE a pracą interaktywną, pewnym rozwiązaniem jest Rodeo. Jest ono wzorowane na RStudio, środowisku powszechnie używanym i uznawanym za standard dla języka R. Ekran Rodeo, podobnie jak RStudio, podzielony jest na 4 części: w jednej znajduje się nasz kod, w drugiej konsola Pythona, w której możemy zobaczyć wyniki poszczególnych wierszy naszego kodu, w trzeciej jest miejsce na rysunki, podczas gdy ostatnia zawiera nazwy zmiennych, które do tej pory stworzyliśmy wraz z wartościami.

Cała siła Rodeo polega na tym, że piszemy kod dokładnie w ten sam sposób, jak byśmy to robili w edytorze tekstowym, jednak gdy zaznaczmy pewien fragment kodu lub ustawiemy kursor na danym wierszu, po wcisnięciu odpowiedniego skrótu klawiszowego otrzymujemy na konsoli wynik jedynie tego zaznaczonego fragmentu kodu. Tak samo po wykonaniu kodu tworzącego rysunek

możemy go od razu zobaczyć w odpowiedniej części IDE. O ile taki kompromis wydaje się idealny, to niestety trzeba pamiętać, że w Rodeo debugowanie, zarządzanie plikami, przechodzenie do definicji czy refactoring nie jest tak wygodny, jak ma to miejsce w PyCharmie. Tak samo wynik naszej pracy nigdy nie będzie od razu gotowym raportem, gotowym do zaprezentowania szerszemu gronu odbiorców, jak w przypadku Jupytera. Jednak do codziennej pracy, która polega na interaktywnej analizie zbiorów danych i budowaniu prototypów modeli, może okazać się nieco wygodniejszy od notatników Jupytera.

Jak widzimy, dobór odpowiedniego środowiska dla Data Scientista jest trudniejszy, niż dla zwykłego programisty. Poza standaryзовymi potrzebami, jak autouzupełnianie kodu czy wskazywanie błędów syntaktycznych, bardzo ważna jest możliwość pracy interaktywnej, której zazwyczaj nie udostępniają typowe IDE. Dlatego należy tak zorganizować swoją pracę, aby być w stanie wykorzystać mocne strony każdego z dostępnych na rynku narzędzi.



Rysunek 3. Rodeo dzieli nasz ekran na 4 części, które umożliwiają jednoczesną edycję kodu, oglądanie wyników poszczególnych jego fragmentów, a także podglądarki wizualizacji w tym samym oknie



### MACIEJ BARTOSZUK

Prowadzący bootcamp Data Science na [Kodołamacz.pl](http://Kodołamacz.pl). Ukończył z wyróżnieniem informatykę na wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej, gdzie aktualnie pracuje w zakładzie Sztucznej Inteligencji i Metod Obliczeniowych. Przygotowuje doktorat, system antyplagiatowy dla języka R, który obejmuje zarówno algorytmy przetwarzania kodów źródłowych programów, jak i data science. Współautor książki „Przetwarzanie i analiza danych w języku Python” wydanej przez PWN.

# Automatyczne przetwarzanie kodu C++ z Clangiem

Kompilator Clang udostępnia swój front-end w formie bibliotek. Możemy tworzyć własne narzędzia do refaktoryzacji. Wymagany nakład pracy wcale nie jest duży, zwłaszcza jeśli może nas uchronić przed ręcznym poprawianiem ogromnej ilości kodu. Przyjrzymy się też pułapkom, jakie na nas czekają, i problemom przy współpracy z GCC.

## WSTĘP

Wszyscy znamy sytuację, kiedy trzeba poprawić kilkudziesiąt podobnych miejsc w kodzie. Na przykład kiedy mocno zmieniliśmy listę argumentów funkcji. Zwykle radzimy sobie, robiąc „Znajdź i zamień” w edytorze tekstu. Ta metoda nie jest doskonała, zawsze znajdzie się kilka miejsc, które trzeba poprawić ręcznie, bo wywołanie naszej przykładowej funkcji jest rozpisane na kilka wierszy, bo używa makr, sklejanych łańcuchów znaków... Ale na ogół to wystarcza.

W ogromnych projektach nie możemy sobie pozwolić na ręczne poprawianie, musimy zautomatyzować ten proces tak bardzo, jak tylko się da. Potrzebujemy narzędzia, które „rozumie” poprawiany kod. Kompilator Clang daje dostęp do swojego *front-endu* w formie bibliotek z eleganckim API. Możemy wczytać każdy poprawny kod C++ i operować na abstrakcyjnym drzewie składniowym. W artykule zostaną opisane także praktyczne problemy związane z użyciem Clanga na projekcie, który jest budowany przy użyciu GCC.

## CO ZAMIERZAMY

Napiszemy narzędzie, które zamienia wywołania funkcji Trace i Error z klasy Logger na makra TRACE i ERROR, to znaczy zamienia mLogger->Error("Foo") na ERROR((mLogger), ("Foo")). Sama w sobie taka funkcjonalność nie jest niczym ciekawym, służy tylko za prosty przykład. Rzecz w tym, że nasze narzędzie poradzi sobie z różnymi formami wywołań, skomplikowanymi wyrażeniami podanymi jako argumenty i nie zmylią go wywołania funkcji innych klas, które również nazywają się Trace i Error. Obsłużymy też różne przypadki użycia makr. Po drodze poznamy różne zawiłości Clanga, czasami takie, które nie są tutaj wykorzystane, ale otworzą czytelnikowi drogę do własnych eksperymentów.

Spójrzmy teraz na przykładowy plik wejściowy i odpowiadający mu plik wynikowy. Oto, co zamierzamy.

**Listing 1. Przykładowy plik wejściowy**

```
#define PREFIX "Error: "
#define FUNC_ENTER Trace("Entering " FUNC)
#define TR Trace
#define MAIN_LOGGER 0
#define SELF this

class Logger {
public:
    void Trace(const char *fmt, ...);
    void Error(const char *fmt, ...);

#define FUNC "Test"
void Test()
{
    FUNC_ENTER;
}
```

```
this->TR("Foo");
SELF->Trace("Bar");
Trace("Hello world");
this->Error("Hello %s",
            "world");
smLoggers[0].Trace("Foo");
smLoggers[MAIN_LOGGER].Trace("Foo");
Error(PREFIX
      /* really */ "big trouble!");
}

#undef FUNC

static Logger smLoggers[10];
};

class NotLogger {
public:
    void Trace(const char *fmt, ...);
    void Error(const char *fmt, ...);

void Test()
{
    this->Error("Hello %s",
                "world");
    (Logger::smLoggers + 1)->Trace("Foo");
}
};
```

**Listing 2. Wynik działania narzędzia**

```
#define PREFIX "Error: "
#define FUNC_ENTER Trace("Entering " FUNC)
#define TR Trace
#define MAIN_LOGGER 0
#define SELF this

class Logger {
public:
    void Trace(const char *fmt, ...);
    void Error(const char *fmt, ...);

#define FUNC "Test"
void Test()
{
    FUNC_ENTER;
}

this->TR("Foo");
SELF->Trace("Bar");
TRACE(this, ("Hello world"));
ERROR(this, ("Hello %s"), ("world"));
TRACE(&(smLoggers[0]), ("Foo"));
smLoggers[MAIN_LOGGER].Trace("Foo");
ERROR(this, (PREFIX
      /* really */ "big trouble!"));

}

#undef FUNC

static Logger smLoggers[10];
};

class NotLogger {
public:
    void Trace(const char *fmt, ...);
    void Error(const char *fmt, ...);

void Test()
{
    this->Error("Hello %s",
                "world");
    TRACE((Logger::smLoggers + 1), ("Foo"));
}
};
```



ERICSSON

PROJEKTUJEMY.  
ROZWIJAMY.  
DOSTARCZAMY.

Aktualnie szukamy:  
**Software Tester/Developer**  
**Software Developer (C++)**

5G  
THE JOURNEY  
HAS STARTED

Aplikuj:

[Ericsson.com/careerspoland](http://Ericsson.com/careerspoland)



Przyglądając się uważnie, można zauważyc, że nie chcemy zamieniać wszystkich wywołań Trace i Error. Przede wszystkim nie chcemy zamieniać NotLogger::Error, interesują nas tylko wywołania funkcji klasy Logger. Następnie, w niektórych przypadkach, nie chcemy zamieniać wywołań napisanych z użyciem makr. Jakich sytuacji chcielibyśmy uniknąć?

Możemy wyobrazić sobie, na przykład, że makro FUNC\_ENTER jest różnie zdefiniowane w zależności od tego, czy budujemy wersję debugową, czy produkcyjną, i w wersji produkcyjnej nie robi nic. Nie chcielibyśmy zamieniać w kodzie programu FUNC\_ENTER na TRACE(this, "Entering function " FUNC).

Podobnie, gdyby definicje TR i SELF zależały od zewnętrznych parametrów, this->TR("Foo") oraz SELF->Trace("Bar") mogłyby nie zawsze oznaczać wywołania Logger::Trace.

Wolimy raczej zawiadomić użytkownika, że takie rozwinięcie makra wystąpiło, i pozostawić zmianę ciała makra użytkownikowi.

Z drugiej strony argumenty funkcji zawierające makra nie stanowią dla nas problemu (zakładamy, że funkcje Error i Trace nie są przeciążone i nie grozi nam wybór złego wariantu funkcji).

Na koniec zauważmy, że wywołanie, w którym występuje MAIN\_LOGGER, też nie sprawia problemów, ale mimo tego, zachowawczo, pomijamy je, bo inaczej musielibyśmy, zamiast prostej reguły „w obiekcie nie ma makr”, polegać na jakiejś formie analizy semantycznej. To jest wykonalne, ale ogromnie skomplikowałoby artykuł.

## BUDOWANIE

Będziemy potrzebowali źródeł Clanga (i projektu LLVM, którego Clang jest częścią). Kod z tego artykułu został skompilowany razem z wersją 3.6.1. Nowsze wersje Clanga też powinny działać. Nie będziemy opisywać, jak zdobyć i zbudować Clanga i LLVM, to zostało dokładnie wyjaśnione na oficjalnej stronie projektu: <http://clang.llvm.org>. Nasze przykładowe narzędzie jest na tyle proste, że powinno działać na wszystkich platformach wspieranych przez Clanga.

Spośród różnych sposobów budowania narzędzi opartych na Clangu najmniej kłopotliwy polega na umieszczeniu naszego narzędzia w drzewie źródłowym Clanga, tak że zostanie ono zbudowane razem z całym projektem LLVM.

Potrzebujemy stworzyć podkatalog, na przykład *article-example*, w katalogu *tools/clang/tools* w drzewie źródłowym LLVM. W *tools* znajdują się pliki *CMakeLists.txt* i *Makefile*, do których należy dodać nasze narzędzie. Widząc zawartość tych plików, nietrudno domyślić się, co i gdzie należy dopisać.

W podkatalogu *article-example* należy utworzyć kolejne *CMakeLists.txt* i *Makefile*. Wyglądają one tak:

**Listing 3. CMakeLists.txt**

```
set(LLVM_LINK_COMPONENTS
    Option
    Support
)
add_clang_executable(article-example
    article-example.cc
)
target_link_libraries(article-example
    clangASTMatchers
    clangAST
    clangBasic
    clangDriver
    clangFrontend
    clangRewriteFrontend
    clangStaticAnalyzerFrontend
    clangTooling
```

```
clangToolingCore
)
install(TARGETS article-example
    RUNTIME DESTINATION bin)
```

**Listing 4. Makefile**

```
# based on tools/clang-check/Makefile
CLANG_LEVEL := ../../.
TOOLNAME = article-example
# No plugins, optimize startup time.
TOOL_NO_EXPORTS = 1
include $(CLANG_LEVEL)/../../Makefile.config
LINK_COMPONENTS := $(TARGETS_TO_BUILD) asmparser \
    bitreader support mc option
USEDLIBS = clangASTMatchers.a clangFrontend.a \
    clangSerialization.a clangDriver.a \
    clangTooling.a clangToolingCore.a \
    clangParse.a clangSema.a \
    clangStaticAnalyzerFrontend.a \
    clangStaticAnalyzerCheckers.a \
    clangStaticAnalyzerCore.a clangAnalysis.a \
    clangRewriteFrontend.a clangRewrite.a \
    clangEdit.a clangAST.a clangLex.a clangBasic.a
include $(CLANG_LEVEL)/Makefile
```

Kompilator Clang jest podzielony na wiele komponentów, stąd dłuża lista bibliotek. Kompletowanie tej listy jest dość żmudne. Najlepiej skopiować ją z innego narzędzia z katalogu *tools*, które wydaje nam się podobne do naszego, a następnie uzupełnić brakującymi bibliotekami. Niestety, czasami znalezienie biblioteki zawierającej funkcje i klasy z pliku nagłówkowego, który włączamy, wymaga długich dociekań. Przynajmniej nie musimy składać tych licznych komponentów w jedną całość w naszym kodzie źródłowym, bo to dla nas zrobi LibTooling.

## KOD

Jak widać w *CMakeLists.txt*, pozostało nam tylko utworzyć plik *article-example.cc*. Teraz zaprezentujemy jego zawartość. Po pierwsze, będziemy potrzebować następujących komponentów:

**Listing 5. Pliki nagłówkowe**

```
#include "clang/Frontend/FrontendActions.h"
#include "clang/Tooling/CommonOptionsParser.h"
#include "clang/Tooling/Tooling.h"
#include "clang/Tooling/Refactoring.h"
#include "llvm/Support/CommandLine.h"
#include "llvm/Support/raw_ostream.h"

#include "clang/Lex/Lexer.h"

#include "clang/ASTMatchers/ASTMatchers.h"
#include "clang/ASTMatchers/ASTMatchFinder.h"

#include <iostream>
#include <sstream>
#include <list>
#include <string>

using namespace clang;
using namespace clang::ast_matchers;
using namespace clang::tooling;

using namespace llvm;
using std::string;
```

LibTooling (Tooling.h) łączy wszystkie komponenty w działający front-end kompilatora. CommandLine i CommonOptionsParser są potrzebne nie tylko po to, aby zdefiniować swoje linii

polecień, ale przede wszystkim, żeby obsłużyć standardowe opcje kompilatora, bo one mają wpływ na to, jak interpretujemy kod źródłowy: ścieżki do plików nagłówkowych (-I), predefiniowane makra (-D), ... A może używamy *trigraphów*? LibRefactoring to prosta biblioteka, która pozwala nam podmieniać fragmenty kodu źródłowego. Lexer.h potrzebny jest nam do wyciągania fragmentów kodu i śledzenia rozwinięć makr. ASTMatchers to wygodna biblioteka służąca do wyszukiwania w kodzie, aścielj – w abstrakcyjnym drzewie składniowym – miejsc, które pasują do wzorca. W końcu raw\_ostream to preferowany w LLVM sposób pisania na std::err.

Korzystając z CommandLine.h, zdefiniujemy tylko jedną opcję: -l, żeby wylistować zmiany, których ma dokonać nasze narzędzie w pliku wejściowym, zamiast naprawdę je wprowadzić.

#### **Listing 6. Opcje linii poleceń**

```
static llvm::cl::OptionCategory
categoryExampleTool("Example Tool options");
static cl::extraHelp
commonHelp(CommonOptionsParser::HelpMessage);

static cl::opt<bool> listOnly("l", cl::desc(
    "Only list changes without modifying files. "
    "Errors are redirected to standard output."),
    cl::cat(categoryExampleTool));
```

CommandLine.h udostępnia *domain specific language* do definiowania opcji linii poleceń. Dzięki temu wszystko, czego potrzebujemy do obsługiwała przełącznika -l, to te statyczne obiekty. Ich konstruktory automagicznie skonfiguruje CommonOptionsParser. Zmienną listOnly będzie można traktować jak zmienną bool, która wskazuje, czy przełącznik -l pojawił się w linii poleceń.

Kiedy front-end sparsuje plik źródłowy, moglibyśmy, za jego pośrednictwem, samodzielnie przeglądać węzły drzewa składniowego w poszukiwaniu wywołań funkcji, które nas interesują. Jednak prościej będzie skorzystać z biblioteki ASTMatchers. Zdefiniujemy wzorzec, a biblioteka sama znajdzie węzły drzewa, które do niego pasują. Następnie dla każdego dopasowania wywoła callback, w którym określmy zmiany, jakie chcemy wprowadzić w tym fragmencie kodu.

#### **Listing 7. StatementMatcher**

```
StatementMatcher loggerInvocationMatcher =
memberCallExpr(
    thisPointerType(
        recordDecl(hasName("Logger"))),
    callee(
        methodDecl(
            anyOf(
                hasName("Trace"),
                hasName("Error"))
            )
        ).bind("function-decl")
    )
).bind("match");
```

Wzorzec memberCallExpr dopasowuje się do wywołań funkcji składowych. Co dla nas wygodne, pasuje do wszystkich form tego wywołania, a więc złapie, między innymi:

```
» mLogger.Error("Foo")
» mLogger->Error("Foo")
» this->Error("Foo")
» Error("Foo") (niejawne this)
» static_cast<Logger*>(Bar())->Error("Foo")
```

Jako argumenty do memberCallExpr podajemy dodatkowe wymagania: thisPointerType, żeby funkcja była wywołana na instancji klasy lub struktury o nazwie Logger; oraz callee, żeby

dopasować się tylko do funkcji o nazwie Trace lub Error. Funkcja bind() nadaje nazwę, pod którą będziemy mogli w callback odwołać się do dopasowanego fragmentu drzewa. Jak widać, znalezione wywołanie funkcji składowej będzie dostępne pod nazwą "match", a pod nazwą "function-decl" będzie dostępna deklaracja funkcji składowej, która jest wywoływana (a więc fragment kodu, który znajduje się w zupełnie innym miejscu niż wywołanie).

Potrzebujemy teraz zdefiniować kilka funkcji pomocniczych. Paradoksalnie, choć nie robią nic ciekawego, są najtrudniejszym fragmentem naszego kodu. Jest tak, ponieważ pracują z klasami SourceLocation i SourceManager, które służą do wskazywania lokalizacji w kodzie źródłowym i śledzenia rozwinięć makr. To właśnie dzięki nim możemy rozpoznać, czy wywołania Trace i Error musimy pominąć ze względu na makra. Mechanizmy SourceLocation są w dokumentacji Clanga opisane powierzchownie, przez co wydają się proste, a w rzeczywistości są skomplikowane, trudno się wnioskować o ich własnościach i używa się ich inaczej niż podpowiada intuicja po przeczytaniu tego powierzchownego opisu. Dlatego zanim zdefiniujemy funkcje pomocnicze, potrzebujemy obszernych wyjaśnień.

Dla lepszego zrozumienia, czym są SourceLocation i SourceManager, przypomnijmy sobie etapy przetwarzania kodu C++. Rozpoczyna się od preprocessingu. Preprocesor dołącza kod za sprawą dyrektywy #include, uwzględnia lub pomija kod ujęty w dyrektywy #ifdef, zastępuje nazwy makr ich rozwinięciami. Makra bywają predefiniowane (\_\_LINE\_\_), zdefiniowane w linii poleceń (-D\_DEBUG), zdefiniowane w kodzie (#define). Co ciekawe, preprocesor w minimalny sposób parsuje kod: dzieli go na leksemy. A więc, na przykład, stała napisowa w cudzysłowach jest dla preprocesora jednym niepodzielnym obiektem. Strumień leksemów, który wypływa z preprocesora, staje się wejściem dla front-endu kompilatora. Kompilator również zaczyna od przyjrzenia się leksemom. Podziału dokonał już preprocesor, kompilator dokonuje pewnych uściśleń. Na przykład to, co dla preprocesora jest stałą liczbową, dla kompilatora może być stałą całkowitoliczbową, zmiennoprzecinkową – a nawet niepoprawnym ciągiem znaków. Dopiero taki strumień leksemów trafia na wejście parsera.

Chociaż preprocesor i kompilator mogłyby być osobnymi programami połączonymi potokiem, w praktyce tak się nie robi. Zównież ze względu na wydajność, jak i ze względu na jakość komunikatów diagnostycznych. I tak właśnie jest w Clangu. Klasa Lexer i powiązane klasy, choć służą przede wszystkim dostarczaniu leksemów na wejście parsera, jednocześnie wykonują całą pracę preprocesora. Dzięki temu, kiedy front-end natrafi na błąd, jest w stanie wyprodukować pomocny komunikat błędu. Taki jak ten:

#### **Listing 8. Przykładowy komunikat błędu Clanga**

```
test.cc:20:12: error: expected expression
    return MIN3(-x, -y, -);
               ^
test.cc:11:30: note: expanded from macro 'MIN3'
#define MIN3(a, b, c) MIN(a, MIN(b, c))
                         ^
test.cc:12:28: note: expanded from macro 'MIN'
#define MIN(a, b) ((a) < (b) ? (a) : (b))
```

W Clangu do wskazywania lokalizacji w kodzie służy typ SourceLocation. Jest on tylko 32-bitowym identyfikatorem, który pozwala wyciągnąć informacje o lokalizacji z obiektów klasy SourceManager, gdzie te informacje są naprawdę przechowywane. SourceLocation nie znaczy prawie nic bez odpowiadającego mu SourceManager, ale razem udostępniają szczegółowe

informacje: gdy SourceLocation wskazuje na bajt, wiemy, czy znajduje się w pliku wejściowym, czy w pliku włączonym przez #include, czy w anonimowym buforze z rozwinięciem makra.

Jeśli lokalizacja wskazuje plik, znamy numer linii i kolumny (i to z uwzględnieniem dyrektywy #line). W przypadku pliku nagłówkowego włączonego przez plik nagłówkowy, włączonego przez jeszcze inny plik nagłówkowy... możemy znaleźć lokalizacje kolejnych dyrektyw #include, aż dojdziemy do miejsca w pliku wejściowym, które spowodowało ich wczytanie.

W przypadku makr sprawa jest bardziej złożona. Najpierw musimy poznać kilka pojęć. W pierwszej chwili mogą się wydać niezrozumiałe, jednak wszystko się wkrótce rozjaśni, gdy odniesiemy je do naszego przykładowego komunikatu błędu. Tak więc, w przypadku makr, wiemy o każdym bajcie:

- » Czy jest częścią rozwinięcia ciała makra, czy argumentu makra,
- » Gdzie jest jego *spelling location*, czyli skąd został wczytany,
- » Gdzie jest jego *expansion location*, czyli gdzie został wstawiony.

Przeanalizujmy teraz przykład. Ktoś najwyraźniej zgubił z. Na samym dole strzałka wskazuje na nawias za literą b. Po rozwinięciu makra, (b) zostaje zastąpione przez (-). Miałoby być (-z), ale ktoś się pomylił i znajduje się tam (-), dlatego parser, po wczytaniu minusa, odrzuca nawias i zgłasza błąd, bo oczekiwali całego wyrażenia. Ten nawias znajduje się gdzieś w anonimowym buforze, w którym dokonuje się rozwinięcie makra. Tam wskazuje jego SourceLocation. Programistę jednak interesuje bardziej miejsce w kodzie źródłowym. W przypadku nawiasu jest on częścią rozwinięcia ciała makra. To znaczy został przepisany z define'a. Jego *spelling location* wskazuje na miejsce w pliku źródłowym, skąd został przepisany. To jest właśnie miejsce, które wskazuje strzałka na samym dole komunikatu błędu. No dobrze, ale gdzie użyliśmy makra MIN, które nam pokazano? Tego dowiemy się, podążając do *expansion location* problematycznego nawiasu. W pewnym sensie – ale tylko w pewnym sensie – tę lokalizację wskazuje środkowa strzałka. Tak naprawdę zostajemy skierowani do kolejnego tajemniczego bufora, w którym dokonano rozwinięcia makra, tym razem MIN3, a dopiero stamtąd *spelling location* odsyła nas do define'a widocznego nad środkową strzałką. Jak widać, *expansion location* wskazuje nam miejsce, gdzie makro MIN zostaje zastąpione swoim ciałem, którego częścią jest śledzony przez nas nawias. W kolejnym kroku *expansion location* zaprowadzi nas już nie do bufora, ale prosto do pliku źródłowego, tam gdzie wskazuje górną strzałkę.

Prześledźmy teraz drogę innego leksemu. Przypomnijmy: do parsera trafia w pewnym momencie ciąg leksemów (-). Minus jest ostatnim zaakceptowanym leksemem, po nim przychodzi odrzucony nawias, który badaliśmy. Jeśli chcemy prześledzić drogę minusa, musimy postępować odwrotnie. Dlaczego? W jakim sensie odwrotnie? Minus nie jest częścią definicji makra MIN, jest tam podstawiony za argument b. Czyli, używając clangowych terminów, jest rozwinięciem argumentu, a nie ciała. W takiej sytuacji *spelling location* i *expansion location* działają odwrotnie. W poprzednim przypadku *expansion location* odsyłał nas wyżej, do definicji MIN3, teraz *expansion location* to jest właśnie to b, które ma zostać zastąpione minusem. A dokładniej, miejsce w jakimś buforze, które dopiero odeśle nas do tekstu define'a. Clang pokazałby to miejsce, umieszczając dolną strzałkę pod b. Natomiast *spelling location*, czyli miejsce, skąd wczytano minus, odeśle nas do kolejnego niewidocznego bufora, w którym następuje rozwinięcie MIN(-y, -), i będzie wskazywać na ten minus. Clang pokazałby to miejsce, umieszczając środkową strzałkę pod c.

Tak działa SourceLocation. Jest to najtrudniejszy do zrozumienia i najgorzej udokumentowany aspekt działania Clanga, którego tutaj używamy, a jednocześnie bardzo ważny, jeśli chcemy automatycznie przetwarzać fragmenty kodu, które używają makr.

Z SourceLocation i Lexer wiąże się jeszcze jedna przykry niespodzianka. Widząc ten elegancki wydruk z rozwinięciami makr, czytając o tych wszystkich odnośnikach do *spelling location* i *expansion location*, można pomyśleć, że Clang daje możliwość swobodnego przeglądania całej historii wczytanych leksemów, śledzenia wszystkich etapów rozwijania makr, zaglądania do wspomnianych anonimowych buforów. Niestety, Lexer bardzo stara się oszczędzać zasoby. Kiedy tylko coś przestaje być potrzebne do komplikacji, znika, a SourceManager przechowuje tylko niezbędnym minimum informacji potrzebnej do wyświetlenia komunikatu błędu, jeśli zajdzie taka potrzeba. Tak więc, na przykład, jeśli znamy lokalizację, w której zaczyna się wyrażenie, i lokalizację, w której się kończy, to niekoniecznie jesteśmy w stanie znaleźć wszystkie lokalizacje pomiędzy końcami. Inny przykład: gdybyśmy zajrzeli do wnętrza funkcji Lexer::getImmediateMacroName, która mając lokalizację rozwinięcia, znajduje nazwę makra, zobaczylibyśmy zawiąz wędrówkę po wewnętrznych strukturach danych i ponowne uruchomienie Lexera, żeby odzyskać dane, które już zniknęły. Wreszcie zwróciły uwagę, że komunikat błędu pokazuje różne miejsca w plikach, ale nie pokazuje finalnego rozwinięcia makr, choć byłoby przydatne.

Teraz już możemy objaśnić, co robią i jak działają funkcje pomocnicze. Funkcja printCoords, oczywiście, mając lokalizację, wypisuje numer linii, kolumny i nazwę pliku. W Clangu nazywa się to *presumed location*. Jeżeli podamy lokalizację w rozwinięciu makra, SourceManager najpierw przejdzie w górę stosu rozwinięć, aż dotrze do pliku źródłowego. W zależności od tego, czy przedostatnia lokalizacja była rozwinięciem ciała, czy argumentu, wyłajemy albo na identyfikatorze makra, albo na argumentie podanym do makra. Tę wędrówkę realizuje funkcja SourceManager::getFileLoc, której użyjemy w innych funkcjach pomocniczych. Zwróciły uwagę, że do wszystkich funkcji pomocniczych przekazywany jest SourceManager, bo bez niego SourceLocation niewiele znaczy.

Kierując się wytycznymi projektu LLVM, zamiast z std::ostream, korzystamy z raw\_ostream, jego zamiennika w LLVM. Strumień odpowiadający standardowemu wyjściu zwraca funkcja outs(), a strumień odpowiadający standardowemu wyjściu błędu zwraca funkcja errs().

Dalsze funkcje działają na obiektach Stmt. Stmt (skróć od statement) jest klasą bazową dla elementów abstrakcyjnego drzewa składniowego. Jest to nieco mylna nazwa, bo Expr (expression) jest jej podklassą, a przecież w gramatyce C++ expression jest raczej składnikiem statement, a nie jego podtypem.

Jak pamiętamy, chcemy, żeby nasze narzędzie pomijało te wywołania, w których obiekt Logger zależy od makra, i te, w których nazwa funkcji pochodzi z makra (pod te przypadki podпадa też FUNC\_ENTER). Sprawdzimy te warunki, uruchamiając naszą funkcję subexprUsesMacros na podwyrażeniu, które wskazuje obiekt, i na podwyrażeniu, które wskazuje nazwę funkcji. Jak widać, klasa Stmt udostępnia nam informacje o lokalizacji, gdzie reprezentowany przez nią fragment kodu się zaczyna i gdzie kończy. Powinniśmy przeiterować po wszystkich lokalizacjach pomiędzy końcami i sprawdzić, czy nie leżą w rozwinięciach makr. Niestety, jak już wiemy, tak się nie da zrobić. Najlepsze przybliżenie, jakie udało się wymyślić autorowi, to przeiterowanie po wszystkich ele-

mentach poddrzewa, w nadziei, że każdy leksem jest jakimś małym podwyrażeniem i zostanie uwzględniony. Przykład wyrażenia, które potrafi oszukać nasz test, to sklejany literal napisowy "foo" INFIX "bar". Jest on pojedynczym elementem drzewa i ani jego początek, ani koniec nie są rozwinięciami makra.

Jak łatwo zgadnąć, `SourceLocation::isMacroID` mówi nam, czy lokalizacja znajduje się w rozwinięciu makra.

Kolejna funkcja znajduje przedział, jaki zajmuje w pliku źródłowym dany element drzewa składniowego. Ponieważ chodzi nam o miejsce w pliku, używamy wspomnianej już funkcji `SourceManager::getFileLoc`. Oprócz tego opakowujemy wynik w obiekt `CharSourceRange`, którego wymaga `Lexer`. Stała `true`, którą przekazujemy do konstruktora `CharSourceRange`, oznacza, że jest to przedział leksemów, to znaczy koniec przedziału nie wskazuje na ostatni znak, tylko na początek ostatniego leksemu.

W końcu mamy funkcję `sourceCode`, która zwraca nam kod źródłowy wyrażenia jako `std::string`. Tutaj pojawia się nowy argument: `LangOptions`. Są tam opcje, których `Lexer` wymaga do poprawnego podzielenia wejścia na leksemły. Skądś musi wiedzieć, że to C++, a nie Objective-C! Odpowiednie `LangOptions` otrzymamy w `callbacku`, musimy je tylko przekazać do funkcji.

#### Listing 9. Funkcje pomocnicze

```
static raw_ostream& printCoords(
    raw_ostream &out,
    SourceLocation srcLoc,
    const SourceManager &srcMgr)
{
    PresumedLoc loc =
        srcMgr.getPresumedLoc(srcLoc);

    out << loc.getFilename() << " "
        << loc.getLine() << ":" 
        << loc getColumn();

    return out;
}

static void printWarning(
    SourceLocation srcLoc,
    const SourceManager &srcMgr,
    const string &msg)
{
    printCoords(errs(), srcLoc, srcMgr)
        << " warning: " << msg << "\n\n";
}

static bool subexprUsesMacros(
    const Stmt *stmt,
    const SourceManager &mgr)
{
    if (stmt->getLocStart().isMacroID())
        return true;
    if (stmt->getLocEnd().isMacroID())
        return true;
    for (const Stmt *subexpr: stmt->children()) {
        if (subexprUsesMacros(subexpr, mgr))
            return true;
    }
    return false;
}

static CharSourceRange sourceRange(
    const Stmt *stmt,
    const SourceManager &mgr)
{
    SourceLocation start = stmt->getLocStart();
    SourceLocation end = stmt->getLocEnd();
    start = mgr.getFileLoc(start);
    end = mgr.getFileLoc(end);
    return CharSourceRange(
        SourceRange(start, end),
        true);
}
```

```
static string sourceCode(
    const Stmt *stmt,
    const SourceManager &mgr,
    const LangOptions &langOpts)
{
    return Lexer::getSourceText(
        sourceRange(stmt, mgr),
        mgr,
        langOpts
    ).str();
}
```

Mamy już `matcher`, teraz możemy napisać `callback`, który będzie wywoływany dla każdego dopasowania znalezionego przez `matcher`. Zadaniem `callbacku` będzie wskazanie miejsc w kodzie, w których należy zamienić wywołania funkcji `Trace` i `Error`, oraz napisanie wywołań makr, które mają zastąpić te wywołania funkcji. Dokonuje się to poprzez rejestrowanie odpowiednich informacji w obiekcie `Replacements`. Później `RefactoringTool` zastosuje te zmiany do pliku wejściowego (o ile nie działamy w trybie `listOnly`).

To właśnie w `callbacku` nasz program przyjrzy sięwęzłom abstrakcyjnego drzewa składniowego. Wyodrębnimy poszczególne elementy wywołania: obiekt, funkcję, kolejne argumenty. Będziemy badać, czy wyrażenia zostały napisane z użyciem makr i czy w związku z tym je pominąć.

#### Listing 10. Callback

```
class ExampleCallback :
public MatchFinder::MatchCallback
{
public :
    ExampleCallback(Replacements &replacements) :
        replacements(replacements)
    {}

    virtual void run(
        const MatchFinder::MatchResult &Result);

private:
    Replacements &replacements;
    std::list<string> replacementStrings;
};

void ExampleCallback::run(
    const MatchFinder::MatchResult &Result)
{
    const SourceManager &srcMgr =
        *Result.SourceManager;
    const LangOptions &langOpts =
        Result.Context->getLangOpts();

    const CXXMemberCallExpr *expr =
        Result
        .Nodes
        .getNodeAs<CXXMemberCallExpr>("match");

    if (subexprUsesMacros(
        expr->getCallee(),
        srcMgr))
    {
        printWarning(
            expr->getExprLoc(),
            srcMgr,
            "Skipping macro expansion: " +
            sourceCode(expr, srcMgr, langOpts)
        );
        return;
    }

    const Expr *loggerExpr =
        expr->getImplicitObjectArgument();

    if (subexprUsesMacros(
        expr->getCallee(),
        srcMgr))
    {
        printWarning(
            expr->getExprLoc(),
            srcMgr,
```

```

"Skipping macro expansion: " +
sourceCode(expr, srcMgr, langOpts)
);
return;
}

string logger;
if (dyn_cast<CXXThisExpr>(loggerExpr)) {
    logger = "this";
} else {
    if (loggerExpr->getType()
        .getTypePtr()->isPointerType())
    {
        logger =
            string("(") +
            sourceCode(
                loggerExpr,
                srcMgr,
                langOpts) +
            string(")");
    } else {
        logger =
            string("&(") +
            sourceCode(
                loggerExpr,
                srcMgr,
                langOpts) +
            string(")");
    }
}
std::stringstream macro;
if (expr->getMethodDecl()
    ->getNameAsString() == "Trace")
{
    macro << "TRACE(";
} else {
    macro << "ERROR(";
}
macro << logger;
for (unsigned i=0; i<expr->getNumArgs(); i++)
{
    macro << ", (" +
        << sourceCode(
            expr->getArg(i),
            srcMgr,
            langOpts)
        << ")";
}
macro << ")";
if (listOnly) {
    printCoords(
        outs(),
        expr->getExprLoc(),
        srcMgr)
    << ,
    << sourceCode(expr, srcMgr, langOpts)
    << "\n\n"
    << macro.str() << "\n\n";
} else {
    replacementStrings.push_back(macro.str());
    replacements.insert(
        Replacement(
            srcMgr,
            sourceRange(expr, srcMgr),
            StringRef(
                replacementStrings.back())))
}
);
}
}

```

Pierwsze linie callbacku pokazują nam wszystkie dane, które otrzymujemy: SourceManager, LangOptions i znalezione dopasowanie.

Rozpoczynamy od wyciągnięcia znalezionego wywołania funkcji składowej. StatementMatcher znajduje elementy abstrakcyjnego drzewa składowego, a więc obiekty typu Stmt, jednak wiemy, że memberCallExpr znajduje tylko obiekty typu CXXMemberCallExpr, toteż możemy wykonać dynamiczne rzutowanie (getNodeAs<T>).

Teraz zdecydujemy, czy zaakceptować wywołanie, jeśli zostało napisane z użyciem makr. Jak już powiedziano, chcemy po-

minąć te wywołania, w których obiekt zależy od tego, jak zdefiniowano makro, oraz te, w których nazwa funkcji zależy od makr. Natomiast nie sprawdzamy, czy argumenty funkcji są zależne od makr. CXXMemberCallExpr::getImplicitObjectArgument daje nam dostęp do podwyrażenia, które odpowiada obiektowi. Jak widzieliśmy w przykładowym pliku wejściowym, może być skomplikowane. Na przykład (Logger::smLoggers + 1). Może też być niejawnym this. Do podwyrażenia, które określa wywoływaną funkcję, dostajemy się przez getcallee. Na obu podwyrażenach uruchamiamy nasz test subexprUsesMacros.

Powiedzmy, że akceptujemy wywołanie. Teraz chcemy napisać makro, którym je zastąpimy. Za pomocą getMethodDecl()->getNameAsString() dowiemy się, czy to funkcja Trace, czy Error, a więc czy makro ma zaczynać się od TRACE, czy ERROR. Musimy obsłużyć kilka sposobów, w jaki może być podany obiekt. Może być jawnym lub niejawnym this. Może być dany wyrażeniem, które zwraca wskaźnik do Logger – albo referencję. Do referencji chcemy dopisać operator &. Z kolei na niejawnym this nie zadziała funkcja sourceCode.

To, czy obiekt jest dany jako jawnie lub niejawnie this, możemy rozpoznać po typie, jaki go reprezentuje w abstrakcyjnym drzewie składowym. Wyrażenie this (jawnie i niejawnie) jest reprezentowane przez CXXThisExpr. To, czy Expr jest w rzeczywistości obiektem CXXThisExpr, da się sprawdzić między innymi funkcją dynamicznego rzutowania llvm::dyn\_cast<T> dostępną w całym projekcie LLVM. Jeśli rzutowanie nie jest możliwe, zwraca ona nullptr. Niezależnie od tego, czy jest to this jawnie, czy niejawnie, postępujemy z nim tak samo: pierwszym argumentem naszego makra staje się this.

Jeśli jednak obiekt jest dany nie przez wyrażenie this, a inny rodzaj wyrażenia, sprawdzamy, czy zwraca ono wskaźnik, czy referencję do Logger. Typ zwracany przez wyrażenie uzyskujemy przez loggerExpr->getType().getTypePtr(). Wygląda to dosyć dziwnie, bo jak to, pobieramy typ, a potem jakiś wskaźnik do niego? To niezbyt szczęśliwe nazewnictwo wynika stąd, że getType() zwraca QualType, a więc typ razem z kwalifikatorami (const, volatile, restrict). QualType zwiera pola bitowe wskazujące kwalifikatory i wskaźnik do faktycznego typu, który pobieramy przez getTypePtr(). Mając już (niekwalifikowany) typ, możemy określić, czy jest wskaźnikiem, czy referencją do Logger za pomocą isPointerType(). W przypadku referencji poprzedzamy pierwszy argument naszego makra operatorem &, tak aby zawsze był on wskaźnikiem.

Jak widać w kodzie, najłatwiejsza jest w naszym przykładzie obsługa argumentów wywołania: pobieramy je przez getNumArgs i getArg, ujmujemy w nawiasy i dopisujemy do makra. Proste, choć trzeba przyznać, że nie poradzi sobie z wyrażeniami, w których jedno makro rozwija się do kilku argumentów, na przykład Error("Line %d, column %d", COORDS).

Na końcu pozostałe nam tylko dodać do replacements informację, że chcemy podmienić fragment kodu odpowiadający znalezionemu wywołaniu na tekst makra, który zbudowaliśmy. Poniżej StringRef to tylko wskaźnik do bufora i długość, musimy gdzieś przechować teksty makr, aby były dostępne dla RefactoringTool. Temu służy replacementStrings. Jeśli działamy w trybie listOnly, nie rejestrujemy zmian w Replacements, ograniczamy się do wypisania tego, co zrobiliśmy, na standardowe wyjście.

Jak widać, praca z abstrakcyjnym drzewem składowym jest całkiem łatwa. Zwłaszcza w porównaniu z zawiłościami SourceLocation i makr. Otrzymujemy struktury danych pozwalające nie

przejmować się nam szczegółami zapisu. Posiadamy szczegółowe informacje o typach danych, o powiązanych deklaracjach. Wiemy, który wariant przeciążonej funkcji został wybrany – tego nie da się osiągnąć grepem! Możemy nawet obliczyć wartości `constexpr`.

Elementy łączymy w funkcji `main()`. `CommonOptionsParser`, jak już napisano, jest automagicznie skonfigurowany i rozpozna nasz przełącznik `-l`. To, czy przełącznik jest włączony, sprawdzamy niżej, w `if (listOnly)`. `CommonOptionsParser` sparsuje też standardowe opcje kompilatora, które zostaną użyte przez `LibTooling` do skonfigurowania *front-endu*. Podobnie, z `CommonOptionsParser` otrzymamy listę plików do „skompilowania”.

#### Listing 11. Funkcja main

```
int main(int argc, const char **argv)
{
    CommonOptionsParser optionsParser(
        argc, argv, categoryExampleTool);
    RefactoringTool refactoringTool(
        optionsParser.getCompilations(),
        optionsParser.getSourcePathList());

    ExampleCallback callback(
        refactoringTool.getReplacements());
    MatchFinder finder;
    finder.addMatcher(
        loggerInvocationMatcher, &callback);

    int ret;
    auto f = newFrontendActionFactory(&finder);
}
```

```
if (listOnly) {
    ret = refactoringTool.run(f.get());
} else {
    ret = refactoringTool.runAndSave(f.get());
}

return ret;
}
```

## URUCHAMIANIE

Wszystko gotowe. Teraz możemy zbudować projekt LLVM, a nasze narzędzie zbuduje się z nim. Program uruchomimy tak:

```
article-example -l test_source.cc --
```

Jeśli w `test_source.cc` jest kod z Listingu 1, wszystko powinno zadziałać. Z przełącznikiem `-l` otrzymamy listę zmian na standarde wyjście i ostrzeżenia o pominiętych wywołaniach z makrami. Bez przełącznika – zmiany zostaną zapisane w pliku i będzie wyglądał jak na Listingu 2. Ostrzeżenia nadal będą wypisywane. Ponieważ żeby sparsować plik, uruchamiamy *front-end* kompilatora, to jeśli w pliku znajdują się błędy składniowe lub semantyczne, *front-end* wypisze komunikaty błędów, dokładnie takie same, jakie wyprodukowałby kompilator Clang. Czym są te dwa minusy na końcu? Za nimi podajemy parametry dla kompilatora: ścieżki do plików nagłówkowych (`-I`), definicje makr (`-D`) itd. Dla przykładu pliku wejściowego, którym się posłużyliśmy, nie potrze-

reklama

# Obserwuj nas na Twitterze



@PROGRAMISTAMAG



bujemy nic ustawać, więc pozostawiliśmy listę pustą. Generalnie jednak te parametry są bardzo ważne, bo od nich zależy interpretacja kodu.

Tak uruchamiamy nasze narzędzie na pojedynczym pliku, ale obiecaliśmy przecież automatyczne przetwarzanie ogromnych projektów w całości. W dużym projekcie mamy mnóstwo plików źródłowych, komplikowanych z różnymi parametrami. Parametry często zależą od tego, którą z kilku konfiguracji wybierzemy.

Jeżeli budujemy nasz ogromny projekt za pomocą make'a, możemy wykorzystać istniejące *makefile'e* do automatycznego uruchomienia naszego narzędzia na każdym pliku źródłowym. Musimy tylko nieco zmodyfikować reguły uruchamiające kompilator. Na szczęście nasze narzędzie przyjmuje takie same parametry jak kompilator. Przyjmuje nawet te nie mające wpływu na *front-end* (na przykład -o) – po prostu je ignoriuje. Typowa reguła wygląda następująco:

#### Listing 12. Typowa reguła uruchamiająca kompilator

```
$(TARGETOBJ)/%.o: %.cc  
$(CREATE_OBJ)  
$(CCC) $(CCF) $( $(*F)_LOCAL_CCFLAGS ) \  
$(CCPF) $( $(*F)_LOCAL_CCPPFLAGS ) \  
$(LOCAL_INCPATH) $(GLOBAL_INCPATH) \  
$(WERRORFLAG) \  
-c -o $(TARGETOBJ)/$(@F) <  
-c -o $(TARGETOBJ)/$(@F) <
```

Wystarczy zamienić kompilator na nasze narzędzie i przenieść parametry za --. Nie zaszkodzi dodać parametru \$(LIST\_ONLY), żeby móc włączyć tryb -I, nie zaszkodzi też usunąć opcji -c i -o. Potem wystarczy wystartować make, tak jak zawsze. Mniejsza o to, że pliki .o nie powstaną. Jeśli mamy kilka konfiguracji, trzeba zdecydować, która jest tą „kanoniczną”.

#### Listing 13. Przykładowa zmieniona reguła

```
$(TARGETOBJ)/%.o: %.cc  
$(CREATE_OBJ)  
$(TOOL) $(LIST_ONLY) $< -- \  
$(CCF) $( $(*F)_LOCAL_CCFLAGS ) \  
$(CCPF) $( $(*F)_LOCAL_CCPPFLAGS ) \  
$(LOCAL_INCPATH) $(GLOBAL_INCPATH) \  
$(WERRORFLAG)
```

Narzędzie oparte na Clangu całkiem łatwo poradzi sobie z projektem, który normalnie jest budowany za pomocą GCC. Clang stara się zachować kompatybilność z GCC. Przede wszystkim rozumie jego opcje linii poleceń. Tego nie musimy poprawiać w *makefile'u*. Clang wspiera również rozszerzenia GCC: wbudowane funkcje, wbudowane makra. Ma to znaczenie zwłaszcza wtedy, kiedy próbujemy wczytać Clangiem systemowe pliki nagłówkowe stworzone dla GCC. Niemniej możemy natrafić na kilka niespodzianek. Narzędzie może odrzucać kod, który GCC akceptuje, zasypując nas trudnymi do zinterpretowania komunikatami błędów.

Jednym z możliwych problemów są inne domyślne ścieżki, w których szuka się systemowych plików nagłówkowych. Być może, na

przykład, nasz projekt używa kilku niestandardowych rozszerzeń do STL i nie ma ich w plikach nagłówkowych, z których korzysta Clang. Tak może się zdarzyć zwłaszcza wtedy, kiedy nie polegamy na domyślnym zainstalowanym kompilatorze, tylko mamy własną wersję GCC, którą zawsze budujemy nasz projekt.

Z tym problemem możemy poradzić sobie, podając naszemu narzędziu odpowiednie ścieżki za pomocą parametru -I. Ma on pierwszeństwo przed ścieżkami domylnymi, a jeśli umieścimy go przed innym -I, będzie miał pierwszeństwo również przed nimi. Zmodyfikowana reguła może wyglądać tak:

#### Listing 14. Reguła ze ścieżkami do systemowych plików nagłówkowych

```
$(TARGETOBJ)/%.o: %.cc  
$(CREATE_OBJ)  
$(TOOL) $(LIST_ONLY) $< -- \  
$(SYSTEM_INCLUDE_OVERRIDE) \  
$(CCF) $( $(*F)_LOCAL_CCFLAGS ) \  
$(CCPF) $( $(*F)_LOCAL_CCPPFLAGS ) \  
$(LOCAL_INCPATH) $(GLOBAL_INCPATH) \  
$(WERRORFLAG)
```

Gdzieś indziej musimy umieścić w zaznaczonym makrze listę parametrów -I. Ścieżki, które należy tam wpisać, i ich kolejność przeszukiwania, możemy poznać, na przykład wydając polecenie:

```
nasz_gcc/bin/g++ -E -x c++ -v - < /dev/null
```

Druga niespodzianka wiąże się z pewnym obszarem, w którym twórcy Clanga zdecydowali nie utrzymywać zgodności z GCC. Chodzi o rozszerzenia do wykonywania zwektoryzowanych obliczeń (SIMD, SSE itp., miesiące się plikach nagłówkowych takich jak altivec.h, pmmINTRIN.h itp.). O ile GCC ma dla każdej operacji osobną funkcję wbudowaną, Clang, jeśli tylko może, opiera się na operatorach. Dlatego jeśli plik źródłowy włącza, bezpośrednio lub pośrednio, któryś z tych nagłówków w wersji dla GCC, Clang sobie nie poradzi. Żeby rozwiązać ten problem, to chociaż chcemy używać systemowych nagłówków z GCC, musimy zrobić wyjątek dla tych dotyczących zwektoryzowanych obliczeń. One koniecznie muszą pochodzić od Clanga. Krótko mówiąc, należy dodać jeszcze jeden parametr -I przed parametrami, które już dodaliśmy. Interesujące nas pliki nagłówkowe znajdziemy w zbudowanym projekcie LLVM w podkatalogu *lib/clang/3.6.1/include*. Są tam też inne pliki nagłówkowe, więc warto utworzyć osobny katalog, tylko dla tych plików, które chcemy podmienić.

## W sieci

- ▶ Wnętrze Clanga – <http://clang.llvm.org/docs/InternalsManual.html>
- ▶ Dokumentacja API – <http://clang.llvm.org/doxygen>
- ▶ API ASTMatchers – <http://clang.llvm.org/docs/LibASTMatchersReference.html>
- ▶ O zwektoryzowanych operacjach – [http://clang.llvm.org/compatibility.html#vector\\_builtins](http://clang.llvm.org/compatibility.html#vector_builtins)



### TOMASZ KARWALA

Informatyk, niedoszły matematyk. Od zawsze zainteresowany egzotycznymi językami programowania i ich teorią. Od blisko 3 lat pracuje w krakowskim Centrum R&D firmy Ericsson. Zajmuje się bardziej programowaniem systemowym w Uniksach, niż sieciami komórkowymi. Od czasu do czasu przychodzi mu grzebać w kompilatorach, co przysparza mu sporo satysfakcji.

II EDYCJA

# Security PWNING Conference 2017

WARSZAWA | 6-7 LISTOPADA 2017



Wspólnie z Gynvaelem Coldwindem,  
Przewodniczącym Rady Programowej, zapraszamy Was  
do udziału w **II edycji Security PWNING Conference**,  
konferencji poświęconej problematyce współczesnego hackingu  
oraz bezpieczeństwa IT.

Security PWNING Conference 2017  
jest wydarzeniem podczas którego:

- koncentrujemy się na technicznych aspektach bezpieczeństwa informatycznego,
- korzystamy z wiedzy i doświadczenia najlepszych ekspertów z Polski i Europy,
- solidną dawkę wiedzy uzupełniamy czasem na dyskusje i rozmowy z prelegentami,
- zapewniamy element rywalizacji poprzez udział w zawodach mini CTF
- a na koniec zapraszamy na after party!

RABAT

Kod promocyjny:  
**20Programista**

Kod upoważnia  
do rejestracji

**z 20 %  
rabatem**

(ważny do 8.10.)

WIĘCEJ INFORMACJI:  
[www.institutpwn.pl/pwning](http://www.institutpwn.pl/pwning)

INSTYTUT  PWN

# Na granicy światów – technologia bezpieczeństwa ARM TrustZone

Bezpieczeństwo systemów komputerowych to temat bardzo aktualny w dzisiejszych czasach, ponieważ coraz więcej wrażliwych i wartościowych informacji jest przechowywanych i transportowanych w świecie cyfrowym. Ciemna strona mocy nie śpi, poświęca mnóstwo środków na kreowanie coraz to bardziej wymyślnych i skutecznych ataków. Projektanci systemów komputerowych próbują sprostać wymaganiom rynku i przeciwdziałać tym niecnym zamiarom. W tej wojnie liczy się przede wszystkim kreatywność, elastyczność i ciągłe zmiany podejścia. Na tle rozwiązań konkurencji ciekawie prezentuje się firma ARM ze swoją technologią TrustZone, dającą stosunkowo dużą swobodę projektantom chipów oraz inżynierom bezpiecznego oprogramowania. Przyjrzyjmy się bliżej koncepcji dwóch światów i temu, co o sprzęcie musi wiedzieć programista, aby w pełni wykorzystać jego możliwości chronienia wrażliwych zasobów. Za architekturę referencyjną posłuży nam ARMv8.

## CO TO ZNACZY BEZPIECZEŃSTWO

Bezpieczeństwo systemu jest pojęciem abstrakcyjnym i niejednoznacznym, ponieważ pokrywa szeroki zakres właściwości. Trzeba jednak spróbować w jakiś sposób je skonkretyzować. Można powiedzieć, że bezpieczeństwo ma zapewnić, iż wartościowe zasoby systemu nie mogą być skopiowane, zniszczone (zmienione) ani udostępnione osobom, które nie powinny być do tego upoważnione. Artykuł traktuje na temat technologii bezpieczeństwa wykorzystywanych na etapie projektowania systemów komputerowych. Jakie pomysły na zabezpieczanie systemów mają producenci sprzętu elektronicznego?

## ARCHITEKTURY BEZPIECZEŃSTWA

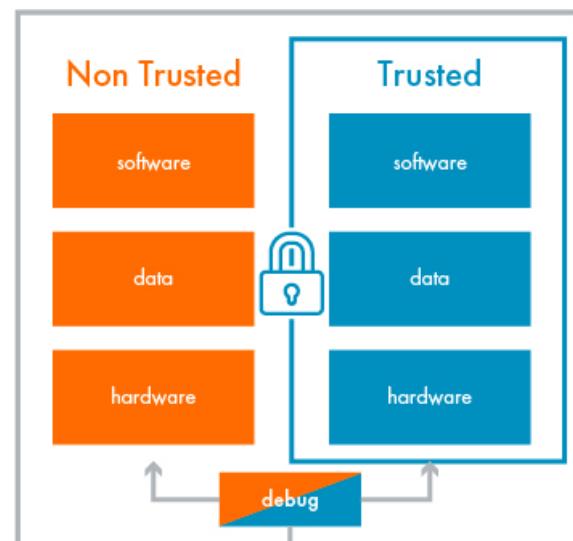
Klasycznym pomysłem jest zastosowanie dedykowanego modułu zupełnie na zewnątrz SoCa (ang. *System on Chip*). Przykładem może być tutaj stosowanie karty SIM wykorzystywanej przez operatorów sieci komórkowych do tego, żeby dane urządzenie było sparowane z konkretną siecią telefoniczną. Innym przykładem zewnętrznego modułu są też karty do dekoderów telewizyjnych.

Drugie podejście to zastosowanie dedykowanego modułu zintegrowanego wewnętrz chipu. Takie urządzenie to zazwyczaj jedno z dwóch – albo bloczek realizujący operacje kryptograficzne oraz przechowywający klucze służące do podpisywania i szyfrowania, albo procesor ogólnego przeznaczenia działający równolegle do podstawowej jednostki CPU. Ten dodatkowy procesor jest wykorzystywany do realizacji różnych zadań na wrażliwych zasobach systemu.

## POMYSŁ NA BEZPIECZEŃSTWO WEDŁUG ARMA

Czym różni się technologia TrustZone od powyżej wymienionych architektur? Generalnie koncepcja jest prosta – zamiast stosować

zewnętrzny moduł sprzętowy, bądź też wydzielić jedno urządzenie wewnątrz SoCa, technologia bezpieczeństwa zostanie zaszczepiona w każdym układzie chipu już na etapie projektowania. Architekt systemu osobiście wybiera elementy, które zostaną oznaczone jako zaufane. Dzięki temu możliwe jest podzielenie systemu na dwa równoległe światy – normalny (*Non-secure, Non-trusted* albo *Normal*) i bezpieczny (*Secure* albo *Trusted*). Idea ta jest przedstawiona na Rysunku 1. Wszystkie zasoby, które mają być chronione, muszą zostać umieszczone w świecie bezpiecznym. Sprzęt musi zapewniać, aby dostęp ze świata normalnego do zasobów świata bezpiecznego był odpowiednio kontrolowany. W dalszej części artykułu opisane są poszczególne komponenty systemu komputerowego, opartego na procesorze ARMv8, w kontekście technologii bezpieczeństwa.



Rysunek 1. Idea dwóch światów – normalnego i bezpiecznego w technologii ARM TrustZone (źródło: <https://www.arm.com/products/security-on-arm/trustzone>)

# Let us bring you home

[www.semihalf.com](http://www.semihalf.com)

Semihalf creates software for advanced solutions in the areas of operating systems, virtualization, networking and storage. We make software which is tightly coupled with the underlying hardware to achieve maximum system capacity for running at scale. We offer a dynamic and open yet principled work environment with exceptional engineering challenges. Our partners and customers are semiconductor industry leaders mainly from Silicon Valley.

Join us for a deep dive into the latest microprocessor technologies and high performance software development.

Currently we are looking for

## Kernel Hacker

(Krakow)

### Your challenges

You will be responsible for bringing up new hardware and creating low-level software running on multicore processors. You will be hacking on operating system kernel (BSD, Linux), writing device drivers and optimizing for the best performance results. Your code will often be submitted to the open source repositories.

### Requirements

- ④ Fluency in C code development and debugging
- ④ Low level coding experience (Linux or BSD kernel, bootloaders)
- ④ x86 or ARM assembly experience
- ④ Handling of standard shell utilities and tools like GCC, GDB, GIT, DTrace

### Benefits



Flexible working hours aligned to your individual preferences



Small teams (2-6 persons) with real influence on projects



Individual yearly training budget



Unique employee profit sharing programme



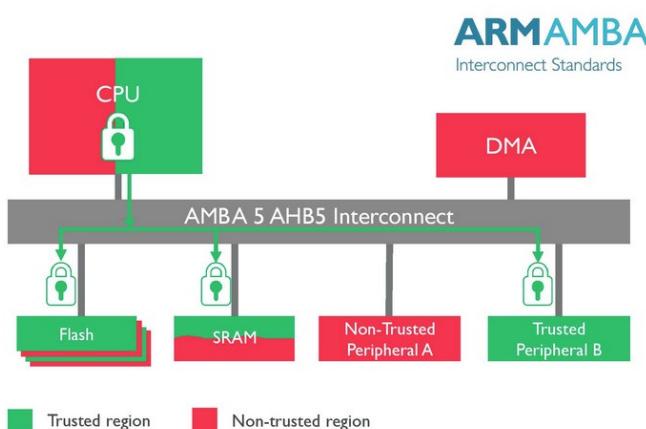
Social package, medical health care, multisport card



Chillout room, game console

## SZYNA SYSTEMOWA

Najbardziej istotnym dodatkiem, jaki wprowadza TrustZone do projektu szyny systemowej, jest dodatkowy sygnał kontrolny oznaczony jako *non-secure* bit. Można patrzeć na niego jak na dodatkową linię adresową. Wszystkie urządzenia główne magistrali systemowej (ang. *master device*), czyli np. CPU, podczas wykonywania transakcji odczytu i zapisu, muszą odpowiednio ustawić wspomniany bit, w zależności od swojej konfiguracji. W szczególności urządzenia ze świata normalnego podczas transakcji ustawiają *non-secure* bit w stan wysoki. Kiedy próbują zaadresować urządzenie podrzędne (ang. *slave device*) ze świata bezpiecznego, transakcja jest odrzucana, ponieważ adres z ustawionym bitem *non-secure* nie zostanie sparowany z żadnym urządzeniem na szynie. Jeżeli chodzi o transakcje w drugą stronę, tj. *secure device* próbuje dostać się do urządzenia *non-secure*, sytuacja zależy od konkretnej implementacji. W większości przypadków elementy świata bezpiecznego mają dostęp do wszystkich zasobów systemu. Szyna systemowa w architekturze ARM zaprojektowana jest zgodnie ze specyfikacją AMBA (Advanced Microcontroller Bus Architecture) – najnowsza wersja to AMBAv5, natomiast projekt bitu *non-secure* opisany jest w specyfikacji protokołu AXI (Advanced eXtensible Interface). Obrazowo spojrzenie na szynę systemową w kontekście technologii bezpieczeństwa zaprezentowane jest na Rysunku 2.



Rysunek 2. Schematyczne przedstawienie szyny systemowej zgodnej ze specyfikacją AMBA z uwzględnieniem technologii TrustZone (źródło: <http://www.anandtech.com/show/9775/arm-announces-armv8m-instruction-set>)

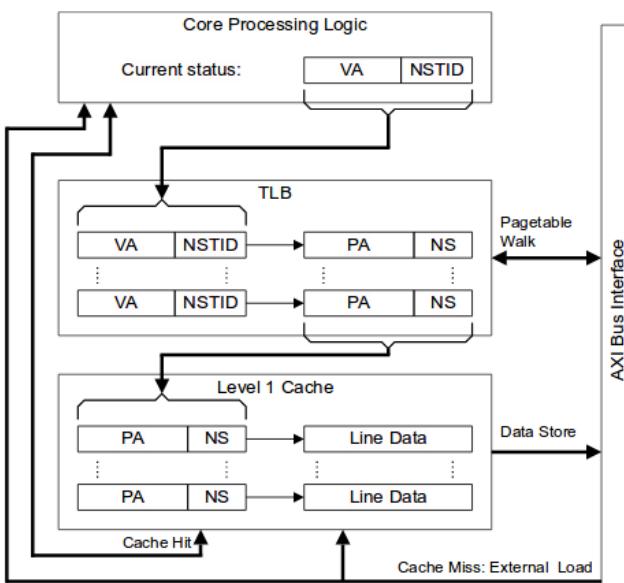
## PAMIĘĆ

Do szyny systemowej podłączony jest między innymi kontroler pamięci<sup>1</sup>. Jest to niezwykle istotne urządzenie, które powinno umożliwić programową konfigurację regionów *secure* oraz *non-secure* RAM'u. Decyzja o tym, czy urządzenie próbujące dostać się do wrażliwych danych powinno dostać na to zgodę, jest podejmowana w oparciu o wyżej opisany *non-secure* bit. Warto podkreślić, że architektura podsystemu pamięci podrzecznej musi być odpowiednio zaprojektowana. Gdyby był to „pospolity” *cache*, procesor w trybie *non-secure* mógłby dostać się do danych znajdujących się w pamięci podrzecznej, mimo że są to dane wrażliwe. Działyby się tak, gdyż dostęp do pamięci podrzecznej pomija kontroler pamięci. Ten problem można rozwiązać programowo, czyszcząc całą zawar-

tość pamięci podrzecznej podczas przechodzenia z jednego świata do drugiego. Rozwiążanie to spowodowałoby jednak spadek wydajności systemu, na które współczesne komputery nie mogą sobie pozwolić. Aby temu zapobiec, tagi pamięci podrzecznych są wyposażone w dodatkowy bit *non-secure*, który pozwala ustalić, z którego świata pochodzi zawartość danej linii pamięci *cache*.

Proces dostępu do danych w systemie pamięci wyposażonej w technologię TrustZone przedstawiony jest schematycznie na Rysunku 3. Kolejność zdarzeń jest następująca:

- » Procesor wykonuje zapis/odczyt danych<sup>2</sup> z pamięci. Sprzęt przekazuje adres wirtualny (VA – Virtual Address) oraz identyfikator świata (NSTID – Non-Secure Table Identifier) do jednostki zarządzania pamięcią (MMU – Memory Management Unit), aby przeprowadziła translację. Oczywiście, jeżeli procesor działający w trybie *non-secure* zechce dostać się do danych uprzywilejowanych, zostanie zwrócony odpowiedni błąd.
- » MMU wyszukuje w buforze zawierającym fragmenty tablicy stron (TLB – Translation Lookaside Buffer) bądź też przechodząc po wpisach w tablicy stron w pamięci głównej odpowiadający adres fizyczny (PA – Physical Address). Bit *non-secure* znajdujący się we wpisach w tablicy stron musi zgadzać się z podanym przez procesor NSTID.
- » Po znalezieniu odpowiedniego mapowania adres fizyczny wraz z bitem *non-secure* zostaje przekazany do pamięci podrzecznej. Jeżeli dane znajdują się w pamięci *cache*, są one od razu udostępniane. W innym przypadku musi zostać wykonany dostęp do pamięci – oczywiście z uwzględnieniem ustawienia bitu *non-secure* na szynie systemowej.



Rysunek 3. Pierwszy poziom pamięci podrzecznej w przykładowym rdzeniu ARMv8 (źródło: PRD29-GENC-009492C)

## URZĄDZENIA PERYFERYJNE

Nie tylko pewne obszary pamięci mogą być zarezerwowane jako zaufane. Teoretycznie wszystkie urządzenia peryferyjne w obrębie systemu mogą występować w domenie *secure* bądź *non-secure*.

2. Dotyczy to również procesu zaciągania instrukcji, ponieważ pamięć podrzeczna instrukcji również wyposażona jest w logikę *security*. Na przykład odpowiednie ustawienia rejestrów systemowych pozwalają na zablokowanie możliwości zaciągania instrukcji z pamięci *non-secure*, kiedy procesor pracuje w trybie *secure*.

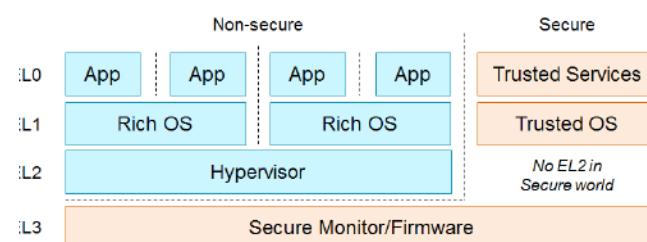
1. Urządzeniem rozgraniczającym dostęp *secure* i *non-secure* do pamięci RAM nie musi być koniecznie sam kontroler. Może to być również North Bridge Unit bądź kontroler Last Level Cache. Zależy to od implementacji, jednak zasada działania jest podobna.

Szyna urządzeń peryferyjnych działa nieco odmiennie niż szyna systemowa, ponieważ nie zawiera ona dodatkowego bitu *non-secure*. Technologia TrustZone zaimplementowana jest w mostku łączącym szynę systemową z szyną urządzeń zewnętrznych. Mostek ten ma za zadanie odrzucać transakcje z niewłaściwymi ustawieniami trybu bezpieczeństwa i nie przekazywać ich dalej do urządzeń. Jakie może być zastosowanie zaufanego sprzętu? Przykładem będzie system z dwoma portami UART. Jeden z nich (UART0) będzie służył jako zwykła konsola, z której korzysta system operacyjny, natomiast drugi (UART1) stanie się „tajnym” kanałem komunikacyjnym, na którym komunikaty wyświetla program działający w tle i wykonujący instrukcje w trybie *secure*. UART1 musi pozostać przezroczysty dla dostępów ze świata niezaufanego, tj. zapisy do jego rejestrów ignorowane, a odczyty zawsze zwracały wartość 0.

## PROCESOR

Procesory architektury ARMv8 mogą pracować na kilku poziomach wyjątków (Exception Levels – ELx, gdzie „x” oznacza identyfikator poziomu od 0 do 3). Poziomy te są odseparowane od siebie sprzętowo, a także posiadają własne zestawy niektórych rejestrów systemowych – m.in. konfiguracyjnych oraz takich jak wskaźnik ramki stosu (SP\_ELx) i licznik programu (PC\_ELx).

Zaimplementowanie dwóch światów dla oprogramowania wykonywanego na jednym procesorze rozwiązane jest w ten sposób, że jeden rdzeń logiczny podzielony jest na dwa rdzenie wirtualne. Sprawia to, że architektura staje się jeszcze bardziej skomplikowana. Istnieją już nie tylko tryby EL0-EL3, ale oprócz tego S-EL1 i S-EL0 (Secure EL1/Secure EL0), co daje w sumie liczbę sześciu<sup>3</sup> różnych trybów, w jakich potencjalnie może pracować procesor ARMv8. W zrozumieniu tego mechanizmu może pomóc Rysunek 4, na którym przedstawiono poszczególne tryby procesora i typowe przypisanie oprogramowania do każdego poziomu wyjątku.



Rysunek 4. Przykładowy stos programowy architektury ARMv8 (źródło: <https://community.arm.com/processors/f/discussions/6774/exceptions-levels-in-the-armv8-architecture>)

W świecie normalnym znajdują się procesy i aplikacje – poziom EL0 – korzystające z oddzielnych zasobów, które odseparowane są od jądra systemu<sup>4</sup> operacyjnego – EL1. Ponad tym pracuje hipernadzorca (ang. *hypervisor*) w uprwywilejowanym trybie EL2. W obszarze zaufanym poziomy S-EL0 oraz S-EL1 są wykorzystywane analogicznie na bezpieczne aplikacje i bezpieczny system operacyjny (przykładem może być tutaj OP-TEE<sup>5</sup>). Istnieją również rozwiązania, gdzie rezygnuje się z systemu operacyjnego działającego w S-EL1, a aplikacje działające w trybie S-EL0 wykonywane są jako niezależne (ang. *standalone*). Zwornikiem między oboma

światami jest poziom EL3, wraz z oprogramowaniem, tzw. Secure Monitor'a, które to ma za zadanie nadzorować bezpieczeństwo systemu. Secure Monitor jest przede wszystkim odpowiedzialny za przełączanie między światami secure/non-secure. W trybie EL3 działa oprogramowanie przeprowadzające inicjalizację platformy, ponieważ ARMv8 uruchamia się na tym poziomie wyjątków.

Wracając do tego, co zostało powiedziane na temat szyny systemowej, można zadać sobie pytanie – w jaki sposób urządzenia na tej szynie rozpoznają, czy dostępy inicjowane przez procesor wykonywane są w domenie *secure* bądź *non-secure*? Bit NS na magistrali jest „wyciągany” bezpośrednio z trybu, w jakim obecnie znajduje się procesor, tj. dla EL0-EL2 zostaje ustawiony, natomiast dla EL3, S-EL1 oraz S-EL1 jest on gaszony. Dzięki temu oprogramowanie nieuprwywilejowane wykonywane np. w trybie EL1 (jądro systemu operacyjnego) nie ma dostępu do zasobów świata zaufanego.

## PRZECHODZENIE PRZEZ GRANICE ŚWIATÓW

Każdy z trybów procesora<sup>6</sup> zawiera swój wektor:

- » wyjątków do obsługi przerwań IRQ,
- » wyjątków do obsługi przerwań FIQ,
- » wyjątków synchronicznych (Synchronous External Abort),
- » przerwań wywołanych przez błędy systemowe (System Error Interrupt).

Warto zauważyć, że poziom EL3 jest najbardziej uprwywilejowany, co wydaje się logiczne, skoro jest to decydująca o całosciowym bezpieczeństwie platformy sfera. Mianowicie procesor może być skonfigurowany w taki sposób, że wszystkie, bądź tylko wybrane z wyżej wymienionych wyjątków, będą przełączały procesor automatycznie w tryb EL3. Dzięki takiemu rozwiązaniu Secure Monitor może nadzorować pracę innych trybów oraz całego systemu.

W jaki sposób można zmienić kontekst świata procesora? Jedyna droga prowadzi poprzez oprogramowanie Secure Monitor, tj. poziom EL3. Wydaje się to jedynym słusznym rozwiązaniem, ponieważ sposób przechodzenia między światami to potencjalnie bardzo wrażliwa z punktu widzenia bezpieczeństwa płaszczyzna. Synchronicznie poziom wyjątków można „podnieść” poprzez instrukcję SMC (Secure Monitor Call) oraz „obniżyć” poprzez instrukcję ERET (Exception RETurn). Zakładając, że aktualnie procesor jest na poziomie EL1, a trzeba przeprowadzić jakąś operację dostępną ze świata *secure* w trybie S-EL1, w pierwszej kolejności należy wykonać operację SMC, co zmieni poziom wyjątków na EL3. Oprogramowanie Secure Monitor'a dokona sprawdzenia, czy żądanie nie narusza zasad bezpieczeństwa, i jeśli wszystko jest w porządku – skonfiguruje odpowiednio rejestrze SPSR\_EL1 oraz ELR\_EL1 i wykona instrukcję ERET, która przeniesie procesor do trybu S-EL1. Powrót wyglądać będzie analogicznie.

Asynchroniczna<sup>7</sup> zmiana świata jest również możliwa, poprzez implementację dwóch światów przerwań.

## PRZERWANIA ZAUFANE I NIEZAUFANE

Korzystając z faktu, że oprogramowanie wykonywane w trybie EL3 jest w stanie przechwytywać przerwania IRQ oraz FIQ, niezależnie od tego, w jakim świecie aktualnie działa procesor, istnieje możli-

6. Oprócz EL0 i S-EL0, które „korzystają” z wektora wyjątków odpowiednio EL1 i S-EL1.

7. Zasadniczo będzie to pół-asynchroniczna operacja w takim sensie, że istotnie egzekucja zostanie przeniesiona do kodu EL3 asynchronicznie, jednak powrót z EL3 do któregokolwiek poziomu wyjątku do drugiego świata będzie wykonany poprzez instrukcję ERET.

3. Zapowiadane przez ARM rozszerzenie ARMv8.4 ma wprowadzać implementację również dla S-EL2, co dawałoby liczbę 7.

4. Będzie też systemów w liczbie mnogiej, jeżeli mamy do czynienia z wieloma maszynami wirtualnymi.

5. Open Portable Trusted Execution Environment – więcej na [www.op-tee.org](http://www.op-tee.org).

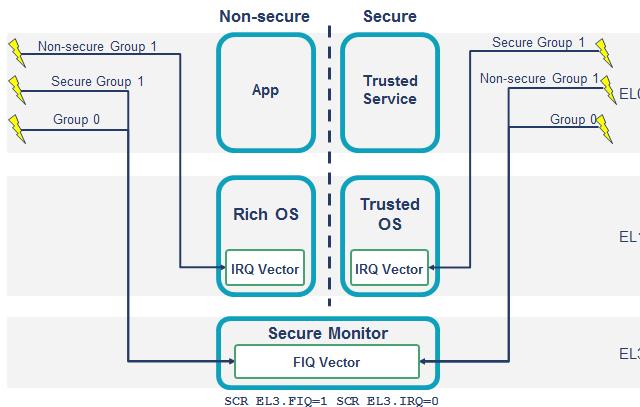
wość implementacji elastycznego systemu przerwań, które mogą zostać podzielone na bezpieczne i normalne. Ponadto niezbędny jest także kontroler przerwań świadomy technologii TrustZone. Dla chipów z procesorami ARMv8 jest to GIC (Generic Interrupt Controller) w wersji 3.

Jakie są założenia projektowe dla podsystemu przerwań w układzie komputerowym wspierającym technologię bezpieczeństwa? Przede wszystkim przerwania *secure* powinny być zupełnie niewidoczne dla domeny *non-secure*. Następne oczywiste założenie jest takie, że obsługa przerwań z określonego świata powinna wykonywać się w kontekście tego świata. Idąc dalej, z uwagi na wydajność systemu, przerwania normalne, podczas gdy procesor działa w trybie *non-secure*, powinny być niewidoczne dla świata bezpiecznego, aby niepotrzebnie „nie zwracać mu głowy” – uniknie się w ten sposób dwóch operacji zmiany kontekstu. Jak wygląda realizacja tych założeń?

Interfejs pomiędzy procesorem ARMv8 a kontrolerem przerwań jest bardzo prosty – składa się z dwóch linii FIQ oraz IRQ. Zazwyczaj konfiguracja będzie wyglądać w ten sposób, że przerwanie IRQ ustawi licznik instrukcji procesora na wektor przerwań dla aktualnego w momencie przyjęcia przerwania poziomu wyjątku – na przykład gdy przerwanie przyjdzie na linii IRQ, podczas gdy procesor jest na poziomie EL1, uruchomi się obsługa przerwania z wektora wyjątków zarejestrowanego dla poziomu EL1. Linia FIQ posłuży jako sposób raportowania przerwań skierowanych do świata przeciwnego do tego, w którym procesor aktualnie wykonuje instrukcje. Mapowanie pomiędzy źródłami przerwań a docelowym światem, w którym ma być obsłużone dane przerwanie, jest konfigurowane w rejestrach GLCa. Na Rysunku 5 przedstawiono, jak wygląda przekierowywanie przerwań z poszczególnych poziomów wyjątków. Źródło przerwań można zarejestrować jako jedno z trzech typów:

- » **Normalna grupa 1** – kierowane do świata normalnego i obsługiwane na poziomie EL1,
- » **Bezpieczna grupa 1** – kierowane do świata zaufanego i obsługiwane na poziomie S-EL1,
- » **Grupa 0** – kierowane do Secure Monitor'a – obsługiwane w trybie EL3.

## Routing example



Rysunek 5. Przykładowy model przekierowywania przerwań w systemie opartym o ARMv8 i GICv3 (źródło: <https://community.arm.com/processors/b/blog/posts/programmable-interrupt-controllers-a-new-architecture>)

Po co aż trzy typy przerwań, skoro zarówno przerwania z grupy 0, jak i bezpiecznej grupy 1 będą obsługiwane w świecie *secure*? Prze-

rwania grupy 0 są wykorzystywane jedynie do obsługi specyficznych urządzeń, które zasadniczo mają służyć pracy samego Secure Monitor'a. Może to być np. architektonalny timer, pozwalający na okresowe sprawdzanie stanu systemu. Generalnie kod wykonywany w EL3 jest bardzo istotny z punktu widzenia bezpieczeństwa (w końcu to najbardziej uprzywilejowany tryb), a co za tym idzie – warto zredukować jego objętość do minimum. W ten sposób ograniczona zostanie również potencjalna płaszczyzna ataku. Ponadto w trakcie egzekucji w trybie EL3 wszystkie przerwania są wyłączone<sup>8</sup>, co może zbytnio wydłużyć czas oczekiwania na przerwania kierowane do światów *secure* bądź *non-secure*. Jest to bardzo poważny problem szczególnie dla systemów operacyjnych, które są oparte o działanie schedulera wykorzystującego przerwania timer'owe – zbyt długie zablokowanie takiego przerwania może spowodować zaburzenie ich pracy.

## ZAKOŃCZENIE

Należy pamiętać, że niezależnie od tego, jak duże nakłady czasowe i finansowe zostaną poświęcone na projektowanie i wdrażanie technologii bezpieczeństwa, zawsze znajdzie się ktoś, kto będzie w stanie poświęcić więcej i być w stanie złamać istniejące zabezpieczenia. Najważniejsze jest to, żeby zasoby były chronione adekwatnie do swojej wartości, tak aby potencjalny atak był po prostu nieopłacalny. Technologia ARM TrustZone nie narzuca projektantom twardych ram, w których ma zamykać się jego linia obrony – to od kunsztu programistycznego zależy, jak bezpieczny będzie jego system. Atakujący skupi się na najbardziej wrażliwej płaszczyźnie, dlatego warto być świadomym, co dzieje się na granicy światów.

## W sieci:

- ▶ [http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C\\_trustzone\\_security\\_whitepaper.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf) – dokument o tworzeniu bezpiecznych systemów w oparciu o technologię ARM TrustZone
- ▶ [http://www.gstt.ece.ufl.edu/courses/fall15/eel4720\\_5721/labs/refs/AXI4\\_specification.pdf](http://www.gstt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf) – specyfikacja ARM AXI
- ▶ <http://www.slideshare.net/linaroorg/arm-trusted-firmware-for-armv8alcu13> – prezentacja nt. stosu oprogramowania architektury ARMv8
- ▶ [http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM\\_DEN0028B\\_SMC\\_Calling\\_Convention.pdf](http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM_DEN0028B_SMC_Calling_Convention.pdf) – opis instrukcji SMC, pozwalającej na synchroniczne zmianianie światów
- ▶ <https://github.com/ARM-software/arm-trusted-firmware/blob/master/docs/firmware-design.md> – szczegółowy opis ARM Trusted Firmware, czyli przykładowej implementacji Secure Monitora



**JAN DĄBROS**

jsd@semihalf.com

Programista systemów wbudowanych w krajobrazowej firmie Semihalf. Obecnie zajmuje się tworzeniem firmware'u dla platform serwerowych opartych na architekturze ARM. Pasjonat programowania niskopoziomowego. Prywatnie mąż i miłośnik astronomii.

8. Nie jest to wymóg architektonalny, natomiast rozwiązanie zalecane przez ARM.

CAN U  
ESCAPE?

[14-15/11/2017] KNH@wro\$



# Office UI Fabric

„Czy ta aplikacja może wyglądać jak Excel?”. Takie pytanie może zadać użytkownik końcowy Twojej aplikacji, który przyzwyczał się do interfejsu Office i z tego powodu oczekuje podobnych rozwiązań również i w innych aplikacjach. Z punktu widzenia programisty rozwiązanie tego problemu polega na odpowiednim przygotowaniu warstwy wizualnej. To zadanie można jednak znacznie uprościć w oparciu o narzędzie Office UI Fabric, które dostarcza style i komponenty umożliwiające szybkie tworzenie aplikacji wyglądających jak Office. W tym artykule pokażę, w jaki sposób wykorzystać Office UI Fabric do zaimplementowania aplikacji ASP.NET Core 2.0 w architekturze Single Page Application, obsługiwanej przez bibliotekę React.js.

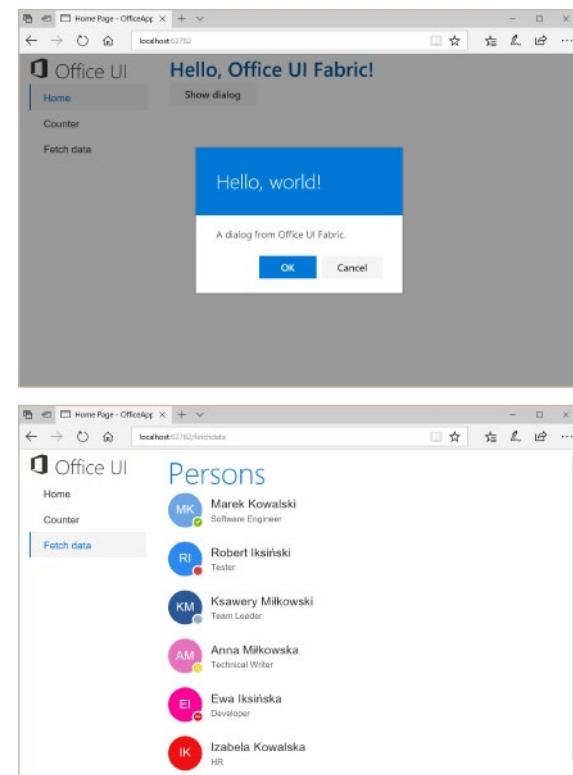
## WPROWADZENIE

O tym, jak istotna jest intuicyjność interfejsu użytkownika aplikacji, można się łatwo przekonać, gdy widzimy, jak kilkuletnie dziecko doskonale radzi sobie z obsługą nowoczesnego smartfonu. Niemal automatycznie rozpoznaje i interpretuje wszystkie ikony, dzięki czemu może korzystać z wielu aplikacji, nie potrafiąc jeszcze czytać. Z przeciwną sytuacją spotykamy się w przypadku starszych użytkowników, dla których obsługa komputera nie zawsze jest ulubioną czynnością. Można im jednak ułatwić korzystanie z nowych aplikacji poprzez dostarczenie interfejsu użytkownika, który pokrywa się z tymi, które już dobrze znają. Zakładając, że w większości przypadków aplikacje pakietu Office są znane (chociażby na podstawie liczby opinii o Microsoft Word w Google Play), tworzenie interfejsu zgodnego z Office może być bardzo przydatne. Tym bardziej że Microsoft udostępnił Office UI Fabric. Jest to darmowe narzędzie umożliwiające tworzenie aplikacji internetowych i mobilnych w oparciu o gotowe komponenty, znane z aplikacji Office.

W tym artykule pokażę, w jaki sposób wykorzystać Office UI Fabric w warstwie wizualnej aplikacji ASP.NET Core 2.0 o nazwie *OfficeApp*. Tę aplikację zaimplementuję w architekturze Single Page Application, obsługiwanej przez React.js, aby uzyskać efekt końcowy przedstawiony na Rysunku 1. Jak widać, aplikacja *OfficeApp* składa się z bocznego paska nawigacyjnego. Wygląd tego paska jest taki sam jak w aplikacjach Office 365 (np. OneDrive). Pasek nawigacyjny umożliwia przełączanie pomiędzy widokami: Home, Counter oraz Fetch data. Jak za chwilę pokażę, te widoki pochodzą z szablonu aplikacji ASP.NET Core with React.js. Widoki zmodyfikuję w taki sposób, aby w panelu Home umieścić etykietę oraz przycisk. Kliknięcie przycisku spowoduje wyświetlenie okna dialogowego. Z kolei w widoku Fetch data utworzę listę, w której każdy element będzie prezentował dane osoby z wykorzystaniem kontrolek Office UI Fabric.

## TWORZENIE PROJEKTU

Implementację aplikacji rozpoczęłem od instalacji narzędzi. Mianowicie: node.js oraz .NET Core 2.0. Jako edytor kodu źródłowego wykorzystuję Visual Studio 2017 Community w wersji 15.3.5. Po przygotowaniu narzędzi utworzyłem nowy projekt aplikacji *OfficeApp* w Visual Studio, a następnie wybrałem szablon React.js (Rysunek 2). Aplikację bazującą na tym szablonie można również utworzyć z linii komend z wykorzystaniem polecenia `dotnet new`

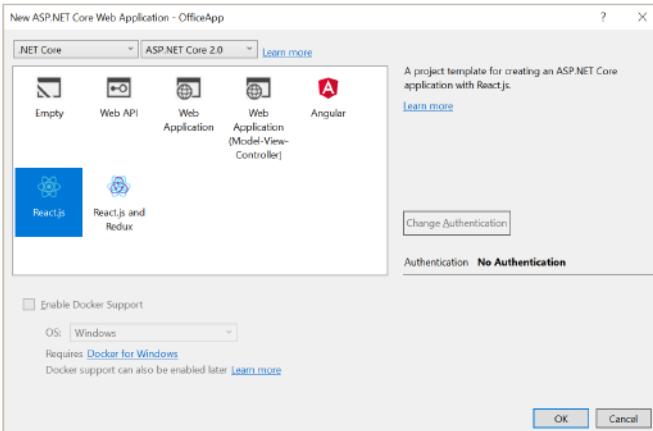


Rysunek 1. Przykładowe widoki aplikacji *OfficeApp*, którą utworzę w tym artykule

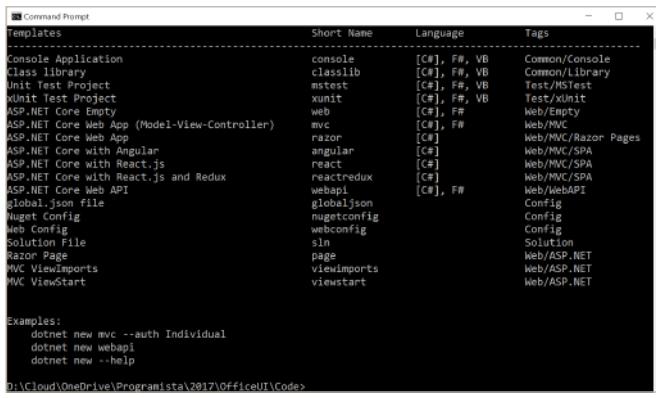
`react`. Wydanie komendy `dotnet new` wyświetli listę wszystkich dostępnych szablonów (Rysunek 3). Jest to przydatne w sytuacji, gdy korzystamy z innej platformy niż Windows, np. Mac. Nawiasem mówiąc, wszystkie przedstawione tu informacje są oczywiście kompatybilne z platformami Mac i Linux, ze względu na wieloplatformowość bibliotek .NET Core.

## STRUKTURA PROJEKTU

Niezależnie od wybranego sposobu tworzenia projektu jego struktura w obu przypadkach będzie identyczna. Jak pokazano na Rysunku 4, struktura projektu jest typowa dla aplikacji ASP.NET MVC Core. Punkt wejścia jest zaimplementowany w metodzie `Main` klasy `Program`. Konfiguracja middleware odbywa się za pomocą klasy `Startup`. Projekt zawiera również dwa kontrolery: `SampleData`

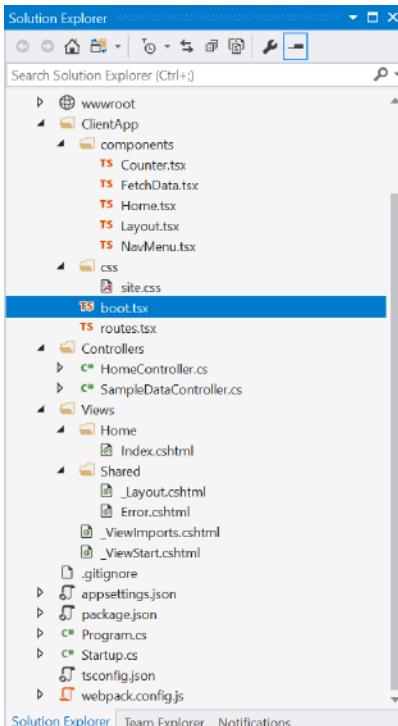


Rysunek 2. Lista szablonów aplikacji .NET Core 2.0



Rysunek 3. Lista szablonów aplikacji .NET Core 2.0 w wierszu poleceń

i Home. Pierwszy implementuje prosty serwis Web API, który wykorzystamy w dalszej części artykułu, a drugi implementuje dwie metody: Index i Error. Pierwsza z nich jest skojarzona z widokiem głównym (Home/Index.cshtml) o definicji przedstawionej w Listingu 1.



Rysunek 4. Struktura projektu aplikacji OfficeApp

### Listing 1. Kod źródłowy widoku Index kontrolera Home

```
@{
    ViewData["Title"] = "Home Page";
}

<div id="react-app">Loading...</div>

@section scripts {
    <script src="~/dist/main.js" asp-append-version="true"></script>
}
```

### Listing 2. Inicjalizacja aplikacji React.js

```
function renderApp() {
    const baseUrl = document.getElementsByTagName('base')[0]
        .getAttribute('href')!;
    ReactDOM.render(
        <AppContainer>
            <BrowserRouter>
                children={ routes }
                basename={ baseUrl } />
        </AppContainer>,
        document.getElementById('react-app')
    );
}
```

Jak to przedstawiono w Listingu 1, widok Index kontrolera Home zawiera jeden blok o identyfikatorze react-app. Do tego bloku zostanie wstrzyknięta aplikacja SPA React.js zaraz po jej inicjalizacji. Jest to realizowane w ramach funkcji renderApp, zaimplementowanej w pliku *ClientApp/boot.tsx* (Listing 2). Widzimy, że wykorzystywany jest tam komponent AppContainer, w ramach którego jest utworzony obiekt BrowserRouter. Ten ostatni zarządza routowaniem pomiędzy obiektami potomnymi, wskazanymi za pomocą atrybutu children. W tym przypadku atrybut ten wskazuje na stałą routes, zdefiniowaną w pliku *routes.tsx*:

```
export const routes = <Layout>
    <Route exact path="/" component={ Home } />
    <Route path='/counter' component={ Counter } />
    <Route path='/fetchdata' component={ FetchData } />
</Layout>;
```

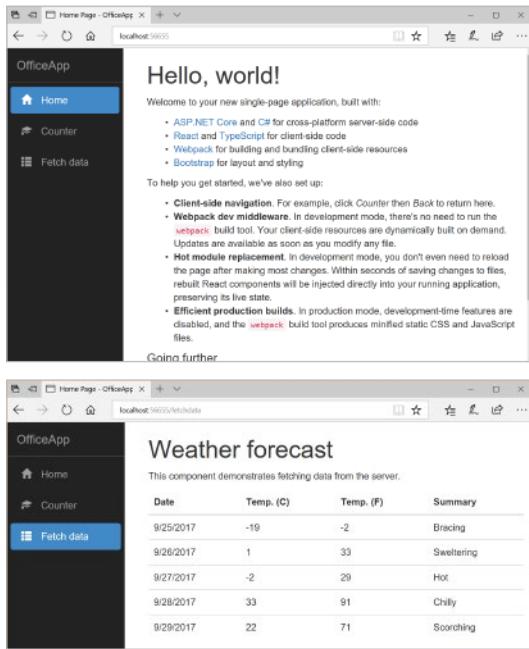
W kolejnym etapie nastąpi utworzenie komponentu Layout (Listing 3). Na jego definicję składa się jedna metoda render, w ramach której konstruowana jest główna struktura wizualna aplikacji. W tym przypadku składa się ona z dwóch kolumn (w kontekście bootstrapa) w stosunku 25:75. Pierwsza kolumna zawiera menu nawigacyjne, a druga prezentuje widoki komponentów: Home, Counter lub FetchData. Warto podkreślić, że komponent Home jest komponentem React.js i nie należy go mylić z kontrolerem Home. Po uruchomieniu aplikacji uzyskamy wynik przedstawiony na Rysunku 5.

### Listing 3. Główny layout witryny OfficeApp

```
export class Layout extends React.Component<LayoutProps, {}> {
    public render() {
        return <div className='container-fluid'>
            <div className='row'>
                <div className='col-sm-3'>
                    <NavMenu />
                </div>
                <div className='col-sm-9'>
                    { this.props.children }
                </div>
            </div>
        </div>;
    }
}
```

# PROGRAMOWANIE APLIKACJI WEBOWYCH

Każdy z komponentów jest zdefiniowany w ramach odpowiedniego pliku. Będziemy je modyfikować w kolejnym etapie.



Rysunek 5. Domyślne widoki aplikacji OfficeApp

## INSTALACJA PAKIETÓW

Przejdźmy teraz do zmodyfikowania aplikacji, aby uzyskać efekt przedstawiony wcześniej na Rysunku 1. Wymaga to zainstalowania Office UI Fabric w postaci pakietu npm. W tym celu skorzystałem z wiersza poleceń i w folderze aplikacji *OfficeApp* wydałem następującą komendę:

```
npm i --save office-ui-fabric-react
```

Następnie musiałem jeszcze doinstalować dwa pakiety @types/prop-types oraz @types/react w wersji 15.0.38:

```
npm i @types/prop-types
npm i @types/react@15.0.38
```

Wersja tego drugiego pakietu jest kluczowa dla zachowania kompatybilności z Office UI Fabric. W ostatnim kroku uzupełniłem jeszcze sekcję head pliku *\_Layout.cshtml* o referencję do stylów Office UI Fabric Core:

```
<link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/4.1.0/css/fabric.min.css">
```

## WIDOK HOME

Mając gotowe wszystkie elementy zależne i znając logiczną strukturę projektu, przystąpiłem do edycji pliku *Home.tsx*. Zmodyfikowałem go zgodnie z Listingiem 4.

Listing 4. Pełna definicja komponentu Home

```
import * as React from 'react';
import { RouteComponentProps } from 'react-router';
import {
  DefaultButton,
  PrimaryButton
} from 'office-ui-fabric-react/lib/Button';
```

```
import {
  Dialog,
  DialogType,
  DialogFooter
} from 'office-ui-fabric-react/lib/Dialog';

export class Home extends
  React.Component<RouteComponentProps<{}>, any> {
  constructor() {
    super();
    this.state = {
      isDialogVisible: false
    };
  }

  public render() {
    const divClasses =
      'ms-font-xxl ms-fontWeight-semibold'
      + ' ms-fontColor-themeDark';

    return (
      <div>
        <div className={divClasses}>
          Hello, Office UI Fabric!
        </div>

        <DefaultButton
          onClick={this.changeDialogVisibility.bind(this, true)}
          text='Show dialog' />

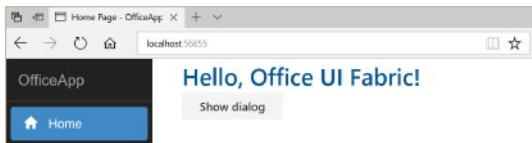
        <Dialog
          hidden={!this.state.isDialogVisible}
          onDismiss={this.changeDialogVisibility.bind(this, false)}
          dialogContentProps={{
            type: DialogType.largeHeader,
            title: 'Hello, world!',
            subText: 'A dialog from Office UI Fabric.'
          }}
          modalProps={{
            isBlocking: false,
            containerClassName: 'ms-dialogMainOverride'
          }}>
          <PrimaryButton
            onClick={this.changeDialogVisibility.bind(this, false)}
            text='OK' />
          <DefaultButton
            onClick={this.changeDialogVisibility.bind(this, false)}
            text='Cancel' />
        </DialogFooter>
      </div>
    );
  }

  private changeDialogVisibility(value: Boolean) {
    this.setState({ isDialogVisible: value });
  }
}
```

Zasadniczym elementem definicji komponentu *Home* jest metoda *render*, która jest odpowiedzialna za przygotowanie warstwy wizualnej. W tym przypadku składa się ona z trzech elementów (Rysunek 6):

- » statycznego łańcucha o treści Hello, Office UI Fabric – jest on formatowany w oparciu o trzy klasy z Office UI Fabric Core: *ms-font-xxl* (rozmiar), *ms-fontWeight-semibold* (po-grubienie) i *ms-fontColor-themeDark* (kolor). Nazwy tych klas zapisałem w stałej *divClasses*, aby uprościć listing;
- » przycisk utworzony za pomocą komponentu *DefaultButton* z Office UI Fabric. Jego kliknięcie spowoduje wywołanie metody *changeDialogVisibility*. Przełącza ona wartość właściwości *isDialogVisible*, która determinuje stan okna dialogowego;
- » okno dialogowe, reprezentowane przez komponent *Dialog*. Okno to prezentuje statyczny napis z dwoma przyciskami (Rysunek 1).

Dodatkowo w ramach klasy komponentu *Home* utworzyłem konstruktor, w którym definiuję jedną właściwość stanu aplikacji: *isDialogVisible*. Jej początkowa wartość to *false*. Z tego powodu dialog staje się widoczny dopiero po kliknięciu przycisku.



Rysunek 6. Zmodyfikowany widok Home

## KLASA PERSON

W kolejnym etapie przystąpiłem do implementacji widoku Fetch data. Ponieważ wykorzystuje on dane, które są ładowane dynamicznie z mikroserwisu, rozpoczęłem pracę od zdefiniowania klasy Person, będącej modelem reprezentującym poszczególne elementy w liście osób. W Listingu 5 pokazano pełną definicję klasy Person. Zawiera ona pięć automatycznie zaimplementowanych właściwości: FirstName, LastName, JobTitle, Presence i InitialsColor. Trzy pierwsze są typu string, a dwie ostatnie wykorzystują typy wyliczeniowe: Presence i InitialsColor (Listing 6). Reprezentują one stany obecności (Presence) oraz kolory ikon z inicjałami (InitialsColor). Stany obecności to małe ikonki w prawym górnym rogu ikony z inicjałami. Klasa Person definiuje również dwie właściwości FullName oraz Initials. Wykorzystuję je do prezentacji pełnego imienia i nazwiska oraz inicjałów. Ostatnim elementem klasy Person jest zbiór sześciu statycznych metod. Tworzą one poszczególne obiekty prezentowane następnie w liście. W ogólności pochodząby one ze źródła danych. Tutaj, dla uproszczenia, zdefiniowałem je statycznie.

Listing 5. Definicja klasy Person, będąca abstrakcyjną reprezentacją osoby w liście

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string JobTitle { get; set; }
    public Presence Presence { get; set; }
    public InitialsColor InitialsColor { get; set; }

    public string FullName
    {
        get => $"{FirstName} {LastName}";
    }

    public string Initials
    {
        get => $"{FirstName[0]}{LastName[0]}";
    }

    public static Person Marek()
    {
        return new Person()
        {
            FirstName = "Marek",
            LastName = "Kowalski",
            JobTitle = "Software Engineer",
            Presence = Presence.Online,
            InitialsColor = InitialsColor.LightBlue
        };
    }

    // Further definitions of sample persons
}
```

Listing 6. Definicje typów wyliczeniowych

```
public enum Presence
{
    None = 0, Offline, Online, Away, Dnd, Blocked, Busy,
}

public enum InitialsColor
{
    LightBlue = 0, Blue, DarkBlue, Teal, LightGreen,
    Green, DarkGreen, LightPink, Pink, Magenta,
    Purple, Black, Orange, Red, DarkRed
}
```

## KONTROLER WEB API

Po przygotowaniu klasy Person uzupełniłem definicję kontrolera SampleData o metodę typu Get o nazwie Persons (Listing 7). Metoda ta tworzy listę osób, wykorzystując do tego celu przygotowane wcześniej statyczne metody klasy Person (Listing 5).

Listing 7. Tworzenie listy osób do wyświetlenia

```
[HttpGet("[action]")]
public IEnumerable<Person> Persons()
{
    return new List<Person>
    {
        Person.Marek(),
        Person.Robert(),
        Person.Ksawery(),
        Person.Anna(),
        Person.Ewa(),
        Person.Izabela()
    };
}
```

## WIDOK LISTY OSÓB

Mając gotowy mikroserwis, przystąpiłem do zasadniczej implementacji komponentu FetchData (Listing 8). Zasadnicza struktura tego komponentu nie odbiega od analogicznej definicji z Listingu 4. Mianowicie klasa FetchData składa się z konstruktora, metody render oraz pomocniczej funkcji renderPersons. W konstruktorze klasy pobieram listę osób z mikroserwisu za pomocą metody fetch. Następnie otrzymane dane zapisuję we właściwości persons. Dodatkowo, w celu poinformowania użytkownika o tym, że lista jest pobierana z serwisu, wykorzystuję właściwość loading. Ma ona wartość false do momentu pobrania danych. Metoda render tworzy statyczną etykietę o wartości Persons. Pod tą etykietą, w zależności od wartości właściwości loading, prezentowany jest albo statyczny łańcuch Loading, albo faktyczna lista osób otrzymana z mikroserwisu (metoda renderPersons). Kolejne elementy tej listy są tworzone za pomocą komponentu Persona z Office UI Fabric. Wszystkie wykorzystane tam właściwości komponentu Persona ustawię na podstawie danych z mikroserwisu. W efekcie uzyskałem wynik przedstawiony na Rysunku 7.

Listing 8. Definicja komponentu FetchData

```
import * as React from 'react';
import { RouteComponentProps } from 'react-router';
import 'isomorphic-fetch';
import {
    Persona,
    PersonaSize,
    PersonaPresence,
    PersonaInitialsColor
} from 'office-ui-fabric-react/lib/Persona';

interface FetchDataExampleState {
    persons: Person[];
    loading: boolean;
}

export class FetchData extends React.Component<
    RouteComponentProps<{}>, FetchDataExampleState> {
    constructor() {
        super();
        this.state = { persons: [], loading: true };

        fetch('api/SampleData/Persons')
            .then(response => response.json() as Promise<Person[]>)
            .then(data => {
                this.setState({ persons: data, loading: false });
            });
    }

    public render() {
        let contents = this.state.loading
            ? <p><em>Loading...</em></p>
            :
```

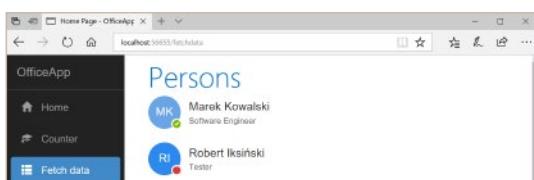
```

        : FetchData.renderPersons(this.state.persons);
    return <div>
      <div className='ms-font-su ms-fontColor-themePrimary'>
        Persons
      </div>
      { contents }
    </div>;
}

private static renderPersons(persons: Person[]) {
  return (
    <div>
      {persons.map(person =>
        <div>
          <Persona presence={person.presence}
            size={PersonaSize.regular}
            primaryText={person.fullName}
            secondaryText={person.jobTitle}
            imageInitials={person.initials}
            initialsColor={person.initialsColor} />
          <br />
        </div>)
      );
    }
}

interface Person {
  fullName: string;
  jobTitle: string;
  presence: number;
  initialsColor: number;
  initials: string;
}

```



Rysunek 7. Zmodyfikowany widok komponentu Fetch data

## MENU NAWIGACYJNE

Na koniec pozostało mi już jedynie zmodyfikowanie menu nawigacyjnego. W tym celu zastąpiłem domyślną zawartość pliku `Nav.tsx` tą z Listingu 9. Mianowicie domyślne menu nawigacyjne zastąpiłem komponentem `Nav` z Office UI Fabric. W celu wskazania odnośników tego menu wykorzystuję się atrybut `groups`. W przykładzie z Listingu 9 atrybut ten wskazuje na kolekcję trzech odnośników: `Name`, `Coutner` i `Fetch data`. Umożliwiają one wykonanie nawigacji do poszczególnych komponentów aplikacji SPA.

Dodatkowo ponad menu umieściłem ikonę Office oraz napis Office UI. Do utworzenia ikony wykorzystałem klasy `ms-Icon` oraz `ms-Icon--OfficeLogo`. W konsekwencji, po odświeżeniu aplikacji w przeglądarce, `OfficeApp` będzie prezentować się tak jak to pokazałem wcześniej na Rysunku 1.

### Listing 9. Definicja menu nawigacyjnego

```

import * as React from 'react';
import { Nav } from 'office-ui-fabric-react/lib/Nav';

export class NavMenu extends React.Component<{}, {}> {
  public render() {
    return (
      <div>
        <div className='ms-font-xxl'>
          <i className='ms-Icon ms-Icon--OfficeLogo'></i> Office
          UI
        </div>
        <Nav
          groups={[{
            links: [
              { name: 'Home', key: 'Home', url: '/' },
              { name: 'Counter', key: 'Activity', url: '/counter' },
            ]
          }]}
        </Nav>
      </div>
    );
  }
}

```

## UWAGI KOŃCOWE

Do napisania tego artykułu zainspirowała mnie prezentacja o Office UI Fabric na jednym ze stoisk podczas tegorocznej konferencji Microsoft Build w Seattle, WA. Przedstawione tu komponenty Office UI Fabric stanowią jedynie drobny kawałek pełnych możliwości tego narzędzia. Pełna lista komponentów wraz z przykładowym kodem jest dobrze udokumentowana na stronie projektu: [uifabric.io](http://uifabric.io). Dodatkowo cały kod Office UI Fabric wraz z przykładowymi aplikacjami jest dostępny jako open source w postaci repozytorium GitHub. Dzięki temu, gdy zajdzie taka potrzeba, można dowolnie dostosować Office UI Fabric do swoich potrzeb. Jak mówi przysłowie, jeden obraz jest wart tysiąc słów, więc jako dodatkową ciekawostkę zrobiłem zdjęcie torby na zakupy Office UI Fabric (Rysunek 8). Proszę zwrócić uwagę na szkic komponentu Persona, znajdujący się w prawym dolnym rogu.



Rysunek 8. Oficjalna torba na zakupy Office UI Fabric

## PODSUMOWANIE

W tym artykule zaprezentowałem, w jaki sposób od podstaw utworzyć aplikację ASP.NET Core 2.0 w architekturze Single Page Application, w której warstwa wizualna zbudowana jest w oparciu o Office UI Fabric. Wykorzystanie tych narzędzi pozwoliło mi na szybkie utworzenie aplikacji webowej opartej o najnowocześniejsze wzorce architektoniczne i komponenty wizualne. Chociaż do zaimplementowania SPA posłużyłem się React.js, to w ogólności Office UI Fabric można również wykorzystać w aplikacjach AngularJS, vanilla JavaScript, jak również w natywnych aplikacjach iOS Swift.

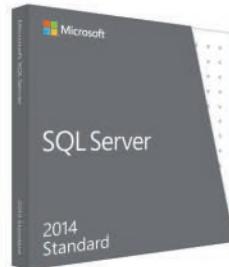
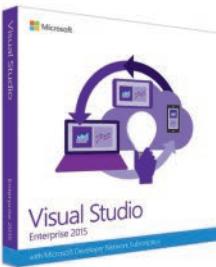
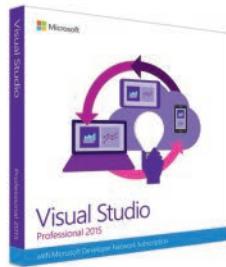
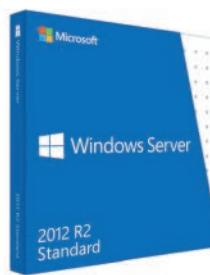
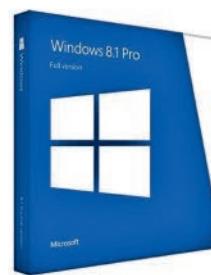
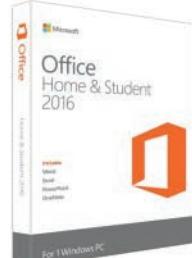
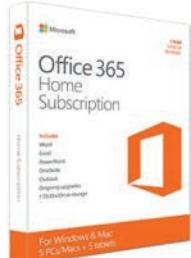


**DAWID BORYCKI**

Naukowiec, programista, autor wielu książek, amerykańskich patentów, artykułów i videotutoriali o programowaniu w różnych technologiiach. Pierwszy polski autor w Microsoft Press i MSDN Magazine. Z wykształcenia doktor fizyki teoretycznej. Obecnie pracuje w Polskiej Akademii Nauk, gdzie zajmuje się rozwojem nowoczesnych, nieinwazyjnych urządzeń do analizowania mikroprzepływu krwi w mózgu i obrazowania przez ośrodkie nieprzezroczyste optycznie.



**TTS Company rekomenduje oprogramowanie Microsoft ®**



**www.OprogramowanieKomputerowe.pl**

Microsoft Azure

Office 365

OneDrive

Więcej informacji: ☎ (22) 272 94 94 ✉ [sales@tts.com.pl](mailto:sales@tts.com.pl)

Sprzedaż



Dystrybucja



Import na zamówienie

# Co każdy webmaster o dostępności wiedzieć powinien

Coraz częściej mówi się o dostępności witryn internetowych. Niestety większość developerów nadal traktuje ten temat pobieżnie, a część z nich w ogóle nie wie, czym ona jest. Co oznacza pojęcie „dostępna strona” i dlaczego warto się tym tematem zainteresować?

## CO, JAK, PO CO IDLA KOGO?

Dostępność strony internetowej zakłada przede wszystkim to, że będzie ona dostosowana dla wszystkich użytkowników sieci. Szczególny nacisk kładzie się na jej przystosowanie do osób o wszelkich niepełnosprawnościach i schorzeniach – zarówno fizycznych, jak i psychicznych. Należy mieć na uwadze istnienie takich osób oraz ich potrzebę dostępu do informacji. Może się przecież zdarzyć, że po drugiej stronie znajduje się osoba, która nie widzi, bądź straciła rękę w wyniku wypadku, przez co nie może posługiwać się myszką. Według szacunków Międzynarodowej Organizacji Zdrowia na całym świecie żyje ponad miliard osób dotkniętych niepełnosprawnością, co stanowi ok. 15% populacji (dane za rok 2016). Należy pamiętać, że kiedy nadarzy nam się okazja tworzenia strony dla instytucji publicznej, mamy prawny obowiązek jej dostosowania do potrzeb osób niepełnosprawnych (Rozporządzenie Rady Ministrów w sprawie Krajowych Ram Interoperacyjności – Dz.U. 2016 poz. 113). W celu zunifikowania wymagań stawianych wobec dostępnych witryn internetowych powstał dokument WCAG (Web Content Accessibility Guidelines) – obecnie wersja 2.0 – który zawiera wskazówki dotyczące budowy dostępnych serwisów internetowych.

### Najczęstsze błędy developerów

1. Niestosowanie tekstu alternatywnego (atribut „alt”) lub ich zły dobór (np.: `` zamiast ``);
2. Zbyt niski kontrast elementów na stronie;
3. Niewłaściwe wykorzystywanie elementów semantycznych HTML5;
4. Niewłaściwe posługiwanie się nagłówkami;
5. Stosowanie zbyt dużej liczby animacji.

## PROJEKTOWANIE STRONY

Strona już na etapie projektowania powinna się charakteryzować odpowiednim podejściem do tematu dostępności. Przede wszystkim należy pamiętać o niestosowaniu zbyt dużej ilości animacji. Choć są one nieodzowną częścią współczesnego web designu, wypada pamiętać o zdrowym umiarze, gdyż mogą one negatywnie wpływać na osoby dotknięte epilepsią bądź nerwicą. Kolejną bolączką stron, już na etapie tworzenia layoutu, jest zbyt niski kontrast elementów, przez co osoby niedowidzące mogą mieć spory problem z przeczytaniem opublikowanych przez nas treści. Pojęcie „odpowiedniego kontrastu” jest jednak względne. To, co dla dobrze widzącej osoby jest „wystarczające”, może sprawiać problemy

osobom z wadą wzroku. Według wspomnianej normy WCAG 2.0 za minimalny uznaje się kontrast na poziomie 4,5 do 1, a zalecaný to 7 do 1. Zalecenie to dotyczy nie tylko tekstu, ale także obrazków zawierających tekst. Tutaj może nauszać się wniosek, że im wyższy kontrast, tym lepiej. Otóż nie do końca. Badania pokazują, że dla osób z dyslekcją łatwiejsze jest czytanie tekstu na tle o niezbyt wysokim poziomie kontrastu. W związku z tym nie zaleca się stosowania jaskrawych kombinacji (m.in. czarny + biały lub czarny + żółty). Do pomiaru kontrastu możemy wykorzystać szereg stworzonych do tego celu programów (m.in. Colour Contrast Analyser lub narzędzia dostępnego na stronie webaim.org).

Niemal na każdej stronie widzimy zdjęcia – czy to w galerii, czy na podstronach w celu przedstawienia produktu lub usługi. Dobre zdjęcie niejednokrotnie potrafi przekazać więcej niż słowa! Ważne więc jest ich odpowiednie oznaczenie. Służy temu atrybut alt. Zdecydowanie zbyt często traktowany pobieżnie – niestosowany w ogóle lub źle dobrany. Wartość tego atrybutu powinna jednoznacznie mówić o tym, co przedstawia dane zdjęcie.

Mimo że standardem we współczesnym web developmentie jest stosowanie języka JavaScript, należy mieć na uwadze, aby w przypadku, gdy przeglądarka użytkownika z jakichś powodów nie obsługuje tego języka, nasza strona nadal potrafiła dostarczyć odbiorcy treść. Sytuacji, w których skrypt JS nie zostanie uruchomiony, może być kilka (np. błąd w nowej wersji przeglądarki lub zerwanie połączenia z serwerem w trakcie dosyłania użytkownikowi plików JS). Słownem: skrypt JS nie może blokować ani uniemożliwić wyświetlenia treści. Takie podejście nosi nazwę Progressive Enhancement – zakłada całkowitą niezależność warstw (treści, metadanych, wyglądu oraz mechaniki działania) oraz ich właściwą komunikację. Treść, jaką chcemy przekazać, musi bezwzględnie trafić do osób chcących ją otrzymać! Pamiętajmy, że treść pozbawiona „fajerwerków” tworzonych poprzez CSS, JS itd. nadal jest wartościowa (a jeśli nie jest, to czas najwyższy coś z tym zrobić).

Współczesne strony często zawierają dynamiczne sekcje, których zawartość zmienia się po wykonaniu określonej akcji przez użytkownika. Przykładem takiego rozwiązania są m.in. zakładki czy też akordeony. Język HTML nie przewiduje na tę okoliczność specjalnych znaczników, dlatego też wyżej wspomniane elementy budowane są ze znaczników niesemantycznych takich jak `<div></div>`, czy `<span></span>`. Dla osób niewidomych bądź niedowidzących stanowią one problem, tym bardziej że ich działanie nie jest wspierane przez technologie asystujące. Aby nadrobić te niedociągnięcia i naprowadzić czytniki na właściwy trop, stosuje się atrybuty ARIA. Dzięki nim możemy dodać semantyczne znaczenie dowolnym elementom stronie (w tym również bardziej zaawansowanym dodatkom).

## Podstawowe atrybuty ARIA

1. Role (np. `role="tab"`) – przeznaczone do opisywania przeznaczenia danego elementu
  - » `alert`
  - » `slider`
  - » `tooltip`
  - » `tab`
  - » `toolbar`
  - » `banner`
  - » `search`
2. Stany i właściwości
  - » `aria-hidden`
  - » `aria-label`
  - » `aria-checked`
  - » `aria-readonly`
  - » `aria-required`
  - » `aria-atomic`
  - » `aria-busy`
  - » `aria-dropeffect`
  - » `aria-grabbed`
  - » `aria-controls`

Jeśli w jakimś miejscu na naszej stronie treść jest generowana dynamicznie poprzez skrypty języka JavaScript, warto rozważyć zastosowanie znacznika `<noscript></noscript>`, który w przypadku niezaładowania skryptów pokaże użytkownikowi domyślną, wcześniej ustaloną treść. Takie rozwiązanie sprawi, że nie zostawimy użytkownika „z niczym”.

Na tym etapie powinniśmy się też zastanowić nad rozmieszczeniem tekstu, odpowiednim zredagowaniu tytułów oraz podzieleniu treści na odpowiednie sekcje. Treść główna strony powinna posiadać trafne tytuły, które pomogą użytkownikowi w zorientowaniu się w jej treści. Tekst powinien zostać podzielony na tematyczne sekcje (`<section></section>`) opatrzone nagłówkami (`<header></header>`). Wspomniane tytuły powinny posiadać odpowiednie oznaczenie (h1-h6), które określa ich hierarchię. Zaleca się także, aby sam tekst był odpowiednio sformatowany. Mile widziane jest justowanie tekstu jedynie do strony lewej, gdyż jego rozcięgnięcie może wywoływać wrażenie „ściany tekstu” uniemożliwiającej jego przeczytanie osobom z dyslekcją. Jeśli w tekście znajdują się linki, należy je odpowiednio wyróżnić, aby przeglądający mógł je bez trudu zauważyc. W przypadku gdy na naszej stronie udostępniamy materiały audio lub wideo, powinniśmy udostępnić również alternatywne wersje dla osób niesłyszących bądź niewidomych (np. transkrypt filmu).

## DOSTĘPNE MENU

Jednym z kluczowych elementów na każdej stronie jest menu. Zarówno osobom zdrowym, jak i niepełnosprawnym pomaga w odnalezieniu się na stronie. Jednak niewłaściwie zaprojektowane lub wdrożone może powodować więcej problemów niż jego brak. W swoich założeniach dostępne menu powinno być intuicyjne w obsłudze, a jego znalezienie nie powinno sprawiać nikomu najmniejszego problemu.

Zacznijmy od tego, że menu powinno być zbudowane z wykorzystaniem elementów semantycznych HTML5. Mowa tutaj o znaczniku `<nav></nav>`.

Kolejną ważną kwestią jest umożliwienie użytkownikowi korzystanie z elementów menu, nawet gdy ten nie ma możliwości korzystania z myszki. Osoby dotknięte schorzeniami ruchowymi takimi jak np. choroba Parkinsona często zmuszone są korzystać z alternatywnych urządzeń wejścia, takich jak head wand, mouth stick czy też single switch access oraz oczywiście klawiatura (zwłaka bądź brajlowska).

Wszystkie te urządzenia mimo zdecydowanie różniącej się budowy opierają się na podobnej zasadzie działania.

Najprostszą metodą stworzenia dostępnego menu jest korzystanie z semantycznych znaczników HTML5, takich jak `<a href="#">Odnosnik</a>` lub `<button>Przycisk</button>`. Te elementy z uwagi na swoje semantyczne znaczenie od razu umożliwiają dostęp poziomu klawiatury. Niestety na niektórych stronach nadal panuje semantyczny bałagan, przez co osoby posługujące się klawiaturą nie mają łatwego zadania, aby dostać się w wybrane przez siebie miejsce na stronie. W Internecie znajdziemy też „kwiatki”, które w nazbyt jasny sposób mogą posłużyć jako antywzorzec. Przyjrzyjmy się jednemu z nich:

### Listing 1. Przykład niewłaściwego wykonania menu

```
<div id="menu">
<ul>
<li onclick="window.location.href='http://niesemantycznastrona.pl'">Główna</li>
<li onclick="window.location.href='http://niesemantycznastrona.pl/o-nas'">O nas</li>
<li onclick="window.location.href='http://niesemantycznastrona.pl/kontakt'">Kontakt</li>
</ul>
</div>
```

Jak widać powyżej, nie został zastosowany ani jeden element semantyczny, który pomógłby (nawet w najmniejszym stopniu) w nawigacji. Ponadto w sytuacji, gdy użytkownik ma wyłączoną

reklama



# devstyle.pl

ŚWIAT OKIEM PROGRAMISTY

obsługę skryptów w przeglądarce, elementy listy stają się kompletne bezużyteczne (już pomijając wykorzystanie archaicznej metody przypisania zdarzenia – mowa tutaj o „onclick”).

Aby powyższe menu stało się semantyczne, wystarczyłoby do każdego elementu listy dodać odnośnik (np: <a href="http://niesemanticznastrona.pl">Główna</a>), a div o id="menu" zastąpić przez znacznik <nav></nav>.

Niemal każde menu może poszczycić się przeróżnymi animacjami, które sygnalizują użytkownikowi najechanie kursorem na dany jego element. Warto także te animacje przystosować do obsługi poprzez klawiaturę. W tym celu oprócz standardowego selektora :hover zastosować można również :focus. Wówczas po wskazaniu danego elementu za pomocą klawiatury (przycisk Tab) zostanie pokazany identyczny efekt jak wówczas, gdy najedziemy na niego kursem myszy.

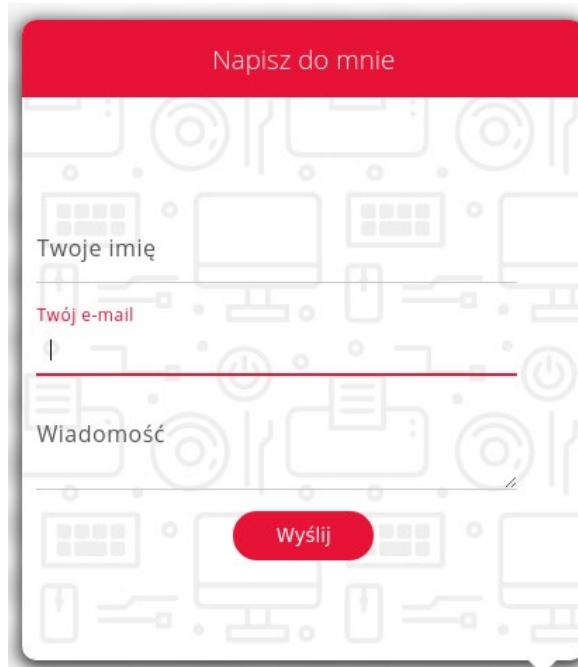
Coraz popularniejszym trendem staje się stosowanie ukrytego menu, dostępnego po kliknięciu odpowiedniego przycisku (tzw. hamburgera). Pamiętajmy, aby ten element strony również był prawidłowo zbudowany. Przede wszystkim by taki przycisk posiadał własną, odpowiadającą mu etykietę. Można to osiągnąć poprzez wpisanie jej wewnątrz znacznika button (<button>Menu</button>). W przypadku gdy jest to niemożliwe (gdy np. przycisk jest tylko grafiką i nie ma w nim miejsca na tekst), powinniśmy zastosować atrybut aria-label (<button aria-label="Otwórz menu"></button>). Podobnie jak w przypadku elementów menu, również przycisk jego otwarcia powinien otrzymywać widoczny focus klawiatury. Jeśli zastosowaliśmy poprawny element semantyczny (<button></button>), nie musimy się martwić, gdyż otrzymuje on focus nawet bez naszej ingerencji. Sam focus powinien być na tyle widoczny, aby po jego wybraniu osoba korzystająca z naszej strony nie miała wątpliwości co do jego stanu. Dobrą praktyką jest ułożenie w kodzie menu bezpośrednio pod przyciskiem otwierającym. W ten sposób po kliknięciu w przycisk focus zostanie automatycznie przeniesiony na pierwszy element menu. Na koniec warto poinformować użytkownika o stanie przycisku za pomocą atrybutu aria-expanded. Za jego pomocą informujemy użytkownika o tym, czy menu jest otwarte, czy zamknięte (<button aria-expanded="false" type="button"></button>). Wartość tego atrybutu powinna być zatem kontrolowana przez skrypt języka JavaScript.

## DOSTĘPNE FORMULARZE

Formularze na stronie są niezwykle istotne z punktu widzenia utrzymywania kontaktu z Klientem. Ich odpowiednie zaprojektowanie oraz prawidłowa implementacja umożliwia zachowanie prawidłowego przebiegu komunikacji również z osobami dotknietymi niepełnosprawnością.

Jedną z najczęściej pomijanych kwestii dotyczących formularzy są odpowiednie oznaczenia pól oraz przycisków w nich używanych.

W większości współczesnych stron na potęgę wykorzystywany jest atrybut placeholder. Jest on o tyle nieprzyjazny dla odwiedzających, że zazwyczaj występuje sam (bez odpowiadającemu polu znacznika label). Powoduje to, że gdy uaktywnimy pole zawierające jedynie ten atrybut, możemy zwyczajnie zapomnieć, co mieliśmy w to pole wpisać. Poprawnie oznaczone pole powinno zawierać odpowiadającą mu etykietę (<label for="">Treść etykiety</label>, gdzie for="" ma zawierać nazwę odpowiadającego mu pola). Co ważne: taki zabieg wcale nie musi oszpecić naszej strony. Wręcz przeciwnie – istnieje wiele pomysłów na kreatywne wykorzystanie znacznika <label></label>.



Rysunek 1. Przykład zastosowania znacznika <label></label>

Podobnie jak w przypadku menu, każdy element wykorzystany w formularzu kontaktowym powinien zawierać widoczny dla użytkownika focus, a także być możliwym do obsłużenia z poziomu klawiatury bądź innego alternatywnego urządzenia wskazującego.

### Przydatne narzędzia do sprawdzania dostępności stron internetowych

1. [www.wave.webaim.org](http://www.wave.webaim.org) – validator umożliwiający łatwe przeanalizowanie pojedynczej strony pod kątem dostępności według standardu WCAG 2.0;
2. Colour Contrast Analyser – program badający kontrast strony internetowej (program dostępny na platformy Windows i macOS);
3. PDF Accessibility Checker – narzędzie pozwalające sprawdzić dostępność plików PDF.



### BARTŁOMIEJ KRAKOWSKI

[bartlomiej.krakowski@interia.eu](mailto:bartlomiej.krakowski@interia.eu)

Web developer i web designer. Zwolennik czystego kodu i wolnego oprogramowania. Osoba zafascynowana możliwościami, które daje nam Internet, a także technologią, która nas otacza. Obcowanie z internetowymi witrynami przestało mu wystarczać, więc zaczął tworzyć je sam.

# Najlepsze od

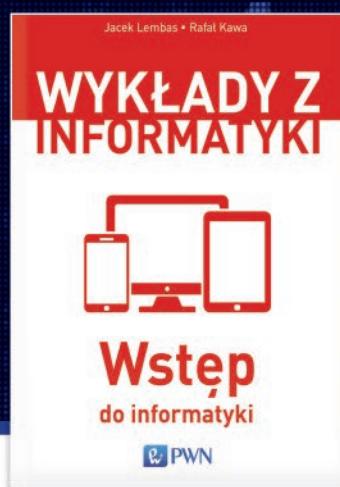
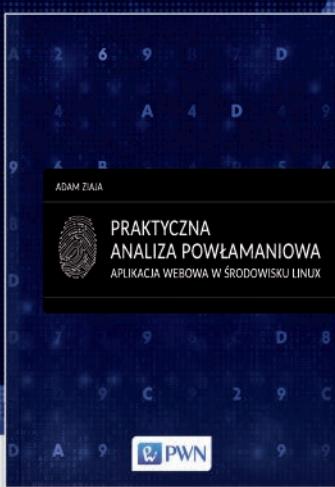


## PATRONI



NOWE  
Wydanie

Polecamy:



KSIAŻKI DOSTĘPNE NA: [WWW.KSIEGARNIA.PWN.PL](http://WWW.KSIEGARNIA.PWN.PL)

Odwiedź nas na:



[IT.PWN.PL](http://IT.PWN.PL)

# Być jak Sherlock Holmes. Przetwarzanie obrazów w biometrii

Sherlock Holmes, bohater literacki występujący w prozie autorstwa Sir Arthur Conan Doyle'a, bez cienia wątpliwości jest jednym z najbardziej fascynujących przykładów na to, jak nauka i umiejętność logicznego myślenia pozwalają na zdobywanie szczytów. Podstawowym celem detektywa-konsultanta było rozwiązywanie zagadek kryminalnych i wskazywanie sprawców zbrodni. W jednym z dzieł szkockiego pisarza [1] Sherlock Holmes udowadnia winę podejrzanej poprzez wykorzystanie dowodu z miejsca przestępstwa, którym były odciski palców. W niniejszym artykule pokażę, że każdy z nas może nabyć umiejętności wspomnianego śledczego i poprzez zastosowanie prostych algorytmów przetwarzających obrazy dokonać analizy odcisku palca. Pamiętajmy bowiem, że „dla wielkiego umysłu nie ma rzeczy małych” [2].

## „KONCEPCJE MUSZĄ BYĆ TAK SZEROKIE, JAK NATURA, JEŻELI MAJĄ JĄ OGARNĄĆ”

W poprzednim artykule dotyczącym przetwarzania obrazów [3] jednym z poruszanych tematów były metody binaryzacji. Należy jednak zwrócić uwagę czytelnika, że były to jedynie metody manualne. W tym kontekście oznacza to, że programista sam wprowadzał pewną wartość progową i na jej podstawie generował finalny zbinaryzowany obraz. Jednakże istnieją pewne metodyki, które nie wymagają takiej ingerencji twórcy (tudzież użytkownika danej implementacji). W przypadku tego artykułu chciałbym omówić trzy metody, które służą do automatycznej detekcji progu binaryzacji i generacji finalnego obrazu. Pierwszą z metodą będzie najbardziej znany algorytm Otsu.

## BINARYZACJA OTSU

Metoda binaryzacji, która zostanie omówiona w niniejszym podrozdziale, otrzymała swoją nazwę na cześć jej twórcy. Tymże był prof. Nobuyuki Otsu. Stworzył on bowiem metodę, która jest bardzo często stosowana i stała się swoistym standardem w algorytmach przetwarzania wstępnego obrazów.

Pierwszym etapem tego algorytmu jest zamiana obrazu ze skali kolorów RGB na odcienie szarości. Metoda Otsu nie narzuca odgórnie żadnego sposobu dokonującego takiej zmiany. W związku z tym istnieją cztery najbardziej powszechnie możliwości. Wszystkie z nich polegają na ustaleniu dokładnie tej samej wartości na wszystkich kanałach (czerwonym, zielonym i niebieskim) każdego piksela. Różnica polega na tym, że może to być wartość pobrana z kanału czerwonego, zielonego, niebieskiego albo wartość uśredniona z tych trzech kanałów. Na Rysunku 1 przedstawione zostały obrazy w skali szarości uzyskane z wykorzystaniem różnych kanałów.

Pytaniem, które może być tutaj postawione, jest kwestia związana z tym, kiedy należy stosować który kanał. Tak naprawdę nie ma na tak postawione pytanie jednoznacznej odpowiedzi. Otóż wiele zależy od tego, w jaki sposób (i z wykorzystaniem jakiej jakości urządzeń) dochodzi do akwizycji danych. Aby uzmysłowić czy-



Rysunek 1. Obraz oryginalny (a) oraz obrazy przekształcone do skali szarości z wykorzystaniem kanału czerwonego (b), niebieskiego (c), zielonego (d) i wartości uśrednionej (e)

telnikowi sposób działania tego prostego przekształcenia, przedstawiam fragment kodu wykonujący zadaną operację w Listingu 1.

### Listing 1. Zamiana obrazu na skalę szarości

```
private static BufferedImage changeToGrayScale(BufferedImage input, ImageColors color) {
    for (int w = 0; w < input.getWidth(); w++) {
        for (int h = 0; h < input.getHeight(); h++) {
            Color pixelColor = new Color(input.getRGB(w, h));
            switch (color) {
                case BLUE:
                    int blue = pixelColor.getBlue();
                    Color cBlue = new Color(blue, blue, blue);
                    input.setRGB(w, h, cBlue.getRGB());
                    break;
                case GREEN:
                    int green = pixelColor.getGreen();
                    Color cGreen = new Color(green, green, green);
                    input.setRGB(w, h, cGreen.getRGB());
                    break;
                case RED:
                    int red = pixelColor.getRed();
                    Color cRed = new Color(red, red, red);
                    input.setRGB(w, h, cRed.getRGB());
                    break;
            }
        }
    }
}
```

# SZUKASZ PRACY MARZEŃ?



## ZRÓB PIERWSZY KROK!

**PRZYJDŹ DO SZKOŁY JĘZYKA ANGIELSKIEGO SPEAK UP**

- 🕒 NAUCZYMY CIĘ ANGIELSKIEGO SZYBKO I SKUTECZNIE
- ▶ STAWIAMY NA TWÓJ ROZWÓJ POPRZEZ INNOWACYJNE METODY NAUCZANIA
- ⌚ GWARANTUJEMY ELASTYCZNOŚĆ I INDYWIDUALNE PODJĘCIE
- 🏆 Z NAMI ZDOBĘDZIESZ CERTYFIKAT JĘZYKOWY



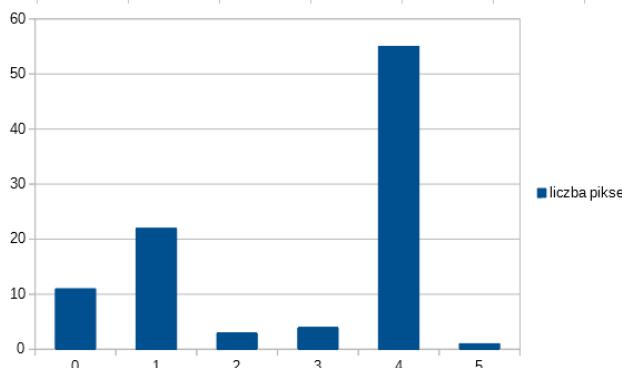
```

        Color cRed = new Color(red, red, red);
        input.setRGB(w, h, cRed.getRGB());
        break;
    case AVERAGE:
        double val = pixelColor.getRed() + pixelColor.
        getBlue() + pixelColor.getGreen();
        val = val / 3;
        int cVal = (int) val;
        Color cAvg = new Color(cVal, cVal, cVal);
        input.setRGB(w, h, cAvg.getRGB());
    default:
        break;
    }
}
return input;
}

```

W tym miejscu należy zauważyć, że pierwszym parametrem jest obraz, który jest w skali kolorów RGB, natomiast drugim elementem wprowadzanym do metody jest informacja o wybranym rodzaju przejścia na skalę szarości. `ImageColors` jest specjalnie przygotowaną w tym celu enumeracją, która może przyjąć jedną z czterech wartości: RED, GREEN, BLUE albo AVERAGE. Następnie w zależności od wybranego typu ustawiana jest odpowiednia wartość na wszystkich kanałach każdego z badanych pikseli.

Należy też nadmienić, że binaryzacja wykonywana metodą Otsu jest również nazywana metodą optymalnego podziału. Mianowicie w kolejnym kroku tego algorytmu wyliczany jest histogram dla skali szarości. Obliczenie histogramu polega na przyporządkowaniu do każdej wartości z zakresu od 0 do 255 liczb pikesli, którą daną wartość posiadają. Na przykład:



Wykres 1. Przykładowy histogram

Zaprezentowany powyżej histogram informuje, że obraz zawiera 11 pikseli o wartości 0, 22 piksele o wartości 1 etc. Operujemy na wartościach skalarnych, ponieważ przejście do skali szarości spowodowało, że trzy kanały występujące w reprezentacji RGB zostały zredukowane do pojedynczego kanalu V oznaczającego szarość.

Kolejnym krokiem w ramach binaryzacji Otsu jest automatyczna kalkulacja progu. Ten etap jest wykonywany dla każdej wartości od 1 do 254. Wartości poniżej aktualnie analizowanej wartości progowej są uznawane za tło, natomiast powyżej za elementy obiektu. Adekwatnym pytaniem, które może zostać w tym miejscu zadane, jest pytanie związane z tym, w jaki sposób wybierany jest próg. Otóż metoda Otsu zakłada, że wybrany próg ma minimalizować wariancję wewnętrzgrupową dla obydwu wyznaczanych zbiorów. Zatem za najlepszy uznawany jest próg, dla którego wartość wariancji wewnętrzgrupowej jest najwyższa. W Listingu 2 przedstawiona została metoda, która służy kalkulacji histogramu, natomiast w Listingu 3 zawarto metody służące wykonaniu obliczeń matematycznych – wariancji oraz wartości średnich.

Listing 2. Obliczanie histogramu dla obrazu w skali szarości

```

private static long[] calculateHistogram(BufferedImage inputImage) {
    long[] histogram = new long[256];
    for (int w = 0; w < inputImage.getWidth(); w++) {
        for (int h = 0; h < inputImage.getHeight(); h++) {
            Color currentPixel = new Color(inputImage.getRGB(w, h));
            histogram[currentPixel.getRed()]++;
        }
    }
    return histogram;
}

```

Listing 3. Matematyczne metody służące do wyznaczenia wariancji wewnętrzklasowych

```

private static double withinClassVarianceCalculator(long[] histogram, int threshold, long pixelNumber) {
    long[] left = new long[threshold];
    System.arraycopy(histogram, 0, left, 0, threshold);
    long[] right = new long[256 - threshold];
    System.arraycopy(histogram, threshold, right, 0, 256 - threshold);

    double weightBackground = calculateWeight(left, pixelNumber);
    double weightForeground = calculateWeight(right, pixelNumber);

    double meanBackground = calculateMeanValue(left, threshold, true);
    double meanForeground = calculateMeanValue(right, threshold, false);

    double varianceBackground = calculateVariance(left, threshold,
    meanBackground, true);
    double varianceForeground = calculateVariance(right, threshold,
    meanForeground, false);

    return weightBackground * varianceBackground + weightForeground *
    varianceForeground;
}

private static double calculateWeight(long[] table, long pixelNumber) {
    double weight = 0;
    for (int i = 0; i < table.length; i++) {
        weight = weight + (double) table[i];
    }
    weight = weight / pixelNumber;
    return weight;
}

private static double calculateMeanValue(long[] table, int threshold, boolean left) {
    double value = 0;
    double occurrences = 0;
    for (int i = 0; i < table.length; i++) {
        if (left) {
            value = value + table[i] * i;
        } else {
            value = value + table[i] * (i + threshold);
        }
        occurrences = occurrences + table[i];
    }
    value = value / occurrences;
    return value;
}

private static double calculateVariance(long[] table, int threshold, double minValue, boolean left) {
    double value = 0;
    double occurrences = 0;
    for (int i = 0; i < table.length; i++) {
        if (left) {
            double pow2 = Math.pow((i - minValue), 2);
            value = value + pow2 * (double) table[i];
        } else {
            double pow2 = Math.pow((i + threshold - minValue), 2);
            value = value + pow2 * (double) table[i];
        }
        occurrences = occurrences + table[i];
    }
    value = value / occurrences;
    return value;
}

```

Ostatnim etapem metody Otsu jest wyznaczenie progu, czyli obliczenie minimalnej wartości wariancji wewnętrzklasowej. Prosty

sposób, w jaki można tego dokonać, jest zaprezentowany w Listingu 4, natomiast w Listingu 5 zawarto informacje, w jaki sposób można dokonać konwersji do obrazu zbinaryzowanego.

#### Listing 4. Obliczenie minimalnej wartości progu binaryzacji

```
private static double calculateThreshold(long[] histogram, long pixelNumber) {
    double minimum = Double.MAX_VALUE;
    double threshold = 0;
    for (int i = 1; i < 255; i++) {
        double value = withinClassVarianceCalculator(histogram, i, pixelNumber);
        if (value < minimum) {
            minimum = value;
            threshold = i;
        }
    }
    return threshold;
}
```

#### Listing 5. Binaryzacja obrazu

```
private static BufferedImage convertToBinary(BufferedImage input, int threshold) {
    for (int w = 0; w < input.getWidth(); w++) {
        for (int h = 0; h < input.getHeight(); h++) {
            Color color = new Color(input.getRGB(w, h));
            if (color.getRed() > threshold) {
                input.setRGB(w, h, Color.WHITE.getRGB());
            } else {
                input.setRGB(w, h, Color.BLACK.getRGB());
            }
        }
    }
    return input;
}
```

W tym miejscu należy również nadmienić, że w metodzie binaryzacyjnej nie ma znaczenia, który z kanałów pobierzemy. Jest to związane z tym, że analizowany przez nas obraz jest już zamieniony do skali szarości, co oznacza, że każdy z pikseli ma dokładnie tą samą wartość ustaloną na każdym z kanałów. Na Rysunku 2 przedstawiony został obraz oryginalny oraz uzyskany z wykorzystaniem metody Otsu.



Rysunek 2. Obraz oryginalny (a) oraz obraz uzyskany po binaryzacji z użyciem metody Otsu (b)

## METODA SELEKCJI ENTROPII

Dobrym źródłem opisującym tę metodę jest strona internetowa [4]. W przypadku obrazów wartość entropii jest obliczana tylko i wyłącznie, gdy są one zaprezentowane w skali szarości. Analogicznie jak w ramach metody poprzedniej, skorzystamy w pierwszym etapie z histogramu. Będzie on obliczany dokładnie w ten sam sposób z wykorzystaniem metody `calculateHistogram(BufferedImage image)`. Kolejnym krokiem tej procedury jest obliczenie prawdopodobieństwa wystąpienia poszczególnych wartości pikseli na naszym obrazie. Obliczamy zatem sumę wszyst-

kich pikseli na obrazie i następnie dla każdej wartości piksela badamy, z jakim prawdopodobieństwem on występuje. Ten parametr obliczamy poprzez odniesienie ilości pikseli danej wartości do wszystkich pikseli. Wyznaczenie prawdopodobieństwa jest przedstawione w ramach następującego wzoru:

$$p(x_i) = \frac{H(x_i)}{\sum_{i=0}^{255} H(x_i)}$$

W ramach powyższego wzoru zmienna  $x_i$  oznacza poziom, dla którego obliczamy wartość prawdopodobieństwa. Ten parametr może przyjmować wartość od 0 do 255.  $H(x_i)$  to z kolei ilość wystąpień pikseli z poziomu  $x_i$ . Należy również zauważyć, że mianownik w powyższym wzorze oznacza sumę wszystkich pikseli obrazu, czyli wysokość obrazu, która została przemnożona przez jego szerokość.

Po wyznaczeniu prawdopodobieństwa dla każdego z poziomów histogramu kolejnym etapem algorytmu jest obliczenie globalnej entropii dla całego obrazu. Entropia to średnia liczba bitów, która jest niezbędna do zakodowania pojedynczego piksela (przy logarytmie o podstawie 2), jak również jest to miara różnorodności zbioru. W tym przypadku skorzystamy z uprzednio przygotowanych prawdopodobieństw dla każdego z poziomów histogramu. Wzór, z którego w tym przypadku należy skorzystać, jest przedstawiony poniżej:

$$\text{entropy} = - \sum_{i=1}^N p(x_i) \cdot \log(p(x_i))$$

Po obliczeniu wartości globalnej entropii należy na jej podstawie wyznaczyć wartość progową, która pozwoli na zbinaryzowanie obrazu. Aby wykonać ten element w sposób prawidłowy, wyznacza się wartość entropii dla każdego z kolejnych poziomów

W momencie, w którym obliczona aktualnie wartość tego parametru jest mniejsza niż połowa entropii globalnej, kończymy wykonanie algorytmu. Wtedy, jako próg binaryzacji, przyjmujemy ostatnią wartość, dla której obliczona entropia wynosiła więcej (lub była równa) 0,5 entropii globalnej. W Listingu 6 przedstawiam kod programu, z użyciem którego taka binaryzacja może zostać wykonana. Potwierdzeniem prawidłowego działania programu jest Rysunek 3, na którym przedstawiono oryginalny obrazek i obraz zbinaryzowany.

#### Listing 6. Binaryzacja obrazu z wykorzystaniem metody selekcji entropii

```
public BufferedImage binarizationEntropy(BufferedImage input){
    BufferedImage imageWorkedOn = deepCopy(input);
    long[] histogram = calculateHistogram(imageWorkedOn);
    long suma = 0;

    for(int i = 0; i < 256; i++) {
        suma = suma + histogram[i];
    }

    double[] probability = new double[256];
    for(int i = 0; i < 256; i++) {
        probability[i] = ((double) histogram[i]) / (double) suma;
    }

    double entropy = 0;
    for(int i = 0; i < 256; i++) {
        if(probability[i] != 0) {
            entropy = entropy - probability[i] * Math.log(probability[i]);
        }
    }
}
```

```

int threshold = 0;
double nextEntropy = 0;
for(int i = 0; i < 256; i++){
    if(probability[i] != 0) {
        nextEntropy = nextEntropy - probability[i] * Math.
        log(probability[i]);
    }
    if(nextEntropy >= (entropy / 2)) {
        threshold = i;
        break;
    }
}
return convertToBinary(imageWorkedOn, threshold);
}

```



(a)



(b)

Rysunek 3. Obraz oryginalny (a) oraz obraz po wykonanej binaryzacji z użyciem metody selekcji entropii (b)

## METODA MINIMUM ERROR

Jako ostatnia metoda binaryzacyjna w ramach niniejszego artykułu przedstawiony zostanie algorytm minimalnego błędu. Analogicznie jak w przypadku dwóch poprzednio przedstawionych algorytmów, tak i tutaj brany jest pod uwagę histogram obrazu. Powoduje to, że ponownie pierwszym krokiem przy przetwarzaniu obrazu jest zmiana do skali szarości. Po przygotowaniu histogramu następnym krokiem jest automatyczna (tj. na podstawie informacji znajdujących się w ramach histogramu) detekcja progu binaryzacyjnego. Otóż metoda błędu minimalnego zakłada, że histogram (skali szarości) stanowi estymację funkcji gęstości prawdopodobieństwa wystąpienia każdego z poziomów szarości.

Metoda błędu minimalnego polega na założeniu, że próg binaryzacyjny dzieli histogram na część tła i część obiektu, przy czym obydwie części mają rozkład normalny. Wtedy to szacuje się przybliżenie histogramu w postaci sumy dwóch rozkładów normalnych. Za najlepszy uznawany jest próg, dla którego odległość przybliżenia histogramu od jego postaci oryginalnej jest najmniejsza. Przybliżenie histogramu obliczamy z użyciem poniższego wzoru:

$$p(g) = \frac{1}{\sigma_1 \sqrt{2\pi}} \cdot e^{\frac{-(g-\mu_1)^2}{2\sigma_1^2}} + \frac{1}{\sigma_2 \sqrt{2\pi}} \cdot e^{\frac{-(g-\mu_2)^2}{2\sigma_2^2}}$$

W powyższym wzorze  $\sigma_1$  oznacza wartość wariancji rozkładu normalnego dla pikseli przydzielonych do obiektu na obrazie, natomiast  $\sigma_2$  jest wartością analogiczną dla pikseli tła. Podobnie rzeczą się tyczy wartości  $\mu_1, \mu_2$ , które stanowią wartości średnie dla pikseli obiektu i pikseli tła. Z kolei wartość  $g$  to wartość aktualnie dobranego progu.

Kod programu, wykonujący obliczenie progu binaryzacji z wykorzystaniem metody błędu minimalnego, jest przedstawiony w Listingu 7, natomiast wynik wykonania tej operacji przedstawiono na Rysunku 4.

Listing 7. Binaryzacja obrazu z wykorzystaniem metody błędu minimalnego

```

public BufferedImage binarizationMinimum()
{
    BufferedImage imageWorkedOn=deepCopy(this.bufferedImage);
    long[] histogram = calculateHistogram(imageWorkedOn);
    double[] pValues = new double[254];
    double[] indexes = new double[254];

    int parametr=0;
    for(int i = 1; i < 255; ++i){
        double l1 = 0;
        double m1 = 0;
        for(int j = 0; j < i; ++j){
            l1 = l1 + j * histogram[j];
            m1 = m1 + histogram[j];
        }
        double l2 = 0;
        double m2 = 0;
        for(int j = i; j < 256; ++j){
            l2 = l2 + j * histogram[j];
            m2 = m2 + histogram[j];
        }

        double m1 = l1/m1;
        double m2 = l2/m2;

        double sigma1 = 0;
        for(int j = 0; j < i; ++j) {
            sigma1 = sigma1 + Math.pow((histogram[j] - m1), 2);
        }

        sigma1 = sigma1 / i;
        sigma1 = Math.sqrt(sigma1);

        double sigma2 = 0;
        for(int j = i; j < 256; ++j) {
            sigma2 = sigma2 + Math.pow((histogram[j] - m2), 2);
        }

        sigma2 = sigma2 / (255 - i + 1);
        sigma2 = Math.sqrt(sigma2);

        double v1=0;
        double powV1 = Math.pow(((double) i - m1), 2);
        double qV1 = Math.pow(sigma1, 2);
        qV1 = 2 * qV1;
        powV1 = powV1 / qV1;
        powV1 = -1 * powV1;

        v1 = Math.pow(Math.E, powV1);
        v1 = v1 * Math.pow((sigma1 * Math.sqrt(2 * Math.PI)), -1);

        double v2=0;
        double powV2 = Math.pow(((double) i - m2), 2);
        double qV2 = Math.pow(sigma2, 2);
        qV2 = qV2 * 2;
        powV2 = powV2 / qV2;
        powV2 = -1 * powV2;

        v2 = Math.pow(Math.E, powV2);
        v2 = v2 * Math.pow((sigma2 * Math.sqrt(2 * Math.PI)), -1);

        double result = v1 + v2;

        if(v1 != 0 && v2 != 0) {
            pValues[parametr] = wynik;
            indexes[parametr] = i;
            ++parametr;
        }
    }

    double min = 0;
    double minIndex = 0;
    for(int i = 0; i < 254; ++i){
        if(i == 0) {
            min = functionpValues[0];
        }
        if(functionpValues[i] <= min){
            if(indexes[i] != 0) {
                min = functionpValues[i];
                minIndex = i+1;
            }
        }
    }

    return convertToBinary(imageWorkedOn, minIndex);
}

```



(a)



(b)

Rysunek 4. Obraz oryginalny (a) oraz obraz po wykonaniu binaryzacji z użyciem metody kąta minimalnego

Jako podsumowanie tej części artykułu, która jest związana z binarizacją, chciałbym zwrócić uwagę na to, że przedstawione przeze mnie algorytmy były algorytmami globalnymi. To znaczy, że próg binaryzacyjny był obliczany na mocy danych znajdujących się na całym obrazie. Z tego względu chciałbym zwrócić uwagę czytelników także na metody, które obliczają wartości poszczególnych pikseli w oparciu o obliczenia lokalne. Są to również interesujące metody, które pozwalają na efektywne przygotowanie obrazu zbinaryzowanego.

## „PO WYELIMINOWANIU RZECZY NIEMOŻLIWYCH TO, CO POZOSTANIE, CHOCIAZBY BYŁO NAJBARDZIEJ NIEPRAWDOPODOBNE, MUSI BYĆ PRAWĄ”

Kolejną niezwykle ważną metodyką, która jest stosowana powszechnie w analizie i przetwarzaniu obrazów, jest ścienianie. Należy stwierdzić, że istnieje szerokie spektrum algorytmów, które wykonyują wspomnianą operację. Na czym ona polega? Opowiedź, choć może wydać się zaskakująca, jest banalnie prosta. Otóż ścienianie to procedura, która służy do zmniejszenia szerokości linii na obrazie do jednego piksela. Kolejnym pytaniem, które samo się nasuwa, jest to o zasadność istnienia takich metod. Po czym one są? Odnieśmy się w tym miejscu do biometrii, która sama podsuwa nam odpowiedź. Zwróciły uwagę na odcisk palca. W momencie, w którym jest on nieprzetworzony, wszystkie linie są szersze aniżeli pojedynczy piksel. W takim przypadku ze względu na znaczną nadmiarowość informacji nie istnieje praktycznie możliwość dokonania detekcji minucji (punktów szczególnych odcisku palca, które umożliwiają jego opisanie). Dopiero zredukowanie odcisku palca do postaci szkieletowej pozwala na prawidłowe wykonanie takiej procedury. Możemy zatem stwierdzić, że podstawowym celem metod ścieniających jest redukcja nadmiarowej informacji z obrazu. W ramach niniejszego artykułu zaprezentowany zostanie algorytm KMM [5].

## KMM

Bardzo interesującą metodą, którą często wykorzystuję w swoich badaniach naukowych, jest algorytm, który swoje korzenie ma na Wydziale Informatyki Politechniki Białostockiej. Otóż został on przygotowany jako wynik badań przeprowadzonych przez prof. Khalida Saeed'a wraz ze swoimi asystentami – dr inż. Markiem Tabędzkim i dr inż. Mariuszem Rybnikiem. Algorytm jest prosty w zrozumieniu i implementacji, a także zapewnia wyniki bardziej dobrej jakości.

Odwrotnie niż w przypadku algorytmów binaryzacji najpierw przedstawię kod programu (Listingi 8 - 10), który służy do prawa-

dłowego wykonania procedury opisanej w artykule [5], a następnie omówię poszczególne kroki tego algorytmu.

**Listing 8. Główna metoda wykonująca ścienianie obrazu zgodnie z semantyką algorytmu KMM**

```
public double[][][] doThinning(double[][][] imageTable) throws
Exception{
    double[][] finalImage = null;

    imageTable = secondThinningStep(imageTable);
    imageTable = thirdThinningStep(imageTable);
    imageTable = fourthThinningStep(imageTable);
    imageTable = deleteAllFours(imageTable);

    double[] deleteList = prepareDeleteList();

    double[] weighes = {128, 64, 32, 1, 0, 16, 2, 4, 8};

    finalImage = tableDeepCopy(imageTable);
    for(int k = 2; k < 4; ++k){
        for(int w = 1; w < this.imageInput.getWidth() - 1; ++w){
            for(int h = 1; h < this.imageInput.getHeight() - 1; ++h){
                if(finalImage[w][h] == k){
                    int pos = 0;
                    double pixelWeight = 0;
                    for(int iw = -1; iw < 2; ++iw){
                        for(int ih = -1; ih < 2; ++ih){
                            double exists = (finalImage[w + iw][h + ih] != 0)
                                ? 1 : 0;
                            pixelWeight = pixelWeight + weighes[pos] * exists;
                            ++pos;
                        }
                    }
                    Boolean decision = false;
                    for(int i = 0; i < deleteList.length; ++i){
                        if(deleteList[i] == pixelWeight){
                            decision = true;
                            break;
                        } else if(deleteList[i] > pixelWeight){
                            break;
                        }
                    }
                    finalImage[w][h] = (decision == true) ? 0 : 1;
                }
            }
        }
    }
    return finalImage;
}
```

**Listing 9. Metody pozwalające na wykonanie poszczególnych kroków algorytmu**

```
private double[][][] secondThinningStep(double[][][] tableInput){
    double[][][] tableOutput = tableDeepCopy(tableInput);
    for(int w = 1; w < this.imageInput.getWidth() - 1; ++w){
        for(int h = 1; h < this.imageInput.getHeight() - 1; ++h){
            if(tableInput[w][h] == 1){
                tableOutput[w][h] = (tableInput[w][h - 1] == 0 ||
                    tableInput[w - 1][h] == 0 || tableInput[w][h + 1] == 0 ||
                    tableInput[w + 1][h] == 0) ? 2 : tableOutput[w][h];
            }
        }
    }
    return tableOutput;
}

private double[][][] thirdThinningStep(double[][][] tableInput){
    double[][][] tableOutput = tableDeepCopy(tableInput);
    for(int w = 1; w < this.imageInput.getWidth() - 1; ++w){
        for(int h = 1; h < this.imageInput.getHeight() - 1; ++h){
            if(tableInput[w][h] == 1){
                tableOutput[w][h] = (tableInput[w - 1][h - 1] == 0 ||
                    tableInput[w + 1][h - 1] == 0 || tableInput[w - 1][h + 1] == 0 ||
                    tableInput[w + 1][h + 1] == 0) ? 3 : tableOutput[w][h];
            }
        }
    }
    return tableOutput;
}

private double[][][] fourthThinningStep(double[][][] tableInput){
    double[][][] tableOutput = tableDeepCopy(tableInput);
    double[] weighes = {128, 64, 32, 1, 0, 16, 2, 4, 8};
    double[] stickyNeighbours = {3, 6, 12, 24, 48, 96, 192, 129,
```

```

7, 14, 28, 56, 112, 224, 193, 131,
15, 30, 60, 120, 240, 225, 195, 135};
for(int w = 1; w < this.imageInput.getWidth() - 1; ++w){
    for(int h = 1; h < this.imageInput.getHeight() - 1; ++h){
        double count = 0;
        if(tableInput[w][h] == 2){
            int pos = 0;
            for(int iw = -1; iw < 2; ++iw){
                for(int ih = -1; ih < 2; ++ih){
                    double exists = (tableInput[w + iw][h + ih] == 0) ?
                        0 : 1;
                    count = count + exists * weighs[pos];
                    ++pos;
                }
            }
            Boolean decision = false;
            for(int i = 0; i < stickyNeighbours.length; ++i){
                if(count == stickyNeighbours[i]){
                    decision = true;
                    break;
                }
            }
            tableOutput[w][h] = (decision == true) ? 4 :
            tableOutput[w][h];
        }
    }
}
return tableOutput;
}

private double[][] deleteAllFours(double[][] tableInput){
    for(int w = 0; w < this.imageInput.getWidth(); ++w){
        for(int h = 0; h < this.imageInput.getHeight(); ++h){
            tableInput[w][h] = (tableInput[w][h] == 4) ? 0 :
            tableInput[w][h];
        }
    }
    return tableInput;
}

```

**Listing 10. Wszystkie wartości, które powinny zostać usunięte**

```

private double[] prepareDeleteList(){
    double[] values = {3, 5, 7, 12, 13, 14, 15,
        20, 21, 22, 23, 28, 29, 30,
        31, 48, 52, 53, 54, 55, 56,
        60, 61, 62, 63, 65, 67, 69,
        71, 77, 79, 80, 81, 83, 84,
        85, 86, 87, 88, 89, 91, 92,
        93, 94, 95, 97, 99, 101, 103,
        109, 111, 112, 113, 115, 116,
        117, 118, 119, 120, 121, 123,
        124, 125, 126, 127, 131, 133,
        135, 141, 143, 149, 151, 157,
        159, 181, 183, 189, 191, 192,
        193, 195, 197, 199, 205, 207,
        208, 209, 211, 212, 213, 214,
        215, 216, 217, 219, 220, 221,
        222, 223, 224, 225, 227, 229,
        231, 237, 239, 240, 241, 243,
        244, 245, 246, 247, 248, 249,
        251, 252, 253, 254, 255};

    return values;
}

```

Pierwszym krokiem wspomnianej metody jest oznaczenie wszystkich czarnych pikseli poprzez wartość 1 (oznacza to, że wejściowy obraz musi być uprzednio zbinaryzowany). Kolejnym etapem jest zbadanie tego, które wartości 1 stykają się z wartościami zerowymi (taką wartość przyjmują piksele w kolorze białym). Po wykryciu tych pikseli zmienia się im oznaczenie do wartości równej 2. Natomiast piksele 1-ki, które znajdują się w narożnikach, przyjmują wartość 3. Oczywiście badanie poszczególnych pikseli odbywa się poprzez nałożenie maski na obraz i analizę sąsiedztwa piksela centralnego. W tym przypadku jednak brana jest pod uwagę wartość piksela centralnego.

W ramach artykułu w tym miejscu wskazuje się, że dla prostych obrazów algorytm może zostać już zakończony. Dlaczego tak się dzieje? Otóż po usunięciu wszystkich wartości równych 2 i 3 po-

zostaną już tylko krawędzie obrazu. Jednakże gdy przetwarzany obraz jest bardziej skomplikowany, należy kontynuować oznaczanie pikseli. 4-kami zostaną oznaczone wszystkie te piksele, które mają 2, 3 lub 4 sąsiadów, z którymi bezpośrednio się stykają. W tym momencie następuje ostatni etap przetwarzania, mianowicie jest to etap obliczania wartości wag poszczególnych pikseli. Polega on na nałożeniu maski na obraz i na podstawie wartości pikseli sąsiadujących obliczeniu wagi piksela centralnego. Prawidłowy kształt maski został przedstawiony na Rysunku 5.

128	1	2
64	x	4
32	16	8

Rysunek 5. Wagî w masce nakładanej na obraz

Po obliczeniu wagi dla każdego z pikseli należy usunąć te, które znajdują się w ramach tablicy usuńć (przedstawiona została w ramach Listingu 10). Należy w tym miejscu wskazać, że tablica usuńć została przedstawiona w ramach artykułu [5] (a jej rozwinięcie i zwiększenie wydajności metody zostało przedstawione w artykule [6]). Ostatnim krokiem tego algorytmu jest wielokrotne wykonywanie zaprezentowanych wcześniej etapów w celu osiągnięcia zbieżności, czyli uzyskania obrazu, na którym wszystkie krawędzie będą miały szerokość jednego piksela. Na Rysunku 6 przedstawiono przykładowy obraz oraz wynik procedury ścieniania.



Rysunek 6. Obraz oryginalny (a) oraz obraz po ścienianiu (b)

## „PAMIĘTAJMY O LOGICE. TAM GDZIE JĘJ BRĄK, NALEŻY DOSZUKIWAĆ SIĘ PODSTĘPU”

Kolejnym interesującym przekształceniem, które w przetwarzaniu obrazów ma bardzo dużo zastosowań, jest Transformata Fouriera. Joseph Fourier (1768 – 1830) przedstawił sposób, z użyciem którego następuje rozkład szerokiej klasy funkcji okresowych (możemy je traktować na równi z sygnałami) na składowe harmoniczne. Taką reprezentację sygnału otrzymuje się poprzez odpowiedni dobór ważonej sumy funkcji harmonicznych wyróżniających się różnymi częstotliwościami. Wzór Transformaty Fouriera jest następujący:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega t} dt$$

W powyższym wzorze  $t$  jest oznaczeniem czasu, natomiast  $\omega$  oznacza częstotliwość. Adekwatnym pytaniem jest, dlaczego w ogóle powinniśmy stosować to przekształcenie. Otóż pozwala ono na przeniesienie sygnału (w naszym przypadku obrazu) z domeną czasu (przestrzennej) do domeny częstotliwości. Ta procedura umożliwia z kolei analizę częstotliwościową sygnału, a nierzadko zawarte w niej informacje pozwolą na uzyskanie cech obrazu, które z kolei

są niezauważalne w dziedzinie przestrzennej. Zaprezentowany powyżej wzór jest przeznaczony dla jednowymiarowej Transformaty Fouriera. Aby móc ją zastosować na obrazie, niezbędna jest Transformata dwuwymiarowa, którą to przedstawiam we wzorze:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \cdot e^{-2\pi j(ux+vy)} dx dy$$

Zwróćmy uwagę na to jakim rodzajem sygnału jest obraz. Czy jest to sygnał ciągły (analogowy) czy też dyskretny? Obrazu nie jesteśmy w stanie przedstawić z wykorzystaniem pojedynczej funkcji natomiast składa się on z wielu próbek, którymi są poszczególne piksele, w związku z tym możemy jednoznacznie stwierdzić, że jest to sygnał dyskretny. Powoduje to, że użycie przedstawionych dotychczas wzorów jest niemożliwe. W tym miejscu należy przedstawić zatem wzory dla Dyskretnej Transformaty Fouriera (DFT). Jaka jest podstawowa różnica pomiędzy Transformatą Fouriera (FT) a DFT? Otóż pierwsza z nich jest przeznaczona dla sygnałów ciągłych natomiast druga z wymienionych jest wykorzystywana w sygnałach dyskretnych, opisywanych przez zbiór próbek. Wzory dla jedno- i dwuwymiarowej DFT przedstawiam poniżej:

$$O_k = \sum_{n=1}^N a_n \cdot w_N^{-kn}$$

$$w_N = e^{i \frac{2\pi}{N}}$$

W ramach powyższego wzoru oznaczenia są następujące:  $i$  – jest to jednostka urojona,  $k$  – numer składowej harmonicznej ( $1 \leq k \leq N$ ),  $n$  – numer próbki sygnału,  $a_n$  – wartość  $n$ -tej próbki sygnału,  $N$  – liczba próbek.

Ponownie należy zauważyć, że w przypadku obrazu powinniśmy skorzystać z dwuwymiarowej DFT. Wzór, który posłuży do obliczeń, jest prezentowany poniżej:

$$F(m, n) = \sum_{x=1}^N \sum_{y=1}^M U(x, y) \cdot w_N^{ny} \cdot w_M^{mx}$$

Oznaczenia w przypadku dwuwymiarowej DFT są analogiczne jak w przypadku transformaty jednowymiarowej.

Istnieje wiele zróżnicowanych bibliotek, które umożliwiają wykonanie prostej implementacji Transformaty w celu użycia jej na obrazie. Jedną z nich jest OpenCV i Numpy. W tym celu skorzystamy z języka Python, który w bardzo prosty sposób pozwoli nam na wykonanie odpowiedniego przekształcenia. Kod programu wykonujący Szybką Transformatę Fouriera na obrazie przedstawiam w Listingu 11. Należy również w tym miejscu nadmienić, że wspomniana transformata wykonuje Dyskretną Transformatę Fouriera (DFT), która opisałem powyżej.

**Listing 11. Kod w języku Python wykonujący Transformatę Fouriera [7]**

```
import numpy as np
import cv2
img = cv2.imread('lena.png', 0)
```

reklama



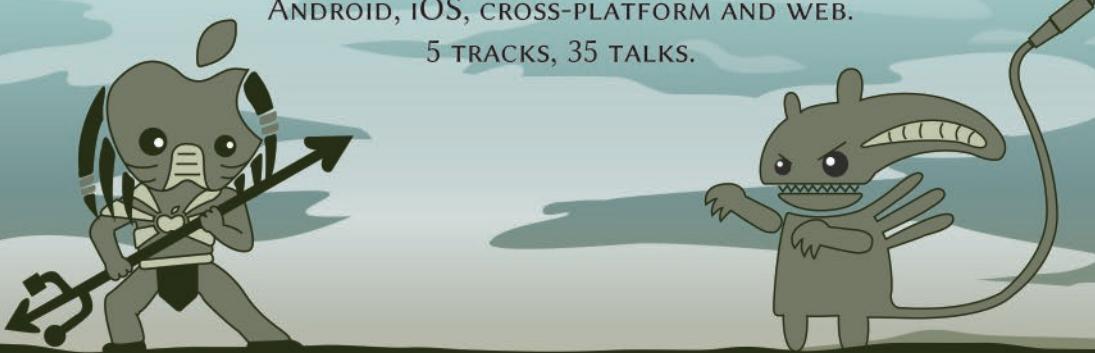
# MOBILIZATION<sup>7</sup>

IN THE CITY OF BOAT AND WITHOUT THE SEA...

WE ENCOURAGE YOU TO TAKE PART IN THE 7TH EDITION OF **MOBILIZATION** CONFERENCE  
21 OCTOBER 2017 IN ŁÓDŹ, POLAND. THE MAIN SCOPE IS THE DEVELOPMENT OF MOBILE SOLUTIONS.

ANDROID, IOS, CROSS-PLATFORM AND WEB.

5 TRACKS, 35 TALKS.



Mobilization is supported by sponsors:



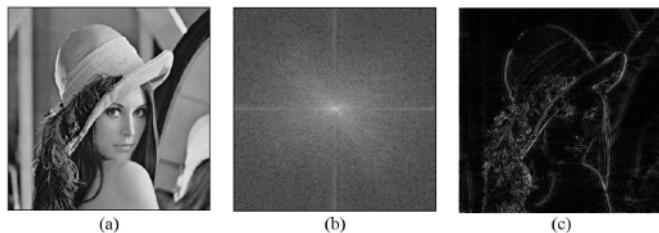
```

fourier = np.fft.fft2(img)
fourierShift = np.fft.fftshift(fourier)
magnitude = 20 * np.log(np.abs(fourierShift))

rows, cols = img.shape
crow, ccol = rows/2, cols/2
fourierShift[crow - 30:crow + 30, ccol - 30:ccol + 30] = 0
f_ishift = np.fft.ifftshift(fourierShift)
img_back = np.fft.ifft2(f_ishift)
img_back = np.abs(img_back)

```

W ramach kodu przedstawionego w Listingu 11 wykonywane są trzy operacje. Pierwsza z nich polega na wczytaniu obrazu (w tym przypadku jest to klasyczny obraz – czyli Lena), a następnie na wykonaniu na nim przekształcenia do domeny częstotliwościowej (Transformata Fouriera). Wynikiem tego są dane przypisane do zmiennej. Kolejna część z kolei pozwala nam na wykonanie przekształcenia HPF (High Pass Filtering), czyli filtracji górnoprzepustowej. Zwróćmy uwagę na to, że ten rodzaj filtrów działa jak detektor krawędzi. W ostatnim kroku drugiej części kodu możemy zaobserwować Odwrotną Transformatę Fouriera (IFT), która pozwala na wykonanie rekonstrukcji obrazu. Na Rysunku 7 przedstawiono uzyskane rezultaty po wykonaniu każdej z zadanego operacji.



Rysunek 7. Obraz oryginalny (a), obraz w domenie częstotliwościowej (b), zrekonstruowany obraz po wykonaniu operacji HPF (c)

Oczywiście przedstawione w tym podrozdziale zastosowania Transformaty Fouriera nie są jedynymi, które mogą zostać uwidocznione w ramach operacji na obrazach. Innymi przykładami, które polecam czytelnikom do samodzielnego przetestowania i sprawdzenia, są m.in. wygładzanie i wyostrzanie obrazu, wyznaczanie kierunku struktur w obrazie czy obserwacja okresowości w ramach obrazu.

## „NA ŚWIĘCIE JEST DUŻO RZECZY OCZYWIISTYCH, NA KTÓRE NIE ZWRACA SIĘ UWAGI”

Na wstępnie wspomniałem o tym, że Sherlock Holmes potrafił wskazać przestępca na podstawie odcisków palca pozostawionych na miejscu przestępstwa. W dzisiejszych czasach najbardziej popularne i najbardziej efektywne rozwiązania oparte są o detekcję minucji. Czym są minucje? Są to pewne charakterystyczne punkty na odcisku palca, które przyjmują zróżnicowane kształty. Do takich kształtów zaliczamy chociażby zakończenie linii czy jej rozwidlenie.

Do prostej detekcji możemy wykorzystać algorytm CN (Crossing Number). Jest to rozwiązanie, które jest bardzo często wykorzystywane i daje zadowalające efekty. Otóż w ramach tego algorytmu wykorzystuje się maskę 3x3 i bierze się pod uwagę ośmiu najbliższych sąsiadów piksela centralnego. Następnie dokonuje się obliczenia wartości CN zgodnie ze wzorem zamieszczonym poniżej.

$$CN = \frac{1}{2} \cdot \sum_{i=1}^8 |P_i - P_{i-1}|$$

W ramach powyższego wzoru, wartości  $P_i$  oraz  $P_{i-1}$  oznaczają wartości pikseli sąsiadujących z pikselem analizowanym. Należy zauważyć, że mogą one przyjmować tylko jedną z dwóch wartości: 0 lub 1. Jest to powodowane tym, że obraz wejściowy musi być obrazem zbinarzowanym.

Po obliczeniu wartości CN dla aktualnie analizowanego piksela należy przejść do jego klasyfikacji. Taka operacja jest wykonywana zgodnie z Tabelą 1.

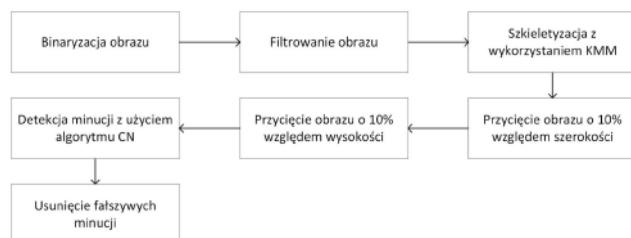
Wartość CN	Rodzaj minucji
0	Brak minucji – piksel należy do tła
1	Zakończenie krawędzi
2	Brak minucji – fragment krawędzi
3	Rozwidlenie krawędzi

Tabela 1. Wartości CN a rodzaje minucji

W ten prosty sposób jesteśmy w stanie dokonać detekcji dwóch podstawowych rodzajów minucji. Umożliwia to stworzenie wektora cech, z użyciem którego będziemy dokonywać porównania odcisków palca.

## „KAŻDY PROBLEM STAJE SIĘ DZIECINNIE PROSTY, GDY ZOSTANIE OBJAŚNIONY”

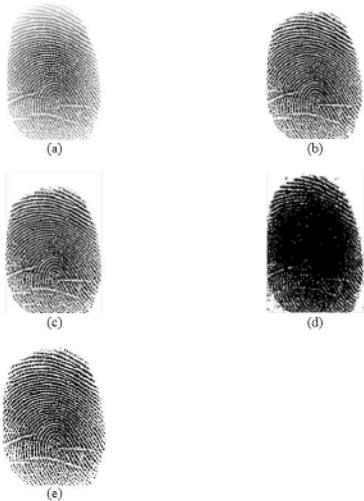
Ostatnim akordem mojego artykułu będzie przedstawienie algorytmu, napisanego przeze mnie i opublikowanego w [8], służącego do przetwarzania odcisku palca w celu zwiększenia jego czytelności oraz uwypuklenia minucji. W ramach tej pracy przedstawię jedynie sposób przetwarzania odcisku palca aż do wykrycia minucji. Wcześniej jednak chciałbym przedstawić Rysunek 8, na którym zaprezentowano przebieg danego algorytmu.



Rysunek 8. Przebieg algorytmu przetwarzania odcisku palca

Pierwszym etapem mojego algorytmu jest binaryzacja obrazu. Otóż ten krok pozwala na uzyskanie znacząco lepszej jakości obrazu, jak również umożliwia usunięcie źle widocznych fragmentów odcisku, które nie zostały do końca dobrze pobrane w trakcie akwizycji danych ze skanera odcisków palca. W tym kontekście przetestowałem kilka algorytmów binaryzacji: binaryzację manualną (próg równy 180), selekcję entropii, metodę błędu minimalnego oraz metodę Otsu. Wyniki tego porównania zaprezentowano na Rysunku 9.

W tym przypadku do dalszej obróbki został wybrany rezultat osiągnięty z użyciem metody manualnej. Jest on bowiem najbardziej informatywny, jak również pozwala na uniknięcie nieadekwatnej reprezentacji odcisku palca – bowiem żadne linie odcisku nie są ze sobą bezpośrednio połączone, jak również każda z nich jest dobrze uwidoczniona. Kolejnym etapem jest zastosowanie filtra medianowego na obrazie, co pozwala na usunięcie drobnych elementów (można je klasyfikować jako swego rodzaju szum na obrazie), a następnie wykonanie procesu szkieletyzacji na obrazie. Wynik

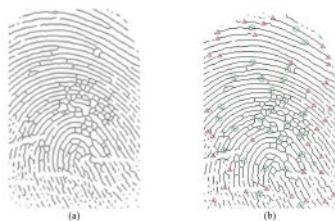


Rysunek 9. Obraz oryginalny (a), obraz zbinaryzowany z wykorzystaniem: metody manualnej (b), metody selekcji entropii (c), metody błędu minimalnego (d) oraz algorytmu Otsu (e)



Rysunek 10. Obraz zbinaryzowany (a), obraz po zastosowaniu filtra medianowego (b), obraz po wykonaniu procedury szkieletyzacji w oparciu o algorytm KMM (c)

wykonania obydwu tych etapów jest przedstawiony na Rysunku 10. W kolejnym etapie następuje przycięcie obrazu o 10% względem wysokości oraz względem szerokości. Ten element został wykonany ze względu na obserwacje, jakie zostały poczynione podczas badań. Otóż w celu prawidłowego rozpoznawania człowieka właśnie w tym regionie odcisku palca znajdują się najbardziej wartościowe informacje. Stąd właśnie została podjęta decyzja o usunięciu nadmiarowych informacji. W kolejnym etapie wykonałem detekcję minucji poprzez algorytm CN, natomiast zwrócił on informację o bardzo dużej liczbie minucji w ramach badanego odcisku palca. Z tego względu należało wziąć pod uwagę, że część z nich jest fałszywymi minucjami i należało je usunąć z obrazu. Dlatego też zastosowałem metodę opisaną w [9], która polegała na detekcji odległości pomiędzy aktualnie analizowaną minucją a już zaakceptowanymi minucjami. Jeżeli taka odległość była mniejsza aniżeli założona wartość progowa, wtedy analizowana minuczka była uznawana za fałszywą i nie była uwzględniana w postaci finalnej odcisku palca. Obrazy uzyskane po przycięciu obrazu oraz po wykryciu minucji i usunięciu fałszywych elementów są zaprezentowane na Rysunku 11.



Rysunek 11. Odcisk palca po przycięciu obrazu (a) oraz wykryte minuczki na odcisku palca (b)

W tym przypadku rozwidlenia zostały oznaczone zielonymi kółkami, natomiast zakończenia czerwonymi trójkątami.

Właśnie poprzez zastosowanie tak prostego algorytmu każdy z nas jest w stanie dokonać analizy odcisku palca i wykrycia minucji.

## PODSUMOWANIE

Podstawowym celem niniejszego artykułu było zaprezentowanie kilku algorytmów przetwarzania obrazów, które mogą być użyteczne dla zróżnicowanych próbek – także tych biometrycznych. Oczywiście należy zauważyć, że istnieje wiele algorytmów, które dla różnych obrazów mogą dawać lepsze rezultaty, a dla innych gorsze. Dlatego nie można jednoznacznie stwierdzić, że wybrana metoda, która na przykład przynosi bardzo dobre rezultaty dla obrazów odcisków palca, będzie uniwersalna i dla każdego rodzaju obrazów zawsze zapewni bardzo dobre wyniki. Jakie cele stawia przed nami analiza i przetwarzanie obrazów? Otóż jest to dążenie do doskonałości. Dążenie do wytyżonej pracy nad jeszcze lepszymi metodami, nad jeszcze bardziej dokładnymi algorytmami. Przedstawiony w ramach niniejszego artykułu algorytm daje przyzwoite wyniki, jednak pytanie jest następujące: jak zachowałby się on dla obrazów, które mają znaczco gorszą jakość? Czy przetwarzanie wstępne jest wystarczające? Otóż na te pytania należałoby sobie odpowiedzieć poprzez przetestowanie tego rozwiązania dla takich obrazów. Następnym krokiem powinna być analiza tego, co jeszcze można poprawić, aby ten algorytm działał dla różnych rodzajów obrazów – na przykład także dla obrazów medycznych. Niech za podsumowanie mojego artykułu posłuży kolejny cytat z Sherlocka Holmesa – „Im jakiś szczegół jest bardziej błahy i śmieszny, tym bardziej zasługuje na dokładne zbadanie, a to samo wydarzenie, które, na pozór, tylko komplikuje sprawę, uważnie rozważone i umiejętnie wykorzystane, najprawdopodobniej posłuży do jej wyjaśnienia.”

## Bibliografia

- [1] Sir Arthur Conan Doyle – „Znak Czterech”, Wydawnictwo Olesiejuk, 2015
- [2] Sir Arthur Conan Doyle – „Studium w szkarłacie”, Wydawnictwo Olesiejuk, 2015
- [3] M. Szymkowski – „Diabeł tkwi w szczegółach”, „Programista”, nr 6/2017.
- [4] <https://goo.gl/b9ra6F> (Dostęp 1.09.2017)
- [5] K. Saeed, M. Rybník, M. Tabędzki – „Implementation and Advanced Results on the Non-interrupted Skeletonization Algorithm”, Computer Analysis of Images And Patterns, Volume 2124 of the series Lecture Notes in Computer Science, p. 601 – 609
- [6] K. Saeed, M. Tabędzki, M. Rybník, M. Adamski – „K3M: A Universal Algorithm for Image Skeletonization and a Review of Thinning Techniques”, International Journal of Applied Mathematics and Computer Science, Vol. 20, No. 2, 2010, p. 317 – 335
- [7] <https://goo.gl/78Zirc> (Dostęp 1.09.2017)
- [8] M. Szymkowski, K. Saeed – „A Novel Approach to Fingerprint Identification Using Method of Sectorization”, „IEEE Proceedings, 2017, International Conference on Biometrics and Kansei Engineering, Kyoto, Japan, September 15-17, 2017”
- [9] K. Surmacz, K. Saeed, P. Rapta – „An improved algorithm for feature extraction from a fingerprint fuzzy image”, Optica Applicata, Vol. 43, No. 3, 2013, p. 515 - 527



**MACIEJ SZYMKOWSKI**

szymkowskimack@gmail.com

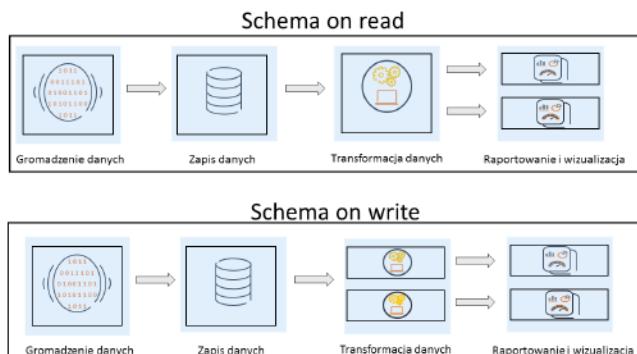
Inżynier Informatyk, Junior Software Developer w Symmetra Sp. z o. o., student Informatyki (II stopień) na Wydziale Informatyki Politechniki Białostockiej. Do swoich zainteresowań zalicza: Biometrię, Przetwarzanie i Analizę Obrazów oraz Sygnałów, a także Elektronikę i Kryptografię. Uwielbia poszerzać swoją wiedzę poprzez uczestnictwo w konferencjach oraz lekturę książek i artykułów. Programuje w językach takich jak: Java, C#, C czy C++. Ostatnio rozpoczął naukę Objective-C i Pythona.

# Data Lake – to robi różnicę

Przez lata hurtownie danych nie miały godnego rywala. Wraz z upływem czasu koszt przechowywania danych znacznie spadł, a alternatywne rozwiązania zaczęły dynamicznie dojrzewać.

**C**hcąc sprostać nowym wyzwaniom na rynku, wykształciło się zupełnie nowe podejście do składowania danych, które zostało określone hasłem *Data Lake*. James Dixon, dyrektor techniczny firmy Pentaho, stwierdził, że aby zilustrować, czym jest *Data Lake*, należy wyobrazić sobie klasyczny *Data Mart* jako butelkę wody mineralnej – czystej, zapakowanej, idealnie przygotowanej do spożycia. *Data Lake* jest czymś w rodzaju zbiornika pełnego wody w naturalnym stanie, która w dalszym ciągu, nieustannie napływa poprzez różne źródła. Dane są przechowywane w formie surowej, co wymaga dalszych nakładów pracy przed ich użyciem, ale stwarza znacznie więcej możliwości<sup>1</sup>.

Od dekad, gdy tylko relacyjne bazy danych zaczęły być powszechnie stosowane w praktyce gospodarczej, używano podejścia, które określono jako *schema on write*. Oznacza to, że najpierw należy określić schemat, a następnie załadować dane do utworzonych struktur. Wiąże się to z koniecznością doskonałej znajomości danych, wykonania wielu transformacji, tak aby zawierały one określony wcześniej typ i format, dzięki czemu są przygotowane do dalszych analiz – np. raportowania bądź budowania modeli predyktacyjnych. Koncepcja mimo wielu zalet stwarzała również mnóstwo ograniczeń. Przedsiębiorcy potrzebowali miejsca, w którym mogą przechowywać dane bez konieczności definiowania schematu. Chcieli oni mieć centralne repozytorium danych, w których będzie możliwe gromadzenie danych surowych, w oryginalnej postaci, a dopiero gdy zajdzie potrzeba wykorzystania ich w codzienności gospodarczej, stworzenia schematu, który idealnie spełnia wymagania aplikacji. W taki sposób wytworzyło się podejście znane jako *schema on read*, dające zdecydowanie większą elastyczność. Ciężko jest stworzyć jeden schemat, który będzie idealnie spełniał wymagania stawiane przez różne jednostki biznesowe w ramach przedsiębiorstwa. Przy wykorzystaniu koncepcji *schema on read* omawiany problem nie istnieje, ponieważ każdy z departamentów może przetworzyć dane w niezależny sposób, bez utraty potencjalnie wartościowych informacji. Oba podejścia zostały zображенione na Rysunku 1.



Rysunek 1. Zobrazowanie koncepcji Schema on read oraz Schema on write

1. James Dixon, Pentaho, *Hadoop and Data Lakes*: <https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/>, 17.08.2017.

Repozytorium *Data Lake* zapewnia również dostęp do danych z jednego, centralnego miejsca. Osoby wykorzystujące dane do raportowania, przeprowadzania wielowymiarowych analiz często zmuszone są do korzystania z wielu źródeł danych – hurtowni, plików płaskich, systemów źródłowych. Przy posiadaniu repozytorium *Data Lake* wszystkie potrzebne dane znajdują się w jednym miejscu, zatem dostęp do nich będzie znacznie ułatwiony, a w przypadku pojawienia się dodatkowych źródeł proces ich importowania znacznie łatwiejszy i szybszy, niż w przypadku załadowania ich do klasycznej hurtowni danych.

## ARCHITEKTURA DATA LAKE W OPARCIU O EKOSystem HADOOP

Projektując nowe repozytorium *Data Lake*, należy dobrze zrozumieć sposób, w jaki będzie ono w przyszłości wykorzystywane, aby przygotować architekturę, która będzie spełniała wszystkie wymagania biznesowe.

Pierwszą istotną kwestią jest zdefiniowanie źródeł danych i częstotliwości ich odświeżania. Dane przetwarzane w trybie *real-time* będą z reguły pozyskiwane innymi narzędziami, niż te, które zostaną ładowane w trybie batchowym (według harmonogramu pojawiania się kolejnych paczek danych). Prawdopodobnie zostaną one wykorzystane również do innych celów – analiz, w których istotny jest jak najkrótszy czas reakcji na zmieniające się uwarunkowania biznesowe. Warto byłoby rozważyć wykorzystanie bazy *in-memory* do przechowywania danych. *Data Lake* oparty o platformę Hadoop i system plików HDFS znajdzie większe zastosowanie w przetwarzaniu wsadowym, aniżeli *real-time*. Oczywiście wszystko zależy od rozważanego scenariusza, ponieważ niemożliwym jest znalezienie jednego rozwiązania, które idealnie spełni się w przypadku wszystkich możliwych scenariuszy.

Istotnym elementem jest także format danych, które będą gromadzone w repozytorium. Jeżeli istotną rolę odgrywają wzajemne relacje między nimi, warto oprzeć całą architekturę *Data Lake* o bazę grafową. Jednym z najbardziej elastycznych podejść jest zbudowanie repozytorium na bazie ekosystemu Hadoop. System plików HDFS zapewnia dużą elastyczność i różnorodność pod względem przechowywania danych. Nie panują tu praktycznie żadne ograniczenia, importowane mogą być całe tabele pochodzące z relacyjnych baz danych, pliki binarne zawierające np. obrazy, dźwięk, sygnały i wiele innych. Na etapie samego przechowywania danych nie musimy martwić się o sposób ich wykorzystania w przyszłości, architekturę rozwiązania, zastosowane narzędzia. Dodatkową zaletą jest możliwość organizowania danych w plikach, które mają strukturę kolumnową (np. Parquet, ORC). Zawierają one nie tylko dane, ale również metadane – informacje o ich strukturze, schemacie. Umożliwiają one zadowalającą kompresję danych, co przekłada się na efektywne planowanie przestrzeni dyskowej na repozytorium.

*Data Lake* posiada cztery podstawowe funkcje, które wspólnie umożliwiają ładowanie, przechowywanie i przetwarzanie danych

z wielu źródeł o różnorodnych formatach (od ustrukturyzowanych przez częściowo strukturyzowane do zupełnie nieustrukturyzowanych), aż do formy finalnej spełniającej wymagania użytkowników końcowych. Funkcje te zostały przedstawione w Tabeli 1 wraz z typowymi narzędziami służącymi do ich implementacji:

- » Ładowanie (*Ingestion*): łatwo skalowalne interfejsy dla ładowania danych surowych z różnych źródeł i formatów. Od klasycznych ładowań wsadowych aż do źródeł strumieniowych lub quasi/strumieniowych. Dane są ładowane w formacie oryginalnym, a pliki uszkodzone/dane niepełne nie są odrzucane.
- » Zapis i archiwizacja (*Storage & Backup*): warstwa przetwarzająca dane surowe we wstępnie przetworzone, oczyszczone i wyselekcjonowane dane, które będą wykorzystane w dalszej analizie. Dopuszczalne jest ich doprowadzenie do zoptymalizowanego formatu ogólnego przeznaczenia w systemach *Big Data*, np. ORC lub Parquet, oraz zastosowanie technologii przechowywania optymalnej dla źródła oraz wymagań biznesowych, np. Hive lub baza NoSQL.
- » Przetwarzanie (*Data Processing*): logika biznesowa i główna część transformacji pozwalająca na przygotowanie danych dla konkretnych analiz. Procesy są zwykle połączone w Workflow i wyzwalane zależnie od ustalonego z góry czasu (*time dependant*) lub w zależności od pojawienia się nowych danych (*data dependant*).

Dostęp do danych (*Data Access*): przetworzone i zoptymalizowane pod kątem danego przypadku użycia dane są udostępniane systemom zewnętrznym za pomocą odpowiednich interfejsów. Utylizacja mocy obliczeniowej *Data Lake* (klastra) jest dostępna przez szereg narzędzi wykorzystywanych przez użytkowników końco-

wych (systemy raportujące/algorytmy machine learning/narzędzia analityczne), które uruchamiają swoje zapytania bezpośrednio na klastrze, unikając ich transferu wpierw na własne serwery. Opisane funkcje *Data Lake* można odwzorować jako warstwy w widoku architektury, co przedstawiono na Rysunku 2.

Wszystkie warstwy *Data Lake* muszą być odpowiednio chronione oraz zarządzane. Solidna strategia utrzymania systemu *Data Lake* wymaga ustalenia polityki i standardów dostępu do danych, jak i szerokiego zastosowania metadanych, aby móc kontrolować ich stan oraz zarządzać dostępem do nich.

## WYZWANIA W TRAKCIE PROJEKTOWANIA I UTRZYMANIA DATA LAKE

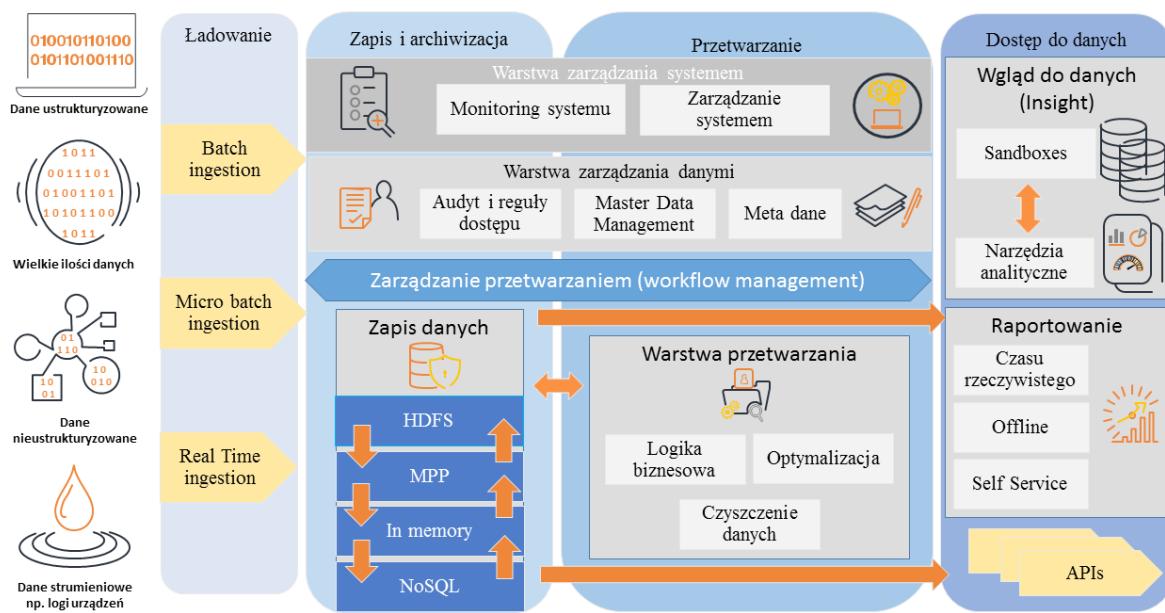
*Data Lake* nie jest łatwe zarówno w implementacji, jak i późniejszym utrzymaniu. Pomimo wielu zalet, jakie oferuje względem klasycznych rozwiązań typu Enterprise Data Warehouse, generuje nowe problemy, które należy mieć na uwadze, podejmując decyzję o wprowadzeniu tej technologii do przedsiębiorstwa. Najważniejsze z tych problemów to:

- » Trudne projektowanie: ekosystem Hadoop funkcjonuje technicznie w zupełnie innym sposobie niż klasyczne technologie DWH/BI. Do tego podlega nieustannym zmianom. Ciągle pojawiają się nowe narzędzia lub nowe wersje istniejących, które mogą być niekompatybilne z wcześniejszymi. Niektóre z kolei przestają być wspierane.
- » Złożoność technologiczna: niemal każde narzędzie w ekosystemie Hadoop funkcjonuje w inny sposób (np. implementacje w Java, Scala, JRuby, Python, JavaScript, konfiguratory ze swo-

Data Lake			
Ładowanie	Zapis i archiwizacja	Przetwarzanie	Dostęp do danych
Użyta technologia: Apache Flume Apache Kafka Apache Sqoop Apache Storm Python/shell NFS gateway	Użyta technologia: HDFS NoSQL (HBase,...) Solr/Elastic Hive MPP (Impala,...) In Memory (Redis,...) Hybrid (kilka technologii wspólnie)	Użyta technologia: Spark Flink Drill Map Reduce Hive Pig	Użyta technologia: Wizualizacja: Qlik Spotfire AngularJS Tableau  Interfejsy(API): REST API Kafka JDBC
Zarządzanie danymi, dostępy, monitorowanie			
Użyta technologia (zarządzanie, monitoring): Cloudera Navigator Cloudera Manager Ambari, Atlas MapR MCS		Użyta technologia (bezpieczeństwo): Sentry Kerberos Knox LDAP	

Tabela 1. Przykładowe narzędzia służące do implementacji poszczególnych warstw repozytorium *Data Lake*

# PROGRAMOWANIE BAZ DANYCH



Rysunek 2. Architektura Data Lake

ją skłonią i posiada różnorakie ograniczenia we współpracy z pozostałymi narzędziami. Orkiestracja wszystkich razem w spójny działający system spełniający zadane wymagania jest bardzo skomplikowana i złożona. Nie ułatwiają tego zadania setki błędów czekających na poprawienie od miesięcy przez programistów-wolontariuszy dla każdego z nich. Mimo że ekosystem Hadoop istnieje już przeszło 10 lat, wciąż brakuje wykwalifikowanej kadry IT oraz analityków, którzy byliby w stanie takie systemy sensownie projektować i implementować. Jest to szczególnie widoczne tam, gdzie droga w kierunku Big Data dopiero raczuje.

- » Efektywne ładowanie danych (*Ingestion*): mimo iż na pierwszy rzut oka ładować „wszelkie dostępne dane” z systemów źródłowych wydaje się nieskomplikowanym, proces ten stanowi prawdziwe wyzwanie dla architektów. Problemy, z którymi należy się zmierzyć, to:
  - » Utrzymanie aktualnych metadanych,
  - » Skalowalność rozwiązania – *Data Lake* zawierają nawet tysiące interfejsów różnych typów,
  - » Ciągłe monitorowanie jakości danych,
  - » Klasyfikacja bezpieczeństwa i dostępu do danych,
  - » Definicja sensu biznesowego ładowanych danych.

## Źródła

Architecting Data Lakes. Data Management Architectures for Advanced Business Use Cases oraz wewnętrzne publikacje firmy Capgemini.



### DAWID BENSKI

Starszy Architekt działu Insights & Data w firmie Capgemini Software Solutions Center we Wrocławiu. Posiada 7 lat doświadczenia w projektach bazodanowych i Business Intelligence oraz 3 lata doświadczenia w projektach Big Data.



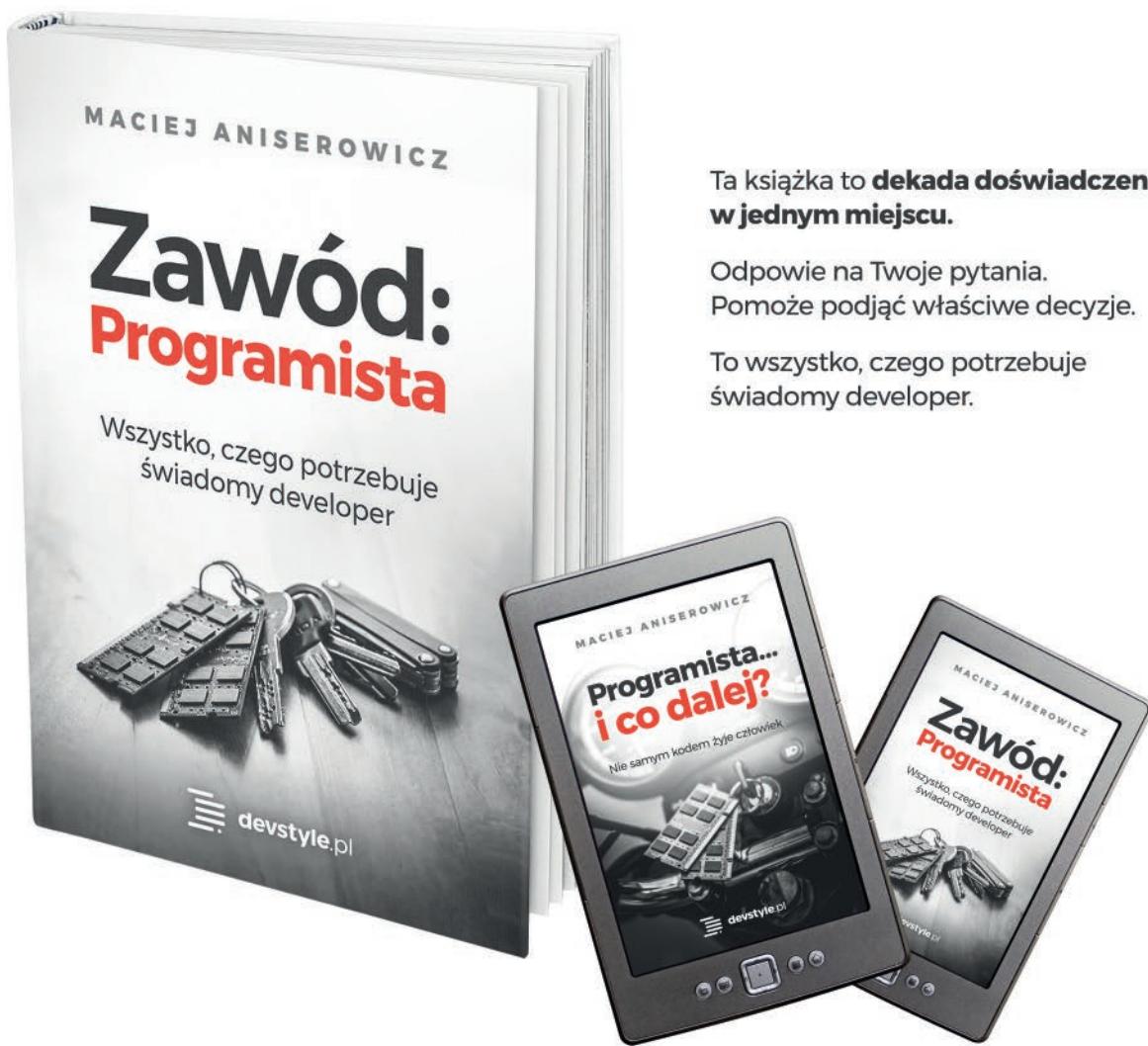
### MICHał DURA

Inżynier Oprogramowania w dziale Insights & Data w firmie Capgemini Software Solutions Center we Wrocławiu. Posiada 2 lata doświadczenia w projektach Big Data.

Jesteś doświadczonym programistą?  
**Wyciśnij maxa z tej profesji!**

Stawiasz pierwsze kroki w branży?  
**Mądrze pokieruj swoją karierą!**

Dopiero zamierzasz wejść w świat IT?  
**Zobacz, co Cię czeka!**



Ta książka to **dekada doświadczenia w jednym miejscu**.

Odpowie na Twoje pytania.  
Pomoże podjąć właściwe decyzje.

To wszystko, czego potrzebuje  
świadomy developer.



**devstyle.pl**  
ŚWIAT OKIEM PROGRAMISTY

**zawodprogramista.pl**

# Gitlab CI – od pomysłu do produkcji

Ciągła integracja jest sposobem na zwiększenie jakości kodu oraz skrócenie cyklu wytwarzania oprogramowania, a największe korzyści czerpiemy z niej, gdy jest w pełni zautomatyzowana. Przyjrzyjmy się z bliska mechanizmom Gitlab CI i zautomatyzujmy proces integracji naszej aplikacji.

## GITLAB – WIĘCEJ NIŻ REPOZYTORIUM

### *Ekosystem Gitlaba*

Gitlab to nie tylko repozytorium – to pełna platforma służąca do wersjonowania i rozwoju oprogramowania oraz zestaw narzędzi ułatwiających tworzenie procesów *continuous integration* i *continuous delivery*.



Rysunek 1. Od pomysłu do produkcji (źródło: [gitlab.com](https://gitlab.com))

Platforma, w przeciwieństwie do popularniejszego GitHuba, jest w całości udostępniana jako otwarte oprogramowanie. Dzięki temu organizacje mogą utrzymywać własną instancję we własnej infrastrukturze. Ponadto dostępne są plany komercyjne dostarczane przez wydawcę oprogramowania. Pełny spis dostępnych planów dostępny jest na stronie [gitlab.com](http://gitlab.com) [1].

## REPOZYTORIUM

Rdzeniem platformy Gitlab jest repozytorium kodu. Silnikiem repozytorium jest git, który podobnie jak w przypadku GitHuba do-

stępny jest za pomocą protokołów HTTPS oraz SSH. Pojedyncze repozytorium odpowiada projektowi – jeden projekt to jedno repozytorium. W ramach projektu konfigurujemy dodatkowe integracje i narzędzia (przykłady na Rysunkach 2, 3 i 4).

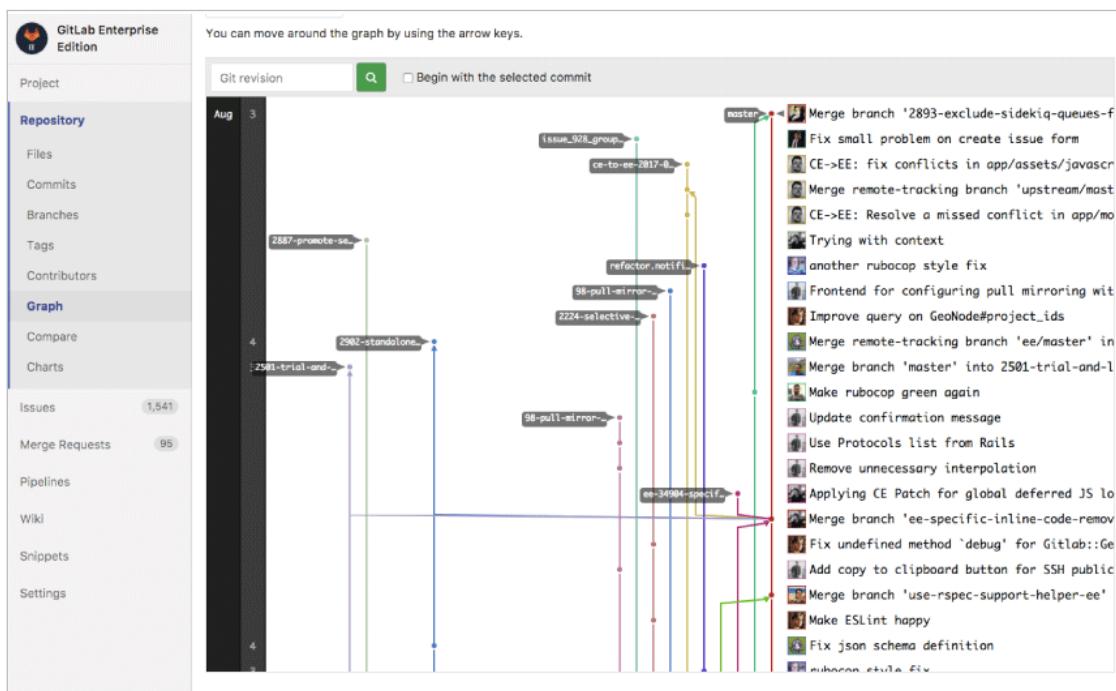
Rozróżniamy dwa rodzaje właścicieli projektów – użytkownika oraz zespołu. Gitlab pozwala definiować zespoły, które później mogą być właścicielami wielu repozytoriów. Zdecydowanie ułatwia to pracę grupy osób nad złożonym systemem, w którym każdy potrzebuje mieć dostęp do wielu repozytoriów. Taki sposób organizacji zachęca również zespoły do tworzenia mniejszych (lub mniejszych do utrzymania i organizacji) repozytoriów na poszczególne moduły projektów.

### Zarządzanie zadaniami

Kolejnym modułem, na który chciałbym zwrócić uwagę, jest *issue tracker* – narzędzie do śledzenia zadań. Jest to odpowiednik narzędzi takich jak JIRA firmy Atlassian [2]. W ramach systemu można zdefiniować kamienie milowe dla projektu (grupy funkcjonalności) – patrz Rysunek 3.

Podobnie jak odpowiadające mu komercyjne narzędzia – moduł śledzenia zadań wyposażony został w widok tablicy Kanban (Rysunek 4).

Pełną efektywność narzędzia uzyskujemy jednak dopiero wówczas, kiedy do obsługi rozwoju projektu w procesie rewizji



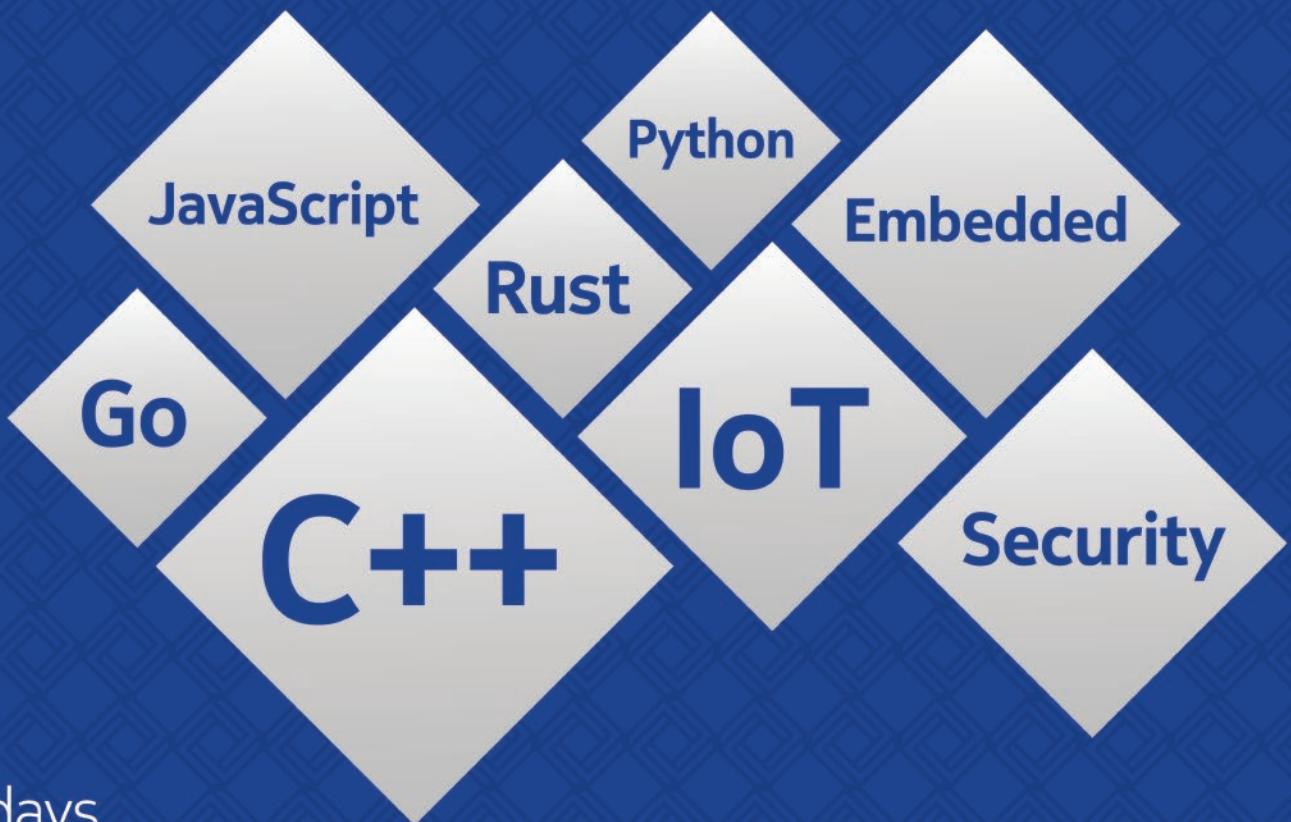
Rysunek 2. Graf branchy (źródło: [gitlab.com](https://gitlab.com))



# code::dive 2017

14-15 November 2017

Kino Nowe Horyzonty  
Kazimierza Wielkiego 19a -21  
Wrocław



2 days,  
4 stages,  
33 speakers,  
40 lectures,  
1500 participants

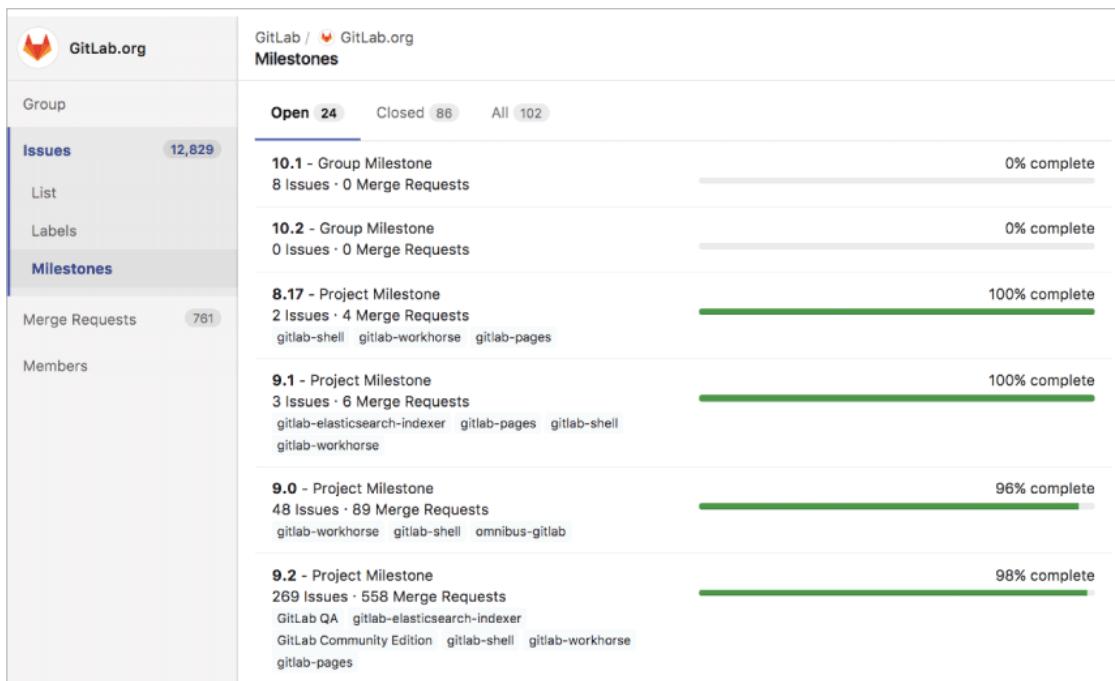
- you have to be there too!



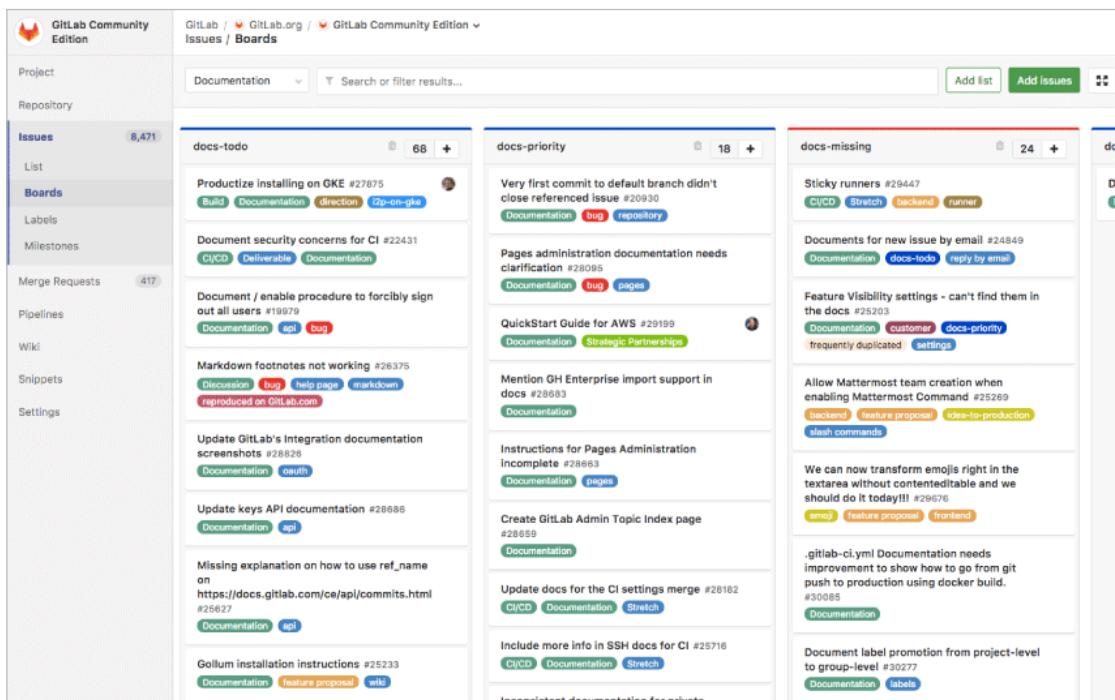
codedive.pl

powered by **NOKIA**

# TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ



Rysunek 3. Widok kamieni milowych (źródło: [gitlab.com](https://gitlab.com))



Rysunek 4. Tablica Kanban (źródło: [gitlab.com](https://gitlab.com))

kodu zastosujemy system *merge request* (odpowiednik *pull request* z GitHub). Dzięki pełnej integracji tych narzędzi możemy między innymi automatycznie zamknąć zadania w chwili pozytywnego zamknięcia *merge requestu*. Każde zgłoszenie może również otrzymać własną wersję rewizyjną aplikacji, którą można dostarczyć do testu funkcjonalnego i zweryfikować, czy problem został rozwiązany.

Ponadto system zarządzania żądaniami scaleń daje takie możliwości jak:

- » Tworzenie poufnych zgłoszeń błędów,
- » Przenoszenie zgłoszeń między projektami,
- » Możliwość wglądu w proces CI/CD dla danego rozwiązania,

- » Wydanie rewizyjnej wersji aplikacji zanim zmiany zostaną zaakceptowane,
- » Oznaczanie *merge request* jako rozwijanego w toku – dzięki czemu zespół nie poświęci czasu na jego rewizję, zanim zostanie ukończony.

Platforma Gitlab wyposażona jest w ogromny arsenał narzędzi realizujących zadania każdej warstwy pokazanej na Rysunku 1. W dalszej części artykułu zapoznamy się z wydawniczą częścią procesu – poznamy Gitlab CI.

Z pozostałymi, równie cennymi i znaczącymi modułami można zapoznać się na stronie [gitlab.com](https://gitlab.com) [3].

## Integracja z serwisami zewnętrznymi

Wiele zespołów decydujących się na korzystanie z platformy Gitlab z różnych względów potrzebuje nadal pracować z innymi narzędziami obsługującymi proces wytwarzania oprogramowania.

Project services	Service	Description
Project services allow you to integrate GitLab with other applications		
	<a href="#">Asana</a>	Asana - Teamwork without email
	<a href="#">Assembla</a>	Project Management Software (Source Commits Endpoint)
	<a href="#">Atlassian Bamboo CI</a>	A continuous integration and build server
	<a href="#">Bugzilla</a>	Bugzilla issue tracker
	<a href="#">Buildkite</a>	Continuous integration and deployments

Rysunek 5. Integracja usług zewnętrznych

Dzięki licznym opcjom integracji możemy na przykład nadal używać narzędzi JIRA do śledzenia zgłoszeń.

Ponadto możemy przygotować integrację z własnym oprogramowaniem za pomocą systemu Webhooks [4]. Z jego pomocą możemy powiadamiać systemy zewnętrzne o zmianach, które występują na różnych etapach procesu produkcyjnego. Mogą być to zarówno zdarzenia repozytorium, jak i innych etapów – stworzenie nowego zgłoszenia czy zakończony proces budowania.

The screenshot shows the 'Webhooks' section of the GitLab CI/CD configuration. It includes fields for 'URL' (http://example.com/trigger-ci.json), 'Secret Token' (a placeholder field), and a list of triggers:

- Trigger**
  - Push events**: This URL will be triggered by a push to the repository.
  - Tag push events**: This URL will be triggered when a new tag is pushed to the repository.
  - Comments**: This URL will be triggered when someone adds a comment.
  - Issues events**: This URL will be triggered when an issue is created/updated/merged.
  - Confidential Issues events**: This URL will be triggered when a confidential issue is created/updated/merged.
  - Merge Request events**: This URL will be triggered when a merge request is created/updated/merged.
  - Job events**: This URL will be triggered when the job status changes.
  - Pipeline events**: This URL will be triggered when the pipeline status changes.
  - Wiki Page events**: This URL will be triggered when a wiki page is created/updated.
- SSL verification**: An option to enable SSL verification.

Rysunek 6. Webhook – konfiguracja

## GITLAB CI

### CI/CD

Elementem platformy, któremu poświęcimy największą część artykułu, jest narzędzie do realizacji *continuous integration* i *continuous delivery* (ciągłej integracji/ciągłego dostarczania). CI/CD to proces, w którym każda zmiana w oprogramowaniu, która zostanie dostarczona przez programistę do repozytorium, zostanie automatycznie przetestowana i w razie konieczności wydana (na przykład w środowisku testowym).

Dzięki stosowaniu takich procesów minimalizujemy ryzyko powstania kodu, który jest trudny do zintegrowania, oraz wcześniej wykrywamy błędy. Usprawnia on również stosowanie praktyk TDD, unaoczniając w procesie CI/CD ilość wykonanych i zakończonych z powodzeniem testów.

### Inne rozwiązania tej klasy

Zanim przyjrzymy się dokładniej realizacji procesu CI/CD z Gitlab CI, warto zaznaczyć, że istnieje wiele narzędzi służących do automatyzacji tego procesu. Najpopularniejszym rozwiązaniem tej klasy (według slant.co [5]) jest Jenkins. Z jego pomocą możemy zdefiniować pojedyncze lub połączone zadania, które mają realizować kolejne etapy budowania, komplikacji, testowania i wydawania oprogramowania.

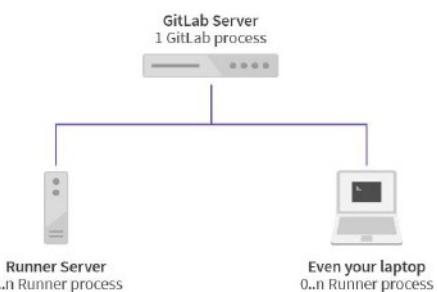
Przytoczony wyżej Jenkins pozwala na zbudowanie skomplikowanych procesów, jednak jest to oprogramowanie, które nie jest domyślnie świadome repozytorium kodu. Ze względu na swoją niezależność od systemu kontroli wersji Jenkins nie jest świadomym istnienia systemu zgłoszeń czy *merge requestów*, co utrudnia budowę złożonych procesów integracji.

### CI/CD w Gitlabie

Gitlab CI, jak wspomniano wcześniej, jest modelem platformy Gitlab. Dzięki tej integracji zarządzanie procesem odbywa się w jednym miejscu z repozytorium kodu – w związku z tym unikamy potrzeby utrzymania dedykowanej infrastruktury CI/CD. Do zalet tego rozwiązania można zaliczyć:

- » Automatyczne skalowanie,
- » System *Pipeline* służący do budowania przepływów,
- » Natywne wsparcie dla Dockera,
- » Budowanie równoległe,
- » Stabilność.

Architektura systemu jest bardzo prosta. Centralny serwer Gitlab jest koordynatorem zadań CI/CD, zgodnie z konfiguracją uruchamiającym procesy na podłączonych do niego maszynach zwanych *gitlab runner*.



Rysunek 7. Architektura Gitlab CI

*Gitlab runner* to jedyna maszyna, którą musimy skonfigurować do pracy z systemem. Jego ogromną zaletą jest możliwość dołączenia do niego nawet maszyn developerskich, co w niewielkich firmach może zdecydowanie przyspieszyć proces i obniżyć koszty.

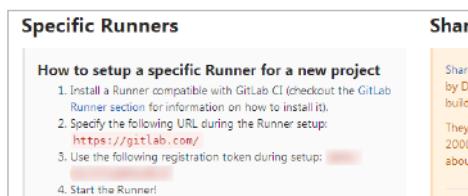
### Gitlab runner

Podczas pierwszej styczności z procesem CI/CD w Gitlabie natknijemy się na dwa podobne pojęcia: *gitlab runner* oraz *gitlab multi runner*. Różnica między nimi jest następująca:

- » *Gitlab multi runner* to usługa instalowana na maszynach wykonawczych. Nasłuchiwa ona zadań z serwera centralnego i zgodnie z konfiguracją uruchamia odpowiednie procesy.
- » *Gitlab runner* odpowiada pojedynczej konfiguracji środowiska uruchamiania. W ramach jednego *multi runnera* definiujemy wiele *runnerów*. Najczęściej poszczególne *runnery* różnią się od siebie sposobem wykonywania zadań. Pośród *executorów*, jak nazywa je dokumentacja [7], znajdziemy możliwość wykonywania w kontenerze dockerowym, bezpośrednio w powłoce lub za pośrednictwem ssh.

Preferowanym sposobem instalacji *gitlab multi runnera* jest zastosowanie dedykowanych paczek dla dystrybucji Linuksa. Dokładny opis instalacji dostępny jest w dokumentacji [6].

Posiadając maszynę zainstalowanym *multi runnerem*, możemy przystąpić do rejestracji *runnera* dla naszego projektu. W tym celu odwiedzamy sekcję *pipelines* w ustawieniach projektu.



Rysunek 8. Rejestracja runnera

Używając danych z tej strony na maszynie wykonawczej, wykonujemy kilka poleceń zgodnie z dokumentacją [7].

#### Listing 1. Rejestracja gitlab runnera

```
sudo gitlab-runner register
Please enter the gitlab-ci coordinator URL (e.g. https://
gitlab.com)
https://gitlab.com
Please enter the gitlab-ci token for this runner
TOKEN
Please enter the gitlab-ci description for this runner
moj runner
Please enter the gitlab-ci tags for this runner (comma
separated):
ulubiony-projet,docker-runner
Whether to run untagged jobs [true/false]:
[false]: false
Whether to lock Runner to current project [true/false]:
[false]: false
Please enter the executor: ssh, docker+machine, docker-
ssh+machine, kubernetes, docker, parallels, virtualbox, docker-
ssh, shell:
docker
Please enter the Docker image (eg. ruby:2.1):
alpine:latest
```

To wszystko – nasz *runner* jest gotowy do wykonywania zadań. Zapamiętajmy tagi, których używamy przy rejestracji – będą służyły do powiązania konkretnych etapów procesu z odpowiednim *runnerem*.

## PRZYKŁADOWY PROCES CI/CD

### Aplikacja

Jak dotąd wszystko wygląda bardzo obiecująco – maszyna wykonawcza jest gotowa do pracy, integracja nie była wymagająca. Ostatnim krokiem przygotowania procesu CI/CD jest opis jego przebiegu dla konkretnej aplikacji. Do prezentacji posłuży nam

aplikacja visit tracker dostępna pod adresem <https://gitlab.com/majk-p/visit-tracker/>, która powstała na potrzeby artykułu „Wdrażanie aplikacji w środowisku OpenShift” (Programista 05/2017). Opis jej działania znajdziemy w repozytorium.

### .gitlab-ci.yml

Opis procesu będzie wymagał od nas stworzenia w głównym katalogu naszego repozytorium pliku *.gitlab-ci.yml*. Jego dokładny opis został oczywiście wyczerpany w odpowiedniej sekcji dokumentacji [8].

Nasz plik *gitlab-ci.yml* będziemy wzorować na przygotowanym przez społeczność pliku [9] dostosowanym do wydawania aplikacji napisanych w języku Python na platformę Heroku.

#### Listing 2. Gitlab-ci.yml

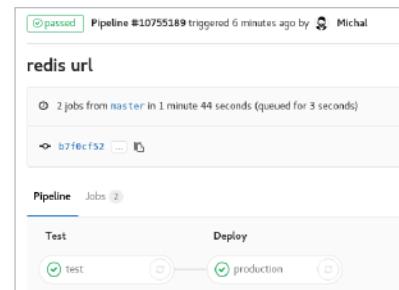
```
stages:
- test
- deploy

test:
image: python:3.6.2-stretch
script:
- pip install -r requirements.txt
- python handler.py -v

production:
stage: deploy
image: ruby
script:
- gem install dpl
- dpl --provider=heroku --app=visit-tracker
--api-key=$HEROKU_PRODUCTION_API_KEY
only:
- master
```

Opis nie jest skomplikowany. Możemy wyróżnić trzy sekcje: deklarację etapów oraz definicję każdego z nich. Deklaracje etapów (*stages*) pozwalają na grupowanie zadań, które odbywają się równolegle. Dzięki temu, jeśli aplikacja składa się z wielu części – zadania testowe można uruchamiać jednocześnie. Nasza aplikacja posiada jeden przykładowy test. Podział jest dobrze widoczny na Rysunku 9.

Kolejnym etapem jest wydanie aplikacji do środowiska produkcyjnego. W naszym przypadku aplikacja wydawana jest na platformie Heroku. W związku z tym zadanie realizowane jest w kontenerze opartym o obraz *ruby*, w którym kolejno instalujemy narzędzie *dpl*, następnie z jego pomocą wydajemy oprogramowanie.



Rysunek 9. Przepływ wygenerowany na podstawie gitlab-ci

### Przepływ CI/CD

Dzięki prostemu w swojej zawartości plikowi *.gitlab-ci.yml* otrzymaliśmy w pełni funkcjonalny przepływ. Nasza przykładowa aplikacja do prawidłowego wydania potrzebuje klucza API na platformie Heroku. Konto na platformie możemy założyć bezpłatnie, żeby uzyskać dostęp do dostarczanego przez platformę Redisa (apl-

kacja korzysta z niego jako *cache*), musimy uzupełnić informacje bilingowe (dostępny jest plan darmowy, informacje bilingowe są wymogiem regulaminowym i nie generują opłat).

Jak widać w Listingu 2, klucz API dostarczany jest za pośredniczeniem zmiennych środowiskowych. Klucz znajdziemy w ustawieniach profilu na platformie Heroku [10].



Rysunek 10. Klucz API Heroku

Tak uzyskany klucz musimy umieścić w konfiguracji naszego projektu. W tym celu w naszym projekcie na platformie Gitlab wchodzimy w *Settings => Pipelines* i uzupełniamy sekcję *Secret Variables*.

Secret variables			
Add a variable <b>Key</b> PROJECT_VARIABLE <b>Value</b> PROJECT_VARIABLE			
<b>Environment scope</b> * <small>This variable will be passed only to jobs with a matching environment name. * is a wildcard that matches all environments (existing or not). <a href="#">?</a></small>			
<input checked="" type="checkbox"/> <b>Protected</b> <small>This variable will be passed only to pipelines running on protected branches and tags. <a href="#">?</a></small>			
<a href="#">Add new variable</a>			
<b>Your variables (1)</b>			
<b>Key</b>	<b>Value</b>	<b>Protected</b>	<b>Environment scope</b>
HEROKU_APP_ID	*****	No	*

Rysunek 11. Zmienne środowiskowe przepływu CI/CD

To wszystko – nasz przepływ jest gotowy do działania. Teraz za każdym razem, kiedy wypchniemy nowy kod na *branch master*, nasze zmiany zostaną wydane na platformie Heroku. Gitlab pozwala śledzić przepływy w sekcji *Pipelines*.

All	Pending	Running	Finished	Branches	Tags
Status	Pipeline	Commit	Stages		
<span>passed</span>	#10755189 by	P master -> b7ff0cf52 best	  redis url	⌚ 00:01:44 🕒 35 minutes ago	
<span>passed</span>	#10754947 by	P master -> f05e3cbe requirements updated	 	⌚ 00:01:59 🕒 43 minutes ago	
<span>passed</span>	#10754609 by	P master -> f5ec0224 int removed	 	⌚ 00:02:06 🕒 53 minutes ago	
<span>passed</span>	#10754518 by	P master -> 3f39a1a8e heroku meta files	 	⌚ 00:01:40 🕒 56 minutes ago	

Rysunek 12. Sekcja pipelines

Na każdym etapie przepływu możemy przyjrzeć się dziennikowi przebiegu, a w razie błędów dowiedzieć się, co było ich przyczyną.

W sieci	
[1] <a href="https://about.gitlab.com/products/">https://about.gitlab.com/products/</a>	
[2] <a href="https://www.atlassian.com/software/jira">https://www.atlassian.com/software/jira</a>	
[3] <a href="https://about.gitlab.com/features/">https://about.gitlab.com/features/</a>	
[4] <a href="https://gitlab.com/help/user/project/integrations/webhooks">https://gitlab.com/help/user/project/integrations/webhooks</a>	
[5] <a href="https://www.slant.co/topics/799/~best-continuous-integration-tools">https://www.slant.co/topics/799/~best-continuous-integration-tools</a>	
[6] <a href="https://docs.gitlab.com/runner/install/linux-repository.html">https://docs.gitlab.com/runner/install/linux-repository.html</a>	
[7] <a href="https://docs.gitlab.com/runner/register/">https://docs.gitlab.com/runner/register/</a>	
[8] <a href="https://docs.gitlab.com/ee/ci/yaml/README.html">https://docs.gitlab.com/ee/ci/yaml/README.html</a>	
[9] <a href="https://goo.gl/mwHZj9">https://goo.gl/mwHZj9</a>	
[10] <a href="https://dashboard.heroku.com/account">https://dashboard.heroku.com/account</a>	

passed Job #28891377 triggered 39 minutes ago by Michał

```
Running with gitlab-ci-multi-runner v9.4.2 (660f6fe)
on docker-auto-scale (6ef328c)
Using Docker image python:3.6-stretch ...
Using docker image sha256:5a8088db35a1468f05983c186837a51f78088f054da79c40270d2b88bf785 for predefined container...
Pulling docker image python:3.6-stretch ...
Using docker image python:3.6-stretch ID:sha256:ece77c12fdb74b05cd70677bc643ef2f32670ea9f4201dd37e72a04d636 for build container...
Cleaning up ...
Cloning into '/Builds/mayk-pv/asset-tracker'...
Checking out b7ff0cf52 as master...
Skipping Git submodules setup
$ pip install -r requirements.txt
Collecting tornado>=4.5.1 (from -r requirements.txt (line 1))
  Downloading redis> 10.5-py2.py3-none-any.whl (60kB)
Collecting tornado>=4.5.1 (from -r requirements.txt (line 2))
  Downloading tornado-4.5.1.tar.gz (448kB)
Building wheels for collected packages: tornado
  Running setup.py bdist_wheel for tornado: finished with status 'done'
    Stored in directory: /root/.cache/pip/wheels/b4/83/cd/6a46b2633457269d161344755e676d24307189b7a67ff4b
Successfully built tornado
Installing collected packages: redis, tornado
Successfully installed redis-2.16.5 tornado-4.5.1
$ python handler.py -v
Trying:
  increment()
Expecting:
  1
ok
Trying:
  increment("1d")
Expecting:
  11
ok
3 items had no tests:
  main_
  RedisHandler
  main_.IncrementHandler.get
1 items passed all tests:
  2 tests in main_.increment
2 tests in 4 items,
2 passed and 0 failed.
Test passed.
208 succeeded
```

Rysunek 13. Przebieg testów

```
Preparing deploy
Cleaning up git repository with 'git stash --all'. If you need build artifacts for deployment, consider saving them.
Creating application archive
Deploying application
Uploading application archive
% Total % Received Xferd Average Speed Time Time Current
% Total % Received Xferd Average Speed Time Time Current
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 1971 0 0 100 1971 0 13713 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 1971 0 0 100 1971 0 13696 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Triggering new deployment
% Total % Received Xferd Average Speed Time Time Current
% Total % Received Xferd Average Speed Time Time Current
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 69 0 0 69 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 163 0 0 163 0 0 21 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 163 0 0 163 0 0 18 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 163 0 0 163 0 0 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 163 0 0 163 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 182 0 0 182 0 0 15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 182 0 0 182 0 0 14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 182 0 0 182 0 0 13 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 182 0 0 182 0 0 12 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 182 0 0 182 0 0 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 286 0 0 286 0 0 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
100 286 0 0 286 0 0 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
No stash found.
Job succeeded
```

Rysunek 14. Przebieg wydania

## PODSUMOWANIE

Gitlab jest prężnie rozwijaną platformą, która jest w stanie zastąpić jednocześnie wiele komercyjnych narzędzi. Odpowiednio utrzymywany – zależnie od wybranego sposobu w postaci wykupionego planu bądź samodzielnego hostingu – może wygenerować również znaczne oszczędności w kosztach licencyjnych w porównaniu z innym oprogramowaniem. Dzięki swoim liczonym integracjom oraz pokryciu całego procesu „od pomysłu do produkcji” ma szansę ułatwić zespołom developerskim optymalizację procesu wytwarzania oprogramowania.



**MICHał PAWLIK**

Architekt oprogramowania w firmie Nokia, entuzjasta programowania funkcyjnego i języka Scala. Absolwent informatyki Politechniki Wrocławskiej.

# Wyjdź poza schemat – testuj dostępność aplikacji

## Dlaczego accessibility testing jest tak ważne

Dostępność aplikacji mobilnych i webowych oznacza umożliwienie wszystkim użytkownikom Internetu, a w szczególności osobom niepełnosprawnym, doświadczać, rozumieć, nawigować i w pełni korzystać z aplikacji. Umiejętność testowania dostępności pozwala na podnoszenie jakości produktów oraz powiększenie grona użytkowników aplikacji.

Niepełnosprawność, o której mowa, to długotrwały stan występowania pewnych ograniczeń w funkcjonowaniu człowieka. Ograniczenia te mogą dotyczyć słuchu, wzroku, funkcji fizycznych bądź umysłowych. Bycie osobą niepełnosprawną nie musi jednak oznaczać cyfrowego wykluczenia. To, na ile przyjazny jest współczesny Internet, zależy w dużej mierze od chęci grafików, UX designerów, programistów i testerów, by sprostać nowym wymaganiom dotyczącym jakości.

Mogliby się wydawać, że osoby niepełnosprawne, ze względu na swoje manualne lub poznawcze ograniczenia, stanowią niewielki procent użytkowników aplikacji webowych i mobilnych. W kontrze do tego twierdzenia stoją jednak wyniki badań prowadzonych przez Google Inc. od 2005 roku (<https://www.google.co.uk/accessibility/initiatives-research.html>), które dowodzą, że na świecie jest ponad miliard użytkowników z niepełnosprawnościami, korzystających na co dzień z aplikacji webowych i mobilnych. Liczba ta może robić wrażenie i pokazuje skalę nowego rynku użytkowników wymagających poprawy jakości software'u, z którego na co dzień korzystają.

Testowanie dostępności i poprawa użyteczności treści stron internetowych, szczególnie dla przeglądarek mobilnych, to już nie tylko kaprys lub fanaberia. Określenie, jakie testy dostępności powinny być wykonane i jak duże pokrycie aplikacji jest wymagane, są wyzwaniem, z którym mierzy się obecnie wiele organizacji. Poprawa dostępności wyznacza nowy status użyteczności i przyjaznego designu, z którym rozpoznawalne marki chcą być kojarzone.

Na czym polega przystosowanie treści strony internetowej dla potrzeb osób niepełnosprawnych? Przede wszystkim na pisaniu wysokiej jakości kodu HTML, z którym mogą bez problemu porozumieć się aplikacje wspierające osoby niepełnosprawne w obsłudze komputera czy smartfona.

Zaniedbania w kodzie, takie jak brak unikalnych id, które często wydają się programistom wygórowanymi wymaganiem ze strony testerów piszących testy automatyczne, są ważnymi błędami wykrywanymi przez testy dostępności, ponieważ sprawiają, że kod staje się nieczytelny dla asystentów mowy lub innych aplikacji ułatwiających osobom niepełnosprawnym korzystanie z naszego oprogramowania.

Inne błędy, powodujące problemy z dostępnością aplikacji, mogą wynikać z niewiedzy. Na przykład: zdjęcie umieszczone w treści strony internetowej, w której powinno dać się kliknąć, aby przejść dalej,

będzie niedostępne dla asystenta mowy osoby niewidomej, chyba że oprócz zdjęcia dodany zostanie aktywny tekst, aby aplikacja asystenta mowy mogła go odnaleźć i odczytać.

### PODSTAWOWE ZASADY TWORZENIA APLIKACJI SPEŁNIAJĄCYCH WYMAGI DOSTĘPNOŚCI

Tworząc aplikacje zaspokajające wyjątkowe potrzeby użytkowników, należy zwrócić szczególną uwagę na implementację elementów interfejsu użytkownika, takich jak:

- » kolory – osoby dotknięte chorobami wzroku, które uniemożliwiają prawidłowe postrzeganie barw, mogą być wykluczone z grona odbiorców aplikacji, jeśli nie będą mogły prawidłowo rozróżnić poszczególnych elementów. Można to zweryfikować stosując konkretne filtry na ekran (np. ze strony <http://colororacle.org/> – kompatybilne z Windows, Mac i Linux), pozwalające na zobaczenie świata oczami z konkretną dysfunkcją.
- » ikonki – ich wielkość i prostota mają szczególne znaczenie dla osób ograniczonych motorycznie lub z trudnościami poznawczymi.
- » możliwość zmiany rozdzielczości wyświetlanych elementów.
- » łatwość użycia i instalacji aplikacji.

oraz elementów backendu:

- » regulację głośności,
- » dostosowanie do asystentów mowy, np. VoiceOver (iOS) lub TalkBack (Android),
- » odpowiednią strukturę kodu HTML aplikacji tak aby umożliwić komunikację między naszą aplikacją, a oprogramowaniem typu asystent mowy dla osób niewidomych,
- » możliwość użycia audio deskrypcji.

Sposoby testowania aplikacji pod kątem dostępności również muszą być dostosowywane do unikalnych potrzeb użytkowników.

W praktyce oznacza to przede wszystkim zwiększoną ilość testów użyteczności. Nie chodzi jednak o subiektywne odczucia estetyczne w stosunku do danej aplikacji, lecz o konkretne zestawy testów zaprojektowanych w oparciu o potrzeby danych grup użytkowników. Spectrum możliwości testera jest nieograniczone – wymagana jest kreatywność i po części postawienie się w roli osoby z określona potrzebą, a w konsekwencji wybranie zestawu

testów ukierunkowanych na dogłębne sprawdzanie wąskiego obszaru aplikacji, np. wybranych elementów interfejsu graficznego.

Należy też zwrócić uwagę na to, że o ile w warstwie kodu wymagania dotyczące jakości są proste – kod musi być przemyślany, czytelny i przystosowany do współpracy z różnego rodzaju asystentami – o tyle opracowanie zestawu testów wymaga dużej pomysłowości i empatii ze strony testera.

Poprawnie działająca aplikacja dostępna dla osób niepełnosprawnych powinna zawsze zapewniać:

- » łatwą instalację,
- » nawigowanie bez pomocy wzroku,
- » możliwość używania aplikacji za pomocą prostych gestów,
- » czytelną informację zwrotną dla użytkownika.

Tworzenie aplikacji mobilnych i stron internetowych, które będą współpracowały z asystentami wspomagającymi osoby niepełnosprawne, to nie tylko poprawna implementacja podstawowych funkcjonalności, lecz także możliwość zastąpienia lub wzbogacenia konkretnych akcji w aplikacji innymi – np. poprzez dodanie vibracji przy każdym dotknięciu aktywnego elementu, powiększenie czcionki lub podświetlenie fragmentów obrazu dla osób niedowidzących. Testowanie takich aplikacji powinno sprawdzać, czy:

- » istnieje możliwość nawigowania bez użycia ekranu dotykowego,
- » audio deskrypcja jest dostępna dla każdego widzialnego elementu,
- » istnieją obszary aplikacji pozbawione audio deskrypcji,
- » wszystkie elementy, w które można kliknąć, mają co najmniej 48 dp (9 mm) długości i szerokości,
- » wszystkie elementy głosowe mają dodatkowy mechanizm wspierający osoby niesłyszące lub niedosłyszące.

Zarówno przed twórcami asystentów mowy, jak i twórcami aplikacji stoi to samo wyzwanie – poszukiwanie takich rozwiązań – aby bez utraty cech nowoczesnego produktu zachować spójność i dostępność aplikacji. Każde nowe rozwiązanie powinno być wspierane przez obie strony.

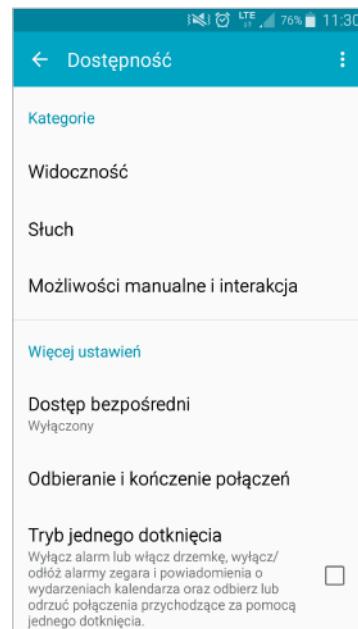
## TESTOWANIE DOSTĘPNOŚCI DLA OSÓB NIEWIDOMYCH

Nie każdy miał możliwość obserwowania, w jaki sposób osoba niewidoma korzysta ze smartfona. Tak – osoby niewidome korzystają ze smartfonów, tak – samodzielnie instalują aplikacje i tak – oczekują, że ich potrzeby w tym zakresie będą wspierane. Niewiele jest aplikacji dedykowanych osobom niepełnosprawnym, zazwyczaj osoby niewidome korzystają z klasycznych aplikacji, lecz w inny sposób.

Smartfon osoby niewidomej przez większość czasu ma wygaszony wyświetlacz (włączona jest tzw. kurtyna), ale wbudowany asystent mowy nieustannie pracuje w tle, aby umożliwić głosową nawigację oraz dotykowy wybór wszystkich dostępnych funkcjonalności. Aby się przekonać, czy Twoja aplikacja przystosowana jest do obsługi przez symulatory mowy, należy uruchomić funkcję dostępności (Accessibility) w urządzeniu.

### Przykład dla smartfona z systemem Android (ta procedura wymaga użycia wzroku):

1. Wybierz kolejno *Ustawienia > Ułatwienia dostępu*.
2. Wykonaj instrukcję w zależności od wersji Androida.
- » Android 4.1 lub nowszy – wybierz *TalkBack* i przesuń przełącznik *TalkBack* do pozycji Wł.



Rysunek 1. Menu Accessibility w telefonie z Androidem

- » Android 4.0 – wybierz *TalkBack* i przesuń przełącznik *TalkBack* do pozycji Wł. Przejdz do poprzedniego ekranu i włącz *Czytanie dotykiem*.
- » Android 3.2 i wcześniejsze – zaznacz pole *Ułatwienia dostępu*, a następnie *TalkBack*.
- 3. Na ekranie potwierdzenia pojawi się lista uprawnień, które umożliwiają *TalkBack* podawanie komunikatów głosowych. Aby potwierdzić, że zezwalasz na te działania, i rozpocząć korzystanie z *TalkBack*, kliknij *OK*.

### Przykład dla iPhone:

Po pierwszym włączeniu iPhone lub iPada naciśnij trzykrotnie przycisk *Początek*.

Po uruchomieniu *TalkBack/VoiceOver* aplikacje zainstalowane na smartfonie są gotowe, aby przejść prawdziwą próbę ognia – bez problemu można zweryfikować użyteczność wszystkich aktywnych elementów w każdej aplikacji, odczytać tekst na stronach internetowych, poruszać się pomiędzy ekranami, a także uruchamiać i zamykać aplikacje.

Testując, musimy na chwilę postawić się w sytuacji osoby niewidomej i korzystać jedynie z funkcji głosowych. Ponadto prawidłowo zaimplementowane funkcje dostępności dla asystenta mowy będą uruchamiały delikatne vibracje urządzenia przy każdej zmianie stanu aplikacji. Dodatkowo warto zapamiętać, że jeśli asystent mowy radzi sobie z odczytywaniem treści w języku angielskim, nie zawsze będzie oznaczało, iż równie bezbłędnie odczyta testy w lokalnym języku użytkownika.

## TESTOWANIE DOSTĘPNOŚCI DLA OSÓB NIESŁYSZĄCYCH

Potrzeba komunikacji jest jedną z silniejszych w społeczeństwie. W związku z tym technologie wykorzystywane w nowoczesnych smartfonach, takie jak video rozmowa czy komunikatory tekstowe, w doskonały sposób mogą zastąpić osobom niesłyszącym i niedosłyszącym tradycyjną rozmowę głosową, z czym głównie kojarzymy telefon.

Według danych T-mobile z 2015 roku (<https://www.t-mobile-trendy.pl/>) w Polsce jest 470 tysięcy osób niesłyszących i 2 miliony osób niedosłyszających. Aplikacje dostosowane do ich potrzeb muszą być testowane pod kątem możliwości zastąpienia dźwięku innymi funkcjonalnościami – dodatkowym opisem, wyświetlanym tekstem, vibracjami urządzenia etc. Nie chodzi tylko o to, aby testując, używać danej aplikacji bez dźwięku, ale by upewnić się, że istotne elementy, które zazwyczaj są sygnalizowane przez dźwięk – np. przychodząca wiadomość – są oznaczone w inny, dodatkowy sposób (migająca dioda, vibracja). Uruchomienie asystenta na urządzeniu mobilnym, wspierającego osoby niesłyszące, możliwe jest w podobny sposób jak włączenie asystenta dla osób niewidomych poprzez opcję w menu Accessibility.



Rysunek 2. Możliwości dostosowania kolorów w iPhone (źródło: <https://support.apple.com/>)

## RESPONSIVE WEB DESIGN

Tak samo jak w przypadku aplikacji mobilnych, strony internetowe, zwłaszcza strony, które wykorzystują responsywny design (RWD), powinny być testowane pod kątem użyteczności – na wszystkich przeglądarkach oraz na rzeczywistych urządzeniach mobilnych (lub na symulatorach) – w różnych rozdzielcościach ekranu. W przypadku testów dostępności to właśnie zmiany rozdzielcości elementów na stronie mają kluczowe znaczenie.

Dobrze zaimplementowany responsywny design na stronie internetowej może być odpowiedzią na wiele potrzeb osób niedowidzących lub z niepełnosprawnością ruchową. Testy użyteczności pod kątem dostępności strony internetowej powinny być poszerzone o:

- » weryfikację rozdzielcości i skalowania elementów z naciśnięciem na maksymalne powiększanie elementów,
- » sprawdzenie jakości i działania elementów graficznych w bardzo dużej rozdzielcości,
- » sprawdzenie poprawności wyświetlania czcionek w dużej rozdzielcości,

- » weryfikację łatwości nawigowania po wszystkich aktywnych elementach na stronie internetowej w różnych rozdzielcościach ekranu.

## NARZĘDZIA WSPOMAGAJĄCE TESTOWANIE DOSTĘPNOŚCI

Istnieje wiele narzędzi i serwisów, które mogą pomóc developerom i testerom w testowaniu dostępności lub wykonać podstawowe testy dymne w tym zakresie, sprawdzające poprawność kodu HTML. Część z tych serwisów jest darmowa i stworzona przez pasjonatów, którym zależy na poprawie dostępności Internetu dla wszystkich, więc zdecydowanie warto z nich skorzystać.

Jednym z najbardziej popularnych narzędzi testujących backend aplikacji jest AATT (Automated Accessibility Testing Tool) stworzone przez PayPal. Pozwala ono na użycie API odpowiedniego dla aplikacji spełniających wymogi dostępności oraz na dostosowanie kodu HTML (<http://github.com/paypal/AATT>).

Istnieje też możliwość testowania dokumentów PDF – tego typu dodatek do testowania zapewnia między innymi Adobe (Acrobat XI Pro) – gdzie w wygodny sposób możemy zidentyfikować konkretnie problemy z dostępnością tworzonych dokumentów i poznaczyć wskazówki na temat ich ulepszenia.

Dodatkowo Google (dostawca Accessibility Developer Tools – dodatku do Chrome Developer Tools) czy Apple (<https://www.apple.com/accessibility/iphone/vision/>) prowadzą serwisy edukacyjne z gotowymi rozwiązaniami dla developerów i prostymi narzędziami testowymi, umożliwiającymi poszerzanie swojej wiedzy i umiejętności testowania dostępności aplikacji.

Oczywiście, jak w przypadku każdego rodzaju testów, nic nie zastąpi naszej pomysłowości i chęci psucia aplikacji. Warto więc przyjrzeć się sposobowi działania automatycznych narzędzi, aby potem móc wzbogacać scenariusze testowe o nowe, kreatywne przypadki użycia.

## PODSUMOWANIE

W dobie walki o nowe rynki dla aplikacji mobilnych i o poszerzanie grona użytkowników serwisów społecznościowych testowanie dostępności może być dobrą wskazówką dla developerów, testerów i UX designerów.

Rozwój nowych technologii powinien pociągać za sobą wzrost świadomości i odpowiedzialności społecznej wszystkich twórców aplikacji. W takim ujęciu – testowanie dostępności przestaje być tylko fanaberią czy dobrą wolą testerów, a staje się realną potrzebą, wynikającą z chęci zapewnienia dostępu do wszelkich usług każdemu użytkownikowi Internetu, bez względu na jego biegłość techniczną, sprawność ruchową czy wiek.



**KINGA WITKO**

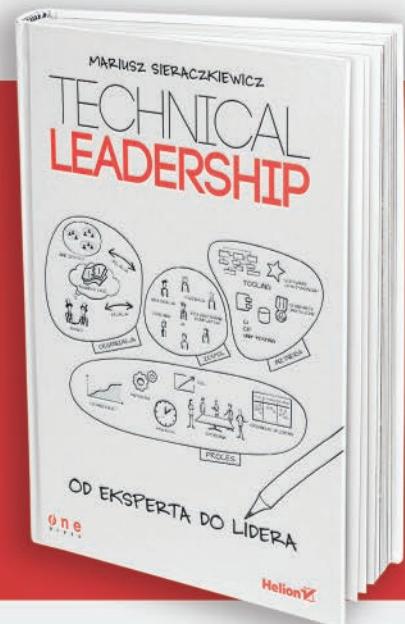
[Mrs.witko@gmail.com](mailto:Mrs.witko@gmail.com)

Dziewczyna w IT, pasjonatka kotów i psucia aplikacji mobilnych. Z zawodu – Tester Oprogramowania w New Voice Media sp z o.o. we Wrocławiu. Nieuustannie poszerza swoją wiedzę na temat testowania oprogramowania i zwinnych metod pracy w zespole. Wie, jak wyjść z VIMa. Autorka bloga o testowaniu <http://kingatest.wordpress.com>. Możecie śledzić ją na Twitterze <https://twitter.com/kingatest>.

# BNS IT - SZKOLENIA OTWARTE

WARSZAWA / 08-10.01.2018  
**TECHNICAL LEADERSHIP™**  
ROLA LIDERA TECHNICZNEGO

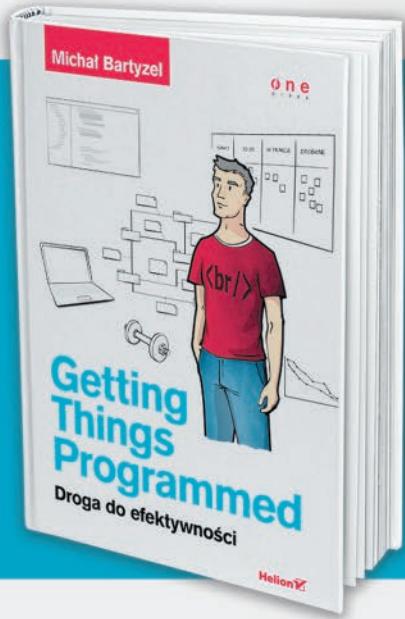
1. Rola lidera technicznego
2. Motywacja własna i innych
3. Ludzie
4. Zespół
5. Kompetencje lidera



ŁÓDŹ / 07-08.12.2017  
**GETTING THINGS PROGRAMMED**  
DROGA DO EFEKTYWNOŚCI

1. Oprogramowanie „na czas”
2. Opracowywanie zadań
3. Planowanie pracy programisty
4. Wykonywanie zadań programistycznych
5. Szacowanie zadań programistycznych

[www.getting-things-programmed-edycja2.evenea.pl](http://www.getting-things-programmed-edycja2.evenea.pl)



## P O Z O S T A Ł E   S Z K O L E N I A   O T W A R T E :

Zbieranie wymagań i współpraca z klientem	Łódź	13-15.11.2017	2100,00 PLN
Techniki pracy z kodem	Warszawa	29.11-1.12.2017	2100,00 PLN
Kanban w 1 dzień	Warszawa	08.12.2017	1300,00 PLN
Zespoły rozproszone - techniki skutecznej pracy zdalnej	Warszawa	14-15.12.2017	1800,00 PLN
Wzorce projektowe i refaktoryzacja do wzorców	Łódź	18-20.12.2017	2100,00 PLN
Nowoczesne architektury aplikacji	Warszawa	21-23.02.2018	2100,00 PLN
CENY NETTO			

# Praktyczna kryptografia: Szyfry blokowe

„Ktakolwiek, poczynając od najbardziej bezmyślnego amatora, na najlepszym kryptografie kończąc, może stworzyć algorytm, którego on sam nie będzie mógł złamać”. Właśnie tak brzmi Prawo Schneiera – amerykańskiego badacza bezpieczeństwa i kryptografa. Postaramy się je zweryfikować i przekonać przy tym czytelników, że nawet korzystając z silnych, nowoczesnych algorytmów, można popełnić błąd, który kompletnie rujnuje bezpieczeństwo całej aplikacji.

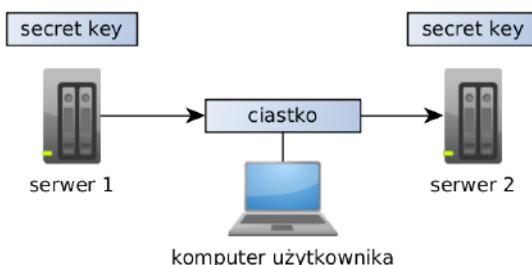
**W**spomniane prawo odnosi się konkretnie do tworzenia własnych kryptosystemów, ale w praktyce równie mocno dotyczy całych programów, które korzystają bezpośrednio z prymitywów kryptograficznych. Jest to szczególnie niepokojące, bo o ile błędy w działaniu zostaną wyłapane przez testy albo przez użytkowników aplikacji, to słaba kryptografia nie będzie budziła żadnych podejrzeń – do momentu, kiedy ktoś nie wykorzysta jej, żeby okraść naszych klientów lub wykrackać nasze dane.

W tym artykule skoncentrujemy się na szyfrach blokowych. Jako przykładu będziemy używać krótkich programów napisanych w języku Python (2.7), ale wszystkie przedstawione ataki są niezależne od języka. Co ważniejsze – dla demonstracji używamy AES, ale ataki byłyby dokładnie takie same na każdy inny szyfr blokowy (np. 3DES, Blowfish, Serpent, a nawet przyszłe, jeszcze nie wymyscone algorytmy).

## SZYFROWANIE NIE SŁUŻY DO UWIERZYTELNIANIA

### Wstęp

Wyobraźmy sobie następującą sytuację: mamy dwie niezależne aplikacje webowe, które z jakiegoś powodu muszą uwierzytelniać się sobie wzajemnie. Na przykład, jedna strona to forum internetowe, a druga to chat internetowy<sup>1</sup>. Jeśli chcemy, żeby można było się logować na chat za pomocą konta z forum, jedną z najprostszych opcji jest ciastko (lub token) współzielone między tymi dwoma aplikacjami. Pierwszy serwer wystawia jakieś ciastko/token, a użytkownik później używa tego ciastka, żeby wykonać jakieś akcje na drugim serwerze.



Rysunek 1. Serwer 1 przekazuje zaszyfrowane dane użytkownikowi, a użytkownik do serwera 2

1. Przykład inspirowany forum [4programmers.net](http://4programmers.net) i chatem, który był tam instalowany przez autora niniejszego artykułu. Choć oczywiście bez popełnienia opisanych tu błędów.

Potrzebujemy jakiegoś prostego formatu serializacji do współdzielenia danych – standardem obecnie jest JSON (jeśli ktoś woli XML, urlencoding lub inne – omawiane metody, po drobnych modyfikacjach, zadziałają dla dowolnego generycznego sposobu serializacji).

```

def serialize_cookie_0(name, has_admin):
    return json.dumps({
        'name': name,
        'has_admin': has_admin
    })

def deserialize_cookie_0(cookie):
    return json.loads(cookie)
  
```

Przetestujmy kod:

```

cookie_msm = serialize_cookie_0('msm', False)
print cookie_msm
print deserialize_cookie_0(cookie_msm)
# {"name": "msm", "has_admin": false}
# {"u'name': u'msm', u'has_admin': False}

cookie_monk = serialize_cookie_0('monk', False)
print cookie_monk
print deserialize_cookie_0(cookie_monk)
# {"name": "monk", "has_admin": false}
# {"u'name': u'monk', u'has_admin': False}
  
```

I tak zakodowane ciastka wysyłam użytkownikowi. Czy jest tu jakiś problem? Oczywiście, ktoś, kto wie, jak działają ciastka w przeglądarkach, już po kilku sekundach rozpozna zagrożenie – użytkownik ma pełną władzę nad własną przeglądarką, więc może po prostu zmienić sobie wartość swojego ciastka na:

```
{"name": "msm", "has_admin": true}
```

A następnie zalogować się jako administrator na chat. Jest to zdecydowanie niepożądane – użytkownik powinien mieć dostęp tylko do swojego konta. W jaki sposób temu zapobiec?

Jeden z pomysłów, na który łatwo wpaść i który jest czasami (niestety) stosowany: zaszyfrować ciastko kluczem dzielonym między dwoma serwerami.

### Trochę teorii o szyfrowaniu

Napiszmy więc odpowiedni kod. JSON pozostanie bez zmian (już do końca artykułu w zasadzie), natomiast dopiszmy odpowiednie szyfrowanie do serializacji.

Najpierw kilka koniecznych słów teorii. Rozróżniamy dwa rodzaje szyfrów symetrycznych:



Zapraszamy na autorskie szkolenia  
z zakresu **bezpieczeństwa IT**

{ Bezpieczeństwo aplikacji WWW }

{ Offensive HTML, SVG, CSS and other Browser-Evil }

{ Wprowadzenie do bezpieczeństwa IT }

{ Szkolenie przygotowujące do egzaminu CEH  
( Certified Ethical Hacker ) }

[www.securitum.pl/oferta/szkolenia](http://www.securitum.pl/oferta/szkolenia)

Patroni medialni: [sekurak.pl](http://sekurak.pl)



[rozwal.to](http://rozwal.to)

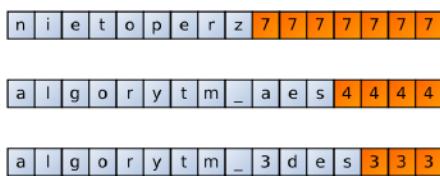


- » Szyfry strumieniowe, gdzie szyfrujemy dane bit po bicie (w praktyce bajt po bajcie).
- » Szyfry blokowe, które szyfrują dane „blokami”, czyli np. po 16 bajtów. Długość szyfrowanych danych musi być wielokrotnością długości bloku (np. 32, 48... bajtów).

Większość stosowanych obecnie szyfrów jest blokowa – w szczególności AES, jeden z najpopularniejszych algorytmów szyfrowania, działa blokowo. Wiąże się z tym fakt, że jeśli mamy dane o długości np. 7 bajtów, przed zaszyfrowaniem musimy je w jakiś sposób „wydłużyć” do wielokrotności długości bloku. Operacja ta profesjonalnie nosi nazwę „padding” (czasami tłumaczona jako „wyrównanie”, „wypełnienie”) i musi być odwracalna – żeby można było przy deszyfrowaniu bezstratnie ją odwrócić. Proste ćwiczenie dla chętnych programistów – wymyślić samodzielnie, jak coś takiego zrobić. Rozwiązań jest wiele<sup>2</sup>, ale najpopularniejszym z nich jest PKCS#7 padding.

Zasada działania jest prosta. Jeśli chcemy mieć blok o długości 16 bajtów, to:

- » Jeśli do wielokrotności 16 bajtów brakuje 1 bajta, dodajemy na koniec danych "\x01" (jeden bajt o wartości 1).
- » Jeśli do wielokrotności 16 bajtów brakuje 2 bajtów, dodajemy "\x02\x02".
- » Jeśli do wielokrotności 16 bajtów brakuje 3 bajtów, dodajemy "\x03\x03\x03".
- » ...
- » Jeśli długość danych już jest wielokrotnością 16 bajtów, na koniec dodajemy 16 bajtów "\x10" (16 bajtów równych 16 – inaczej dekodowanie niestety nie byłoby jednoznaczne).



Rysunek 2. Sposób działania paddingu PKCS#7. Dodawane jest N bajtów o wartości N

Implementacja również nie jest skomplikowana:

```

def pkcs7_pad(data):
    pad_len = 16 - (len(data) % 16)
    pad_len = 16 if pad_len == 0 else pad_len
    return data + chr(pad_len) * pad_len

def pkcs7_unpad(data):
    pad_len = ord(data[-1])
    data, pad = data[:-pad_len], data[-pad_len:]
    assert all(c == pad[0] for c in pad)
    return data
  
```

Testy:

```

msg = 'msm_lubi_koty'
padded_msg = pkcs7_pad(msg)
print repr(padded_msg)
print pkcs7_unpad(padded_msg)
# 'msm_lubi_koty\x03\x03\x03'
# msm_lubi_koty
  
```

## Szyfrowanie 1: ECB

Ok, wyobraźmy sobie, że szef kazał nam zaimplementować takie właśnie zabezpieczenie dla ciastek. Czas zacząć pisać kod:

```

from Crypto.Cipher import AES

# openssl rand -base64 16 - 16 bajtów entropii powinno uchronić
# przed atakami brute-force do końca świata
SHARED_KEY = 'KDbWBsg11cC8vAAmIT/+Ig=='.decode('base64')

def serialize_cookie_1(name, has_admin):
    raw_cookie = json.dumps({
        'name': name,
        'has_admin': has_admin
    })
    cipher = AES.new(SHARED_KEY)
    raw = pkcs7_pad(raw_cookie)
    return cipher.encrypt(raw).encode('hex')

def deserialize_cookie_1(cookie):
    cipher = AES.new(SHARED_KEY)
    raw_cookie = cipher.decrypt(cookie.decode('hex'))
    cookie = pkcs7_unpad(raw_cookie)
    return json.loads(cookie)
  
```

Zaznaczę tutaj, że ten kod jest poprawny<sup>3</sup> – w tym sensie, że programista nie popełnił żadnego błędu przy implementacji szyfrowania, klucz jest silny, padding wykonany poprawnie, a wszystko inne korzysta z wartości domyślnych biblioteki PyCrypto.

Testy kodu:

```

cookie = serialize_cookie_1('msm', False)
print cookie
print deserialize_cookie_1(cookie)
# fbbb0abb8df1565687a99ca5c3990a4bc37d2c03494118c1574af74fa26cfaf8
# {'user_name': 'msm', 'has_admin': 'False'}
  
```

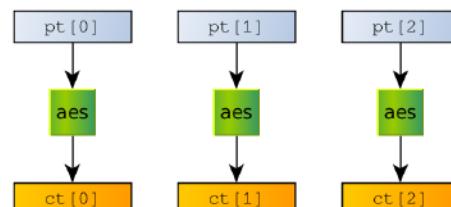
Nasze ciastko jest bezpieczne! Użytkownik otrzymuje tylko zaszyfrowany ciąg fbbb0abb8df1565687a99ca5c3990a4bc37d2c03494118c1574af74fa26cfaf8, którego nie jest w stanie zdeszyfrować (w koncu to AES, prawda?).

No, w sumie... to nie. Sam z siebie AES jest silnym algorytmem, jak na razie nie złamany. Ale po co miałby deszyfrować, skoro i tak wie, co się w ciastku znajduje (każdy zna swoją nazwę użytkownika i wie, czy jest adminem). Popatrzymy więc dokładniej, co się dzieje – w jaki sposób działa szyfrowanie w tym momencie? Domyslnym sposobem działania wielu bibliotek szyfrujących jest, niestety, tak zwany tryb ECB (Electronic Codebook). Działa on w najprostszym możliwym sposób – każdy blok jest szyfrowany niezależnie od siebie, a wyniki sklejane ze sobą:

{ "name": "msm",	2c909d5126184dc25786050b1efecd02
has_admin": fal	435fbfb0461a9da78e5df5236f8add16
se}.....	bffaa1cbe224050677a36d51804cf34b

Rysunek 3. Wiadomość zaszyfrowana w trybie ECB, z podziałem na bloki

Co można przedstawić w postaci grafu:



Rysunek 4. Sposób działania trybu ECB

3. A przynajmniej, jeśli są w nim jakieś błędy, są niezamierzone – autor też jest człowiekiem, a w końcu przewodnią myślą artykułu jest „przy kriptografii łatwo się pomylić”.



odpowiednie funkcje na nową, lepszą wersję:

```
from Crypto import Random
def serialize_cookie_2(name, has_admin):
    raw_cookie = json.dumps({
        'name': name,
        'admin': has_admin
    })
    iv = Random.new().read(AES.block_size)
    cipher = AES.new(SHARED_KEY, AES.MODE_CBC, iv)
    raw = pkcs7_pad(raw_cookie)
    return (iv + cipher.encrypt(raw)).encode('hex')

def deserialize_cookie_2(cookie):
    cookie = cookie.decode('hex')
    iv, cookie = cookie[:16], cookie[16:]
    cipher = AES.new(SHARED_KEY, AES.MODE_CBC, iv)
    raw_cookie = cipher.decrypt(cookie)
    cookie = pkcs7_unpad(raw_cookie)
    return json.loads(cookie)
```

Ponownie podkreślimy – szyfrowanie jest zaimplementowane prawnie. IV jest generowany za pomocą generatora liczb prawdzie losowych, padding wykonano zgodnie ze sztuką itd. Celem artykułu nie jest pokazanie, jak można popełnić błąd podczas implementacji (a można, na naprawdę wiele sposobów). Sprawdźmy więc dla pewności, czy algorytm daje oczekiwane wyniki:

```
cookie_msm = serialize_cookie_2('msm', False)
print cookie_msm
print deserialize_cookie_2(cookie_msm)
# 43fa53c155217afb2f4d481765c53a0de56c
# 3acf52af218c32aa5ccad5155d9a6ff72bb3fe
# b1aa899f8caa5bc174dc90
# {u'name': u'msm', u'admin': False}
```

[ random iv ]	43fa53c155217afb2f4d481765c53a0d
{"name": "msm",	e56c3acf52af218c32aa5ccad5155d9a
"admin": false}.	6ff72bb3feb1aa899f8caa5bc174dc90

Rysunek 8. Wiadomość zaszyfrowana w trybie CBC, z podziałem na bloki

Działa. Programista jest szczęśliwy, szef daje premię, a haker gryzie paznokcie ze złości.

Ale czy na pewno? W końcu dalej użytkownik może dowolnie edytować swoje cookie. Zobaczmy, co się stanie, jeśli trochę namieszamy w zaszyfrowanym ciastku. Stwórzmy najpierw ciastko dla naszego hakera:

```
print serialize_cookie_2('hacker', False)
# 1cd0539e5d370993a62b7c30ffe72f81e6ed642301da
# a252c4e900fab1d95680d4e0f6ee8e336cccd3296801
# 8bbd65cddb8292ead021be75767ad03f9f96e76e83
```

Zmieńmy trzeci bajt w zaszyfrowanym ciastku i zobaczymy, co się stanie podczas deszyfrowania. Logika sugeruje, że powinny wyjść nam kompletne śmieci:

```
cookie = '1cd0519e5d370993a62b7c30ffe72f81e6ed642301da2
52c4e900fab1d95680d4e0f6ee8e336cccd32968018bbd65cddb8292
ead021be75767ad03f9f96e76e83'
# trzeci bajt zmieniony z 51 na 51
print deserialize_cookie_2(cookie)
# {u'admin': False, u'lame': u'hacker'}
```

Zaraz, co się stało? Dlaczego deszyfrowanie powiodło się? Dlatego klucz „name” zmienił się w „lame”? Wiele pytań, czas na odpowiedzi – spójrzmy jeszcze raz na pseudokod przedstawiający spo-

sób działania CBC:

```
ct[0] = aes(pt[0] ^ iv)
ct[1] = aes(pt[1] ^ ct[0])
ct[2] = aes(pt[2] ^ ct[1])
...
```

No tak – na początku szyfrogramu znajduje się IV, a jego modyfikacja wpływa tylko na pierwszy blok. W dodatku wpływa w sposób przewidywalny – konkretnie jeśli zmienimy iv na iv ^ x (iv xorowany z wartością x), po deszyfrowaniu otrzymamy pt ^ x (czyli plaintext xorowany z x). I faktycznie, zobaczymy, co stało się z naszym ciastkiem:

```
plaintext: '{"name": "hacker", "admin": false}'
xor:           2
wynik:      '{"lame": "hacker", "admin": false}'
```

Konkretnie, „n” zamieniło się na „l”, ponieważ znak „n” xorowany z liczbą 2 daje znak „l”:

```
print chr(ord('n') ^ 2) # 'l'
```

Łatwo domyślić się, co można zrobić w ten sposób – skoro dalej możemy modyfikować zawartość ciastka, to wystarczy wykonać kilka obliczeń i można będzie łatwo wejść na chat jako dowolny użytkownik. Spróbujmy zaimplementować atak:

```
cookie = serialize_cookie_2('hacker', False).decode('hex')
change = '\x00' * 10 + xor('haken', 'admin') + '\x00'
cookie = xor(cookie[:16], change) + cookie[16:]
print deserialize_cookie_2(cookie.encode('hex'))
# {u'admin': False, u'name': u'admin'}
```

Udało się – możemy teraz udawać prawie dowolnego użytkownika. Jest tylko jeden problem – dalej nie mamy praw admina, jedynie jego nick. Niestety, zmiana „false” na „true” będzie trochę trudniejsza – wróćmy raz jeszcze do pseudokodu:

```
ct[0] = aes(pt[0] ^ iv)
ct[1] = aes(pt[1] ^ ct[0])
ct[2] = aes(pt[2] ^ ct[1])
```

Jeśli zmienimy coś w ct[1], to owszem, ct[2] zmieni się w taki sposób jak byśmy chcieli, ale podczas deszyfrowania drugi blok zmieni się w losowe (w znaczeniu „nieprzewidywalne”) dane. Na przykład, jeśli zaszyfrujemy {"name": "admin", "admin": false}, otrzymamy ciastko:

```
9fcfa5cd9fc45de713fc0fa02887af5c1a794c1f3330a25710a651c51905
0b99352034310ff53d07bab13597f3ec1f6b591c930f085bf4556b21dfd
e16387d37a
```

Jeśli teraz zmienimy jeden bajt w drugim bloku (czyli ct[0]), zdeszyfruje się ono do:

```
ÆuæA'E+ *!,"atmin": false}
```

Faktycznie, jakiś znak się zmienił („admin” na „atmin”), ale cały pierwszy blok został zniszczony – prawidłowy JSON to już nie jest. Oczywiście, skoro o tym wspominam, jest i sposób na to – wystarczy, że spreparujemy dane tak, żeby śmieci trafiły gdzieś, gdzie są „niegroźne”. Spróbujmy – tym razem nasz haker musi zarejestrować konto o nazwie np. “hacker x”.



Największy wybór profesjonalnego oprogramowania w Polsce !

... w ofercie programy ponad 500 producentów ...



**www.OprogramowanieKomputerowe.pl**



Więcej informacji:

📞 (22) 868 40 42

✉️ [sales@tts.com.pl](mailto:sales@tts.com.pl)

Sprzedaż



Dystrybucja



Import na zamówienie

TTS Company Sp. z o.o.

ul. Domaniewska 44A

02-672 Warszawa

[www.tts.com.pl](http://www.tts.com.pl)

[ random iv ]	593361eeafe757e421836c0a01c3d6ea
{"name": "hacker	0f53fdc786a629247ef60cf264b271af
x	f5b035ebd307311ed6fdffbf8cb7e246
, "admin": false	28092ff40ef6f05620880acedd1317a7
.....	ad98aa5050c0c81e1043300c24bde345

Rysunek 9. Zaszyfrowane ciastko dla nowego użytkownika

Co nam to dało? Że teraz trzeci blok (ten zawierający ciąg spacji oraz „x”) może w zasadzie zawierać prawie dowolne dane – jedyne ograniczenie jest takie, że bajty muszą poprawnie zdekodować się jako utf-8 oraz nie zawierać znaku cudzysłowa („) – żeby stanowiły poprawny JSON.

Ten algorytm można zapisać np. tak:

```
from random import randint
while True:
    try:
        cookie = serialize_cookie_2('hacker' + ' ', False)
        cookie = [ord(c) for c in cookie.decode('hex')]
        cookie[44] ^= ord('f') ^ ord(' ') # zmiana znaku "f" na " "
        cookie[45] ^= ord('a') ^ ord('t') # zmiana znaku "a" na "t"
        cookie[46] ^= ord('l') ^ ord('r') # zmiana znaku "l" na "r"
        cookie[47] ^= ord('s') ^ ord('u') # zmiana znaku "s" na "u"
        cookie = ''.join(chr(c) for c in cookie).encode('hex')
        deserialize_cookie_2(cookie)
    except:
        pass # deserializacja nie powiodła się
    else:
        print cookie # udało się
        print deserialize_cookie_2(cookie)
        break
```

Prawdziwy atakujący oczywiście nie dysponowałby funkcją `deserialize_cookie` bezpośrednio – musiałby sprawdzać, czy atak się udał, wchodząc na naszą stronę z ustawionym ciastkiem i obserwując, czy występuje błąd.

Po wielu próbach atak się udaje – otrzymujemy ciastko, które poprawnie dekoduje się:

```
cookie = "bfe7c87cb5df56f225cef6d8d4a1ad61cbe017e14b
2d767d3b31772148b593dddb34c1c042b4a781c0fa37b4ff490c
b40d78cc15644a2607af1387ed8fc63af2a57c08e03a2fe35417
655a39d61bc3c4"
print deserialize_cookie_2(cookie)
# {'admin': True, 'name': 'hacker8+!E.2C1a)*>>#BJ'}
```

Nazwa użytkownika `hacker8+!E.2C1a)*>>#BJ` nie jest może idealna, ale też nie przeszkadza – ważne jest, że mamy prawa administratora.

I ponownie, nasza metoda zawiodła. Jak to się stało, że AES/CBC, metoda wykorzystywana przez wiele silnych protokołów kryptograficznych, została przez nas właśnie bez większego problemu „złamana”? Do tego pytania wróćmy pod koniec rozdziału.

Swoją drogą – atak, który właśnie wykonaliśmy, jest na tyle popularny, że ma nawet swoją nazwę. Konkretnie mówi się o „bit flipping” albo „byte flipping”<sup>5,6</sup>.

## Złam to sam: CBC

Jeśli komuś tryb CBC niestraszny, może zmierzyć się z praktycznym zadaniem, wchodząc na stronę <https://var.tailcall.net/cbc>.

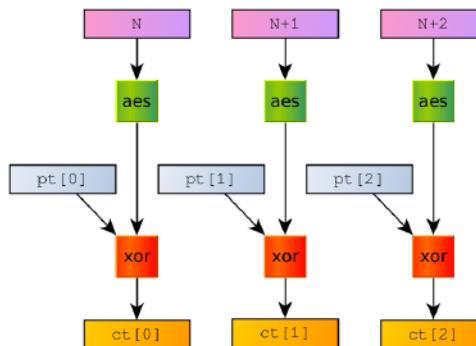
5. Teoretycznie „bit-flipping” dokładniej opisuje charakter ataku, ale termin ten ma już wiele innych znaczeń, więc prywatnie autorzy preferują jednoznaczny termin „byte flipping”.

6. [https://en.wikipedia.org/wiki/Bit-flipping\\_attack](https://en.wikipedia.org/wiki/Bit-flipping_attack)

## Szyfrowanie 3: CTR

A na razie nasz nieszczęśliwy programista musi ponownie naprawiać kod. Tryby ECB i CBC go zawiodły – ale wyczytał w sieci, że jest jeszcze jeden, silniejszy: tryb CTR<sup>7</sup>.

Działa on tak:



Rysunek 10. Sposób działania trybu CTR

W pseudokodzie:

```
ct[0] = aes(n+0) ^ pt[0]
ct[1] = aes(n+1) ^ pt[1]
ct[2] = aes(n+2) ^ pt[2]
...
...
```

Co tu się dzieje? Nie szyfrujemy tu plaintextu, a jakiś losowo wybraną liczbę N (tzw. nonce), następnie N+1, N+2, N+3 itd. Wynik tego szyfrowania (zwany keystream) tylko xorujemy z plaintextem, żeby otrzymać ciphertext.

Dlaczego w ogóle coś takiego robić? Okazuje się, że szyfrowanie w ten sposób jest równie bezpieczne jak CBC (a nawet minimalnie bardziej), a ma nad nim kilka zalet:

- » Można zównoleglić szyfrowanie i deszyfrowanie, co może mieć znaczenie przy współczesnych wieloprocesorowych maszynach.
- » Padding staje się niepotrzebny – długość plaintextu nie musi być wielokrotnością długości bloku. Formalnie rzecz ujmując, zmieniamy szyfr blokowy w szyfr strumieniowy.

Czy ma jakieś wady? Głównie jedną – jest nowszy, gorzej wspierany w bibliotekach i trudniej go dobrze użyć. Nie brzmi to zbyt problematycznie, ale w praktyce błędy bezpieczeństwa w aplikacjach spowodowane złym użyciem trybu CTR są dużo częstsze niż w przypadku CBC. Wynika to z tego, że nonce (skrót od *number used once*) naprawdę musi być unikalne. W momencie, gdy nonce powtórzy się chociaż raz, całe bezpieczeństwo szyfrowania spada do zera – atakujący może łatwo odzyskać keystream, na przykład xorując swój ciphertext ze swoim plaintextem (jeśli go zna), albo wykonać jeden z wielu podobnych ataków.

Zaimplementujmy więc tryb CTR:

```
def int_to_16bytes(n):
    return '{:032x}'.format(n)[-32:].decode('hex')

def counter(nonce):
```

7. Faktycznie, tryb CTR jest w pewien sposób matematycznie silniejszy niż tryb CBC. Jednak różnica w poziomie bezpieczeństwa jest w praktyce pomijalna (rozbiąja się o teoretyczne pojęcie *różnorównialność od losowych danych*, przy ponad  $2^{64}$  bajtach zaszyfrowanych jednym kluczem), za to dużo łatwiej krytycznie pomylić się w implementacji (np. wielokrotnie używając jednego nonce), a konsekwencje pomyłki są gorsze. Z tego powodu trudno wskazać jeden z tych dwóch trybów jako obiektywnie bezpieczniejszy.

```

return Counter.new(128, initial_value=nonce)

from Crypto.Random import random
from Crypto.Util import Counter

def serialize_cookie_3(name, has_admin):
    raw_cookie = json.dumps({
        'name': name,
        'admin': has_admin
    })
    raw = pkcs7_pad(raw_cookie)
    nonce = random.randint(0, 2**128)
    cipher = AES.new(SHARED_KEY, AES.MODE_CTR,
                     counter=counter(nonce))
    return (int_to_16bytes(nonce) + cipher.encrypt(raw)).encode('hex')

def deserialize_cookie_3(cookie):
    nonce, cookie = int(cookie[:32], 16), cookie[32:].decode('hex')
    cipher = AES.new(SHARED_KEY, AES.MODE_CTR,
                     counter=counter(nonce))
    raw_cookie = cipher.decrypt(cookie)
    cookie = pkcs7_unpad(raw_cookie)
    return json.loads(cookie)

```

Jak widać, mieliśmy rację, że w CTR łatwo się pomylić – trzeba było napisać całkiem sporo kodu jak na szyfrowanie standardową metodą. A potencjalnie w każdej linijce kodu czai się błąd...<sup>8</sup>

Upewnijmy się, że przynajmniej funkcje działają zgodnie z naszymi oczekiwaniami:

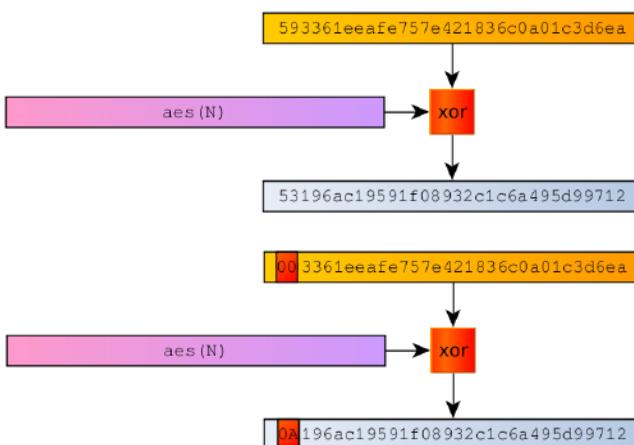
```

cookie = serialize_cookie_3('msm', False)
print cookie
print deserialize_cookie_3(cookie)
# ab56feaf4a1a2762c24c1ee7e454041315051507ca8f5c7717e
# 516addde3d4cfb8a3d01e778961105e67a2e56af458
# {u'admin': False, u'name': u'admin'}

```

Już dwa razy udało nam się „złamać” program wykorzystujący jakąś metodę szyfrowania. Czy i tym razem atak również będzie możliwy? Jeśli tak, jak trudny będzie? Zachęcam wszystkich do chwili namysłu – co stanie się, jeśli zmodyfikujemy fragment ciphertektu?

No to popatrzmy:



Rysunek 11. Efekt modyfikacji szyfrrogramu zaszyfrowanego w trybie CTR

No tak. Ostatecznie szyfrowanie i deszyfrowanie polega tylko na xorowaniu ze strumieniem klucza, więc każda zmiana jednego oktetu w szyfrogramie spowoduje deterministyczną i łatwą do przewidzenia zmianę jednego oktetu w deszyfrowanym plainteksie. Korzystając z tego faktu, jeśli wiemy, jakie dane są zaszyfrowane, możemy je

8. Swoją drogą użycie wartości domyślnych obiektu Counter dostarczanego przez PyCrypto często może łatwo doprowadzić do dziurawego kodu – parametr `initial_value` ma domyślną wartość 1 i łatwo używając go w ten sposób, spowodować powtórzenie się nonce i problemy z bezpieczeństwem. Jest to dobry przykład na to, jak trzeba uważać nawet korzystając z bibliotek.

trywialnie zmodyfikować na cokolwiek tylko chcemy. Atak jest bardzo prosty, ale dla porządku zaimplementujmy go:

```

print deserialize_cookie_3(cookie)
# {u'admin': False, u'name': u'msm'}
cookie = cookie.decode('hex') # get raw bytes

# (...32 bytes...) <"admin": false>.
old = '\x00' * 32 + ".....false...".replace('.', '\x00')
new = '\x00' * 32 + ".....true ...".replace('.', '\x00')

mod = xor(old, new) # remove old bits, add new ones
new_cookie = xor(cookie, mod) # apply modifier
new_cookie = new_cookie.encode('hex') # go back to hex
print deserialize_cookie_3(new_cookie)
# {u'admin': True, u'name': u'msm'}

```

W tym momencie nasz programista prawdopodobnie kompletnie się załamuje. Kolejna metoda szyfrowania nie sprawdziła się w programie. Dlaczego?

## Złam to sam: CTR

Zjadłeś zęby na kryptografii? Tryby ECB i CBC łamiesz na kartce? Zostało ostatnie wyzwanie w temacie: <https://var.tailcall.net/ctr>.

## Encryption is not authentication

No właśnie, w końcu czas na odpowiedź – dlaczego? Często powtarzane motto w kryptografii brzmi: *Encryption is not authentication*, czyli „szyfrowanie nie służy do uwierzytelniania”. Problemem tutaj nie były błędy w implementacji – bo tych nie było. Zrobiliśmy coś gorszego – użyliśmy szyfrowania do celu, do którego nie jest przeznaczone.

Celem szyfrowania jest ukrycie treści wiadomości przed osobami, dla których nie jest przeznaczona. Czy faktycznie mieliśmy coś do ukrycia w tym przypadku? Oczywiście nie – zarówno nick, jak i bycie administratorem to publiczne atrybuty każdego użytkownika – szyfrowanie tutaj jest w zasadzie zupełnie niepotrzebne. Owszem, wydaje się, że jeśli użytkownik dostanie zaszyfrowane tzw. „losowe śmieci”, nie będzie w stanie wpływać na nie w żaden sposób. Jak, mam nadzieję, udowodniliśmy, jest to zupełnie błędne założenie. Szyfrowanie nie służy do uwierzytelniania!

Zamiast tego powinniśmy popatrzeć na tak zwane podpisy... Ale o tym przy innej okazji.

## ZŁE SZYFROWANIE TO NIE SZYFROWANIE

Na razie powrócimy do naszych szyfrów i sprawdzimy, czy przynajmniej spełniają swoje podstawowe zadanie – czyli uniemożliwiają odczytanie danych osobom postronnym.

W naszych poprzednich przykładach zakładaliśmy, że wszystkie dane są jawne. Na potrzeby tej części wyobraźmy sobie, że z jakiegoś powodu użytkownik nie powinien móc odczytać ze swojego ciastka, czy jest administratorem (owszem, jest to zupełnie abstrakcyjna sytuacja – robimy tak dla uproszczenia. Ale łatwo wyobrazić sobie, że np. w ciastku zaszyty jest jakiś sekret potrzebny do komunikacji między serwerami, którego użytkownik nie powinien móc poznać).

Utrzymamy ten sam model ataku co w poprzedniej części – czyli użytkownik może dowolnie zmieniać swój szyfrrogram oraz obserwować odpowiedzi, jakie dostaje od serwera po modyfikacjach.

## Szyfrowanie 1: ECB

Zaczniemy ponownie od trybu ECB. Internet jest pełen ostrzeżeń, że ten tryb jest niebezpieczny, ale w zasadzie dlaczego?

Odpowiedź jest prosta – wyobraźmy sobie, że mamy zaszyfrowane ciastka trzech użytkowników:

- » adm, o którym wiemy, że jest administratorem,
- » hax, nasz użytkownik – wiemy, że nie jest administratorem,
- » msm, użytkownik, o którym nie wiemy, czy jest administratorem, i chcemy się dowiedzieć.

Gdyby szyfrowanie faktycznie dobrze zabezpieczało treść wiadomości, dowiedzenie się czegokolwiek o „msm” powinno być niemożliwe.

Załóżmy, że mamy następujące ciastka:

- adm: d2919b3d93e5797be6ac074f3b3bb71bd61cea54e830fddc6cd32ee775d4ef3b
- hax: 4d5a8ba882d5c302a806bb6853550fd96067db7e3c616e49839f839d123f84e7
- msm: 2c909d5126184dc25786050b1efecd02d61cea54e830fd dc6cd32ee775d4ef3b

Przyjrzyjmy się dokładniej zaszyfrowanym wartościom:

{"name": "adm",	d2919b3d93e5797be6ac074f3b3bb71b
"admin": true}..	d61cea54e830fddc6cd32ee775d4ef3b
{"name": "hax",	4d5a8ba882d5c302a806bb6853550fd9
"admin": false}..	6067db7e3c616e49839f839d123f84e7
{"name": "msm",	2c909d5126184dc25786050b1efecd02
"admin": ???????	d61cea54e830fddc6cd32ee775d4ef3b

Rysunek 12. Porównanie zaszyfrowanych wiadomości

Czy można ustalić grupę, do której przynależy użytkownik „msm”? Po chwil wpatrywania się w rysunek 12 można zauważyc, że owszem, można – drugi blok ciphertext jest identyczny dla użytkowników „msm” i „adm”, co oznacza, że drugi blok plaintext również musi być identyczny.

Oznacza to, że tryb ECB zawiódł nas nawet w swoim podstawowym zadaniu – ukrywaniu treści wiadomości przed postronnymi<sup>9</sup>.

Byla nawet gorzej – jeśli atakujący jest w stanie szyfrować dowolne dane i kontroluje jakiś ich fragment (np. swój nick), istnieje atak, który pozwoli mu również odszyfrować dowolne dane za kontrolowanym kawałkiem (kosztem 256 zapytań do serwera na każdy bajt szyfrogramu).



Rysunek 13. Atakujący kontroluje swój nick i chciałby przeczytać „tajne informacje”

W takiej sytuacji należy tak wybrać wstawiane dane, żeby ostatni blok zaszyfrowanych danych zawierał tylko jeden bajt faktycznych danych oraz 15 bajtów paddingu.

## ECB, CBC, CTR, WTF?

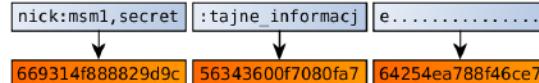
Dlaczego niezrozumiałe trzyznakowe nazwy? Dlaczego nie można było nazwać trybów szyfrowania np. „niebezpieczny”, „standardowy” i „licznikowy”? W zasadzie nie wiemy – być może to tajny plan kryptografów na zapewnienie sobie tak zwanego *job security*. Sprawa ma się jeszcze gorzej, bo trybów jest więcej:

- Tryb OFB, podobny do CTR, ale zamiast aes(n), aes(n+1), aes(n+2) używany jest iv, aes(iv), aes(aes(iv)). Jego niewielką zaletą jest to, że deszyfrowanie działa dokładnie tak samo jak szyfrowanie (więc można zaoszczędzić na bramkach w hardware), ale ma dużo gorsze parametry bezpieczeństwa niż tryb CTR i z tego powodu nie jest polecany
- Tryb OCB, relatywnie nowe odkrycie, które zapewnia jednocześnie szyfrowanie, uwierzytelnianie i wysoką wydajność. Oznacza to, że nie trzeba dodatkowo podpisywać wiadomości. Ten schemat to jeden z nowoczesnych trybów gwarantujących tak zwane *authenticated encryption*<sup>1</sup>. Dlaczego więc nie jest używany powszechnie? Niestety – ciąży na nim patent<sup>2</sup>, co historycznie skutecznie zniechęcało potencjalnych użytkowników i twórców standardów.
- Tryby GCM, CCM, EAX będące próbami stworzenia wolnej alternatywy dla trybu OCB. Niestety, wszystkie mają jakieś ograniczenia lub problemy – z tego powodu trudno wyróżnić jeden z nich (najpopularniejszy obecnie jest GCM). Mimo wszystko, jeśli zależy nam na uwierzytelnionym szyfrowaniu, każdy z nich ma przewagę nad szyfrowaniem i podpisywaniem wiadomości osobno. Wspólną wadą jest średnie wsparcie ze strony bibliotek i frameworków, szczególnie tych starszych.

W praktyce nie trzeba się tym za bardzo przejmować – tryb CBC w zupełności wystarcza do szyfrowania danych, a jeśli potrzebujemy dodatkowo uwierzytelnić dane, to możemy albo dodatkowo podpisać dane, albo użyć trybu GCM.

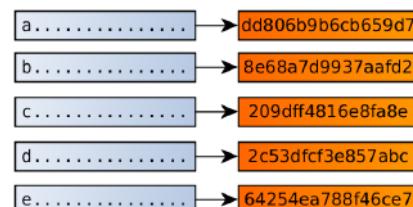
1. [https://en.wikipedia.org/wiki/Authenticated\\_encryption](https://en.wikipedia.org/wiki/Authenticated_encryption)

2. [https://en.wikipedia.org/wiki/OCB\\_mode](https://en.wikipedia.org/wiki/OCB_mode)



Rysunek 14. Atakujący tak dopasował ostatni blok, że zawiera tylko jeden bajt danych

Można wtedy ten jeden bajt znaleźć za pomocą metody siłowej (skoro możemy szyfrować dowolne dane, możemy spróbować wszystkich jednobajtowych wiadomości). Kiedy to się uda, należy wygenerować kolejny szyfrogram, żeby w ostatnim bloku były dwa bajty danych oraz 14 bajtów paddingu. Można wtedy znowu zastosować metodę siłową – jako że ostatni bajt już znamy, wystarczy kolejne 255 zapytań. Tę procedurę należy powtarzać, aż poznamy wszystkie bajty całej wiadomości.



Rysunek 15. Atakujący szyfruje jednobajtowe bloki, aż uda mu się trafić na pasujący

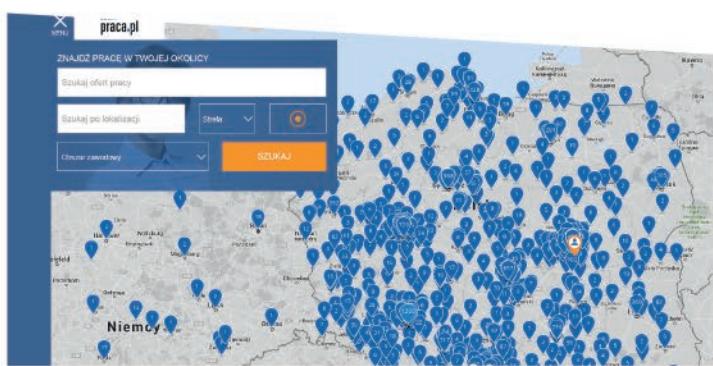
9. Czasami w niektórych materiałach dedykowanych dla programistów pojawiają się zdania w rodzaju „trybu ECB nie powinno się nigdy używać”. Nie zupełnie tak jest. To bardziej uproszczenie, oparte na założeniu, że lepiej być „za bardzo zabezpieczonym” niż „za mało zabezpieczonym”. W szczególności, kiedy wiemy, że zawsze będziemy szyfrować dokładnie jeden blok, tryb ECB nie ma przewagi nad innymi.

Jako że atak ten jest dość rzadko wykorzystywany w praktyce (tryb ECB, na szczęście, prawie wymarł), podarujemy sobie implementację. Zamiast tego przejdźmy do ciekawszego ataku.



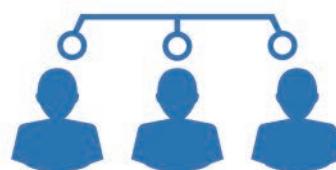
Chcesz dobrze zarobić?

Na Praca.pl codziennie znajdziesz ponad 3 000 ofert pracy z obszaru IT i nowe technologie



Znajdź pracę w Twojej okolicy

Lokalna.praca.pl



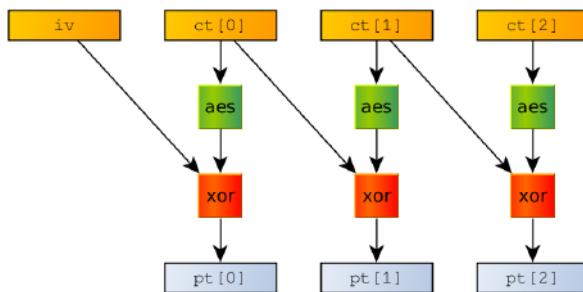
Poleć znajomego do pracy  
i zgarnij 1 000 zł

[Praca.pl/rekomendacje.html](http://Praca.pl/rekomendacje.html)

## Szyfrowanie 2: CBC

Szyfrowanie w trybie CBC zepsuć równie łatwo, a atak ma podobnie (albo nawet bardziej) drastyczne skutki.

Widzieliśmy wcześniej, jak działa szyfrowanie w trybie CBC, ale teraz ważniejsze będzie dla nas deszyfrowanie:



Rysunek 16. Deszyfrowanie w trybie CBC

Ta metoda szyfrowania „matematycznie” jest silna, ale ma jedną pułapkę. Popatrzmy na nasz kod deszyfrujący:

```

def deserialize_cookie_2(cookie):
    cookie = cookie.decode('hex')
    iv, cookie = cookie[:16], cookie[16:]
    cipher = AES.new(SHARED_KEY, AES.MODE_CBC, iv)
    raw_cookie = cipher.decrypt(cookie)
    cookie = pkcs7_unpad(raw_cookie) # throws on invalid padding
    return json.loads(cookie) # throws on invalid json
  
```

W przypadku aplikacji webowych błędy często idą bezpośrednio do użytkownika. Jeśli atakujący dowie się, jaki wyjątek został rzucony, będzie w stanie rozpoznać, czy deszyfrowanie nie udało się z powodu błędego paddingu (AssertionError w pkcs7\_unpad), czy z powodu nieprawidłowego JSON (ValueError w json.loads). Metoda pozwalająca rozpoznawać, czy padding dowolnego szyfrogramu jest poprawny, nazywana jest w kryptografii *Padding Oracle*. Nie brzmi groźnie, prawda?

```

# w praktyce, wyrocznia jest oczywiście zazwyczaj zdalny serwer
# np. strona deszyfrująca ciastko
# dla celów demonstracyjnych, "lokalna wyrocznia"
def padding_oracle(ciphertext):
    try:
        deserialize_cookie_2(ciphertext.encode('hex'))
    except AssertionError:
        return False # invalid padding
    except ValueError:
        return True # invalid json
    return True # all ok
  
```

Niestety, w praktyce to duży problem. Jeśli popatrzymy jeszcze raz na rysunek 16 (albo przypomnijmy sobie poprzednie ataki), widzimy, że atakujący może łatwo wpływać na zdeszyfrowane dane, modyfikując bloki szyfrogramu. W szczególności atakujący może spróbować wszystkich 256 możliwości na ostatni bajt, aż uda mu się tak trafić, że ostatni zdeszyfrowany bajt będzie równy 0x01. Wtedy szyfrogram będzie kończył się jednym bajtem równym 0x1 i ten ostatni bajt, uznany przez algorytm za padding, zostanie usunięty:

```

def attack_1(prev_block, last_block):
    for i in range(256):
        new_block = prev_block[:15] + chr(i)
        if padding_oracle(new_block + last_block):
            print 'padding is correct for prev[15] =', i
            return
    # padding is correct for prev[15] = 221
  
```

I znowu – co z tego wynika? Dochodzimy tu do „punktu kulminacyjnego” naszego ataku. Warto spojrzeć jeszcze raz na Rysunek 16 pokazujący, jak działa deszyfrowanie.

Ustalmy terminologię:

- » *prev* to oryginalna wartość przedostatniego bloku,
- » *intermediate* to wartość ostatniego bloku po deszyfrowaniu, ale przed operacją xor,
- » *new* to zmodyfikowana przez nas wartość ostatniego bloku,
- » *plain* to plaintext (odszyfrowany blok, który chcemy odzyskać).

Wiemy, że:

- » znamy ostatni bajt *new*, który deszyfrowaliśmy (to wartość *i*, którą wypisaliśmy w programie),
- » Wiemy,<sup>10</sup> że ostatni bajt *plain* dla naszego zmodyfikowanego *prev* jest równy 1,
- » znamy oryginalną wartość *prev* (zanim ją zmodyfikowaliśmy).

Zapiszmy to jako równanie:

```

prev[15] ^ intermediate[15] = plain[15]
new[15] ^ intermediate[15] = 1

# z czego wynika:
intermediate[15] = new[15] ^ 1
plain[15] = prev[15] ^ intermediate[15]
  
```

Matematyka nie kłamie – za pomocą dwóch operacji xor możemy odzyskać w ten sposób ostatni bajt plaintextu (czyli inaczej odszyfrować ostatni bajt).

Przetestujmy:

```

def attack_2(prev, last):
    intermediate = [0] * 16
    plain = [0] * 16
    for i in range(256):
        new = prev[:15] + chr(i)
        if i != ord(prev[15]) and padding_oracle(new + last):
            intermediate[15] = ord(new[15]) ^ 1
            plain[15] = ord(prev[15]) ^ intermediate[15]
            print 'last byte of plaintext is', plain[15]
            return

# last byte of plaintext is 4
  
```

Czyli udało nam się zdeszyfrować ostatni bajt wiadomości, do wartości „4”. Wygląda to na popawną wartość (jako że jest stosowany standard PKCS#7, oznacza to, że cztery ostatnie bajty wiadomości to padding).

Faktycznie, coś zdeszyfrowaliśmy, ale jeszcze nie wygląda to drastycznie. Przełom następuje, kiedy zorientujemy się, że możemy rozszerzyć ten atak – i pójść dalej. Skoro znamy ostatni bajt bloku *intermediate*, możemy tak ustawić ostatni bajt bloku *new*, żeby plaintext kończył się na 0x2 – i zgadywać przedostatni bajt:

```

def attack_3(prev, last):
    intermediate = [0] * 16
    plain = [0] * 16
    new = list(ord(c) for c in prev)
    for i in range(256):
        new[15] = i
        if i != ord(prev[15]) and padding_oracle(to_str(new) + last):
            intermediate[15] = new[15] ^ 1
            plain[15] = ord(prev[15]) ^ intermediate[15]
            print 'plaintext[15] is', plain[15]
            break
  
```

<sup>10</sup>. A dużym prawdopodobieństwem jest ryzyko, że tekst zdeszyfruje się do czegoś kończącego się np. na dwa bajty o wartości 0x2.

```

for i in range(256):
    new[15] = 2 ^ intermediate[15]
    new[14] = i
    if padding_oracle(to_str(new) + last):
        intermediate[14] = new[14] ^ 2
        plain[14] = ord(prev[14]) ^ intermediate[14]
        print 'plaintext[14] is ', plain[14]
        break
# plaintext[15] is 4
# plaintext[14] is 4

```

Wygląda dobrze.

Jak łatwo się domyśleć, atak generalizuje się też na trzy bajty, i na cztery, i na więcej. Można by iść dalej tą drogą i skopiować pętlę 16 razy, ale lepiej użyć trochę bardziej zaawansowanych technik programistycznych i zapisać kod w postaci jednej, dość krótkiej pętli:

```

def attack_4(prev, last):
    intermediate = [0] * 16
    plain = [0] * 16
    new = list(ord(c) for c in prev)
    result = ''
    for j in range(16)[::-1]: # from 15 to 0
        for k in range(j+1, 16):
            new[k] = intermediate[k] ^ (16-j)

    for i in range(256):
        if j == 15 and i == ord(prev[j]):
            continue

        new[j] = i
        if padding_oracle(to_str(new) + last):
            intermediate[j] = new[j] ^ (16 - j)
            plain[j] = ord(prev[j]) ^ intermediate[j]
            print 'plaintext[{}]\ is {}'.format(j, plain[j])
            result = chr(plain[j]) + result

```

```

                break
            print 'plaintext:', result

# plaintext[15] is 4
# plaintext[14] is 4
# plaintext[13] is 4
# plaintext[12] is 4
# plaintext[11] is 125
# plaintext[10] is 101
# plaintext[9] is 117
# plaintext[8] is 114
# plaintext[7] is 116
# plaintext[6] is 32
# plaintext[5] is 58
# plaintext[4] is 34
# plaintext[3] is 110
# plaintext[2] is 105
# plaintext[1] is 109
# plaintext[0] is 100
# plaintext: dmin": true}

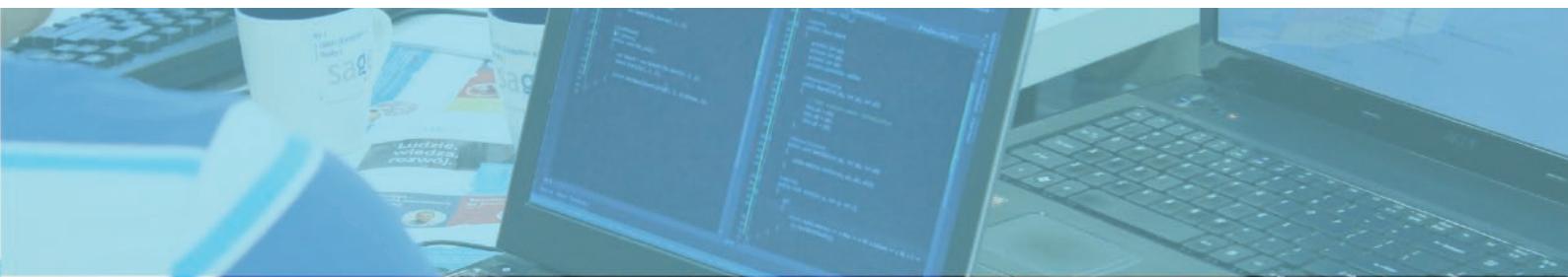
```

Gotowe – możemy odszyfrować dowolne dane, mając do dyspozycji jedynie wyciągnięcie. Takie ataki to jeden z powodów, dla których kryptografia była „straszna”, a implementowanie powiązanych rzeczy odradzane. W końcu w samym trybie CBC nie ma niczego, co sugerowałoby, że zdradzanie informacji o poprawności paddingu prowadzi do tak krytycznych problemów. Szczególnie że zazwyczaj podczas tworzenia oprogramowania priorytety są wręcz odwrotne – dostarczanie dokładnych informacji o błędzie jest uważane za dobrą praktykę. Twórcy API kryptograficznych zauważili w końcu ten problem i w obecnych wysokopoziomowych bibliotekach (np. WebCrypto API<sup>11 12</sup>) ukrywane są szczegółowe informacje o przyczynie błędu.

11. <https://szukaj.programistamag.pl/uuid/c6976487a4749c6e8620aa6db5540dd9fde372e5>

12. <https://www.w3.org/TR/WebCryptoAPI/>

reklama



Szkolenie dla Ciebie lub Twojego zespołu

## Praktyczne wykorzystanie blockchain na przykładzie Ethereum

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages ważnej przy zamówieniach złożonych do końca października 2017 r.  
Hasło: PROGRAMISTAMAG

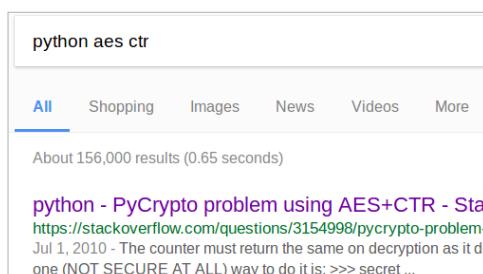
3 dni	24h	zdalnie lub stacjonarnie
trenerzy praktycy	różne lokalizacje	projekty indywidualne

Metoda *Padding Oracle* została odkryta w 2002 roku i spowodowała małe trzęsienie ziemi – była na nią wówczas podatna większość frameworków webowych, w tym Ruby on Rails, JavaServer Faces oraz ASP.NET. Do dzisiaj zdarza się, że w popularnych i stosowanych powszechnie protokołach ktoś znajdzie błąd tego typu – np. atak Lucky Thirteen na TLS<sup>13</sup> z 2013 roku, który łączy opisany tu atak z atakiem czasowym (mierzeniem czasu, jaki zajmuje zapytanie)<sup>14</sup>.

## Szyfrowanie 3: CTR

Na koniec, ku przestrodze: smutno – straszna historia.

Tak jak wspominaliśmy, tryb CTR jest w różnym stopniu wspierany przez biblioteki. Z tego powodu użycie go zazwyczaj nie jest trywialne. Programiści często w takim przypadku szukają pomocy w Internecie. Pierwszy wynik po zapytaniu „Python AES CTR” w moim przypadku prowadzi do StackOverflow<sup>15</sup>:

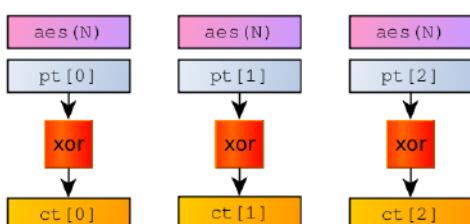


Rysunek 17. Pierwszy wynik po zapytaniu o „szyfrowanie AES CTR w pythonie”

Gdzie najwyżej punktowaną odpowiedzią jest następujący przykład od użytkownika będącego TOP 15 pod względem liczby punktów reputacji<sup>16</sup>:

```
>>> secret = os.urandom(16)
>>> crypto = AES.new(os.urandom(32), AES.MODE_CTR,
counter=lambda: secret)
>>> encrypted = crypto.encrypt("aaaaaaaaaaaaaaaaaa")
>>> print crypto.decrypt(encrypted)
aaaaaaaaaaaaaaaaaa
```

Ponownie zadajmy retoryczne pytanie – czy jest tu jakiś problem? Jak właściwie działa parametr counter? Jest to funkcja, która wywoływana jest dla każdego bloku i za każdym razem powinna zwrócić kolejną wartość licznika (autor jest tego świadomy). Niestety, w tym przypadku przekazana została funkcja, która zawsze zwraca tę samą wartość (!), co prowadzi do pewnych komplikacji:



Rysunek 17. Deszyfrowanie w trybie CTR, przy stałym nonce

13. [https://en.wikipedia.org/wiki/Lucky\\_Thirteen\\_attack](https://en.wikipedia.org/wiki/Lucky_Thirteen_attack)

14. Dla uczciwości trzeba dodać, że ten atak wymaga bardzo dokładnego pomiaru czasu zapytania, przez co jest mało praktyczny poza siedlami lokalnymi (albo środowiskami laboratoryjnymi).

15. <https://stackoverflow.com/a/3155175>

16. Ta odpowiedź została napisana w 2010 roku. Dwa tygodnie temu dokładnie ten kod został wykorzystany w konkursie SHA2017 CTF jako najprostszego (!) zadania z kryptografią. Dopiero wtedy ktoś zlitował się nad innymi i dopisał do odpowiedzi „(NOT SECURE AT ALL)”, (patrz Rysunek 17).

Czyli całe „szyfrowanie AEsem” sprowadza się do prostej operacji xor ze stałym, 16-bajtowym kluczem! Jest to bardzo dobrze znana kryptografom konstrukcja, nie dostarczająca w zasadzie żadnego bezpieczeństwa.

W większości przypadków można ją złamać nawet ręcznie (używając tylko ołówka i kartki papieru), w najgorszym przypadku pozwodzi sobie z nią komputer w milisekundzie za pomocą prostej analizy statystycznej – takie ataki jednak pominiemy ze względu na ograniczone miejsce. Zobaczmy jednak przynajmniej, jak trywialny jest atak, jeśli znamy choć jeden blok ciphertextu. To bardzo realistyczna sytuacja – na przykład, jeśli zaszyfrowane dane to plik .png, pierwsze 16 bajtów będzie zawsze takie samo. Wiemy, że:

```
ct_0 = pt_0 ^ secret
ct_1 = pt_1 ^ secret
ct_2 = pt_2 ^ secret
# ...
ct_n = pt_n ^ secret
```

Teraz, jeśli wiemy, że zaszyfrowano obraz .png, możemy podstawić znaną wartość pod pt\_0 i obliczyć secret:

```
# constant header from .png file
pt_0 = '\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR'
secret = ct_0 ^ pt_0
```

W ten sposób dysponujemy wystarczającą ilością danych, żeby odzyskać cały ciphertext:

```
pt_1 = ct_1 ^ secret
pt_2 = ct_2 ^ secret
# ...
ct_n = pt_n ^ secret
```

Jeśli ktoś skopiował kod ze strony Stack Overflow, to cały „atak” na jego program to kilka do kilkunastu linijek kodu.

Ale to przecież nie problem – przecież nikt<sup>17</sup> nigdy<sup>18</sup> nie<sup>19</sup> kopiuje<sup>20</sup> bezmyślnie<sup>21</sup> kodu ze Stack Overflow...

## ZAKOŃCZENIE

W artykule zaprezentowano kilka stosunkowo prostych ataków, które można przeprowadzić wobec prawidłowo zaimplementowanych, ale błędnie użytych algorytmów szyfrowania. Nie należy zapominać, że prawo do wypowiedzi w Internecie posiada każdy. Skutkuje to tym, że osoby niemające wiedzy z zakresu kryptografii udzielają związanych z nią porad na popularnych portalach. Przed wykorzystaniem jakiegokolwiek prymitywu kryptograficznego konieczne jest osiągnięcie pełnego zrozumienia sposobu jego działania, a to można zrobić, wyłącznie posługując się stosowną, zweryfikowaną literaturą.

17. <https://goo.gl/smRNCE> (W krytycznej funkcji projektu nazywającego się Confidentiality as a Service)

18. <https://goo.gl/y9Cleg> (Szyfrowanie w off the wire crypto, otr with pgp + aes-ctr 256 instead of diffie hellman)

19. <https://goo.gl/w2dEE1> (A rebuild of marionette, encrypted proxy that simulates general webtraffic. (...) Provides strong end to end encryption with PGP + AES-CTR + HMACSHA256)

20. <https://goo.gl/oHvjA> (Zabezpieczanie plików w GUI Text Editor with Crypto operations in Python)

21. <https://goo.gl/EAkJ5b> (Biblioteka do szyfrowania)

## JAROSŁAW JEDYNAK

msm@tailcall.net

Na co dzień broni bezpieczeństwa polskiego Internetu, pracując jako Security Engineer w Cert Polska. Do jego specjalności należą niskopoziomowa analiza malware oraz inżynieria wstępna protokołów sieciowych złożliwego oprogramowania. W wolnym czasie programuje, pomaga w administracji serwisem 4programmers.net i natłogowo gra w konkursy CTF.



KEYLIGHT

WWW.KEYLIGHT.COM.PL

# Kurs angielskiego dla programistów.

## Lekcja 8

Przedstawiam ósmą lekcję minikursu angielskiego dla programistów. Tym razem tematem przewodnim są serwery bazy danych. Zachęcam do wielokrotnego wykonywania ćwiczeń, aby dobrze utrwalić sobie przyswojony materiał. Rozwiązań do ćwiczeń zamieszczono na stronie internetowej, której adres podano na dole artykułu.

### SQLITE

SQLite is an **in-process library** that implements a **self-contained, serverless, zero-configuration, transactional SQL database engine**. The code for SQLite is in the **public domain** and is thus free for use for any purpose, commercial or private. SQLite is the **most widely deployed database** in the world with more **applications** than we can count.

SQLite is an **embedded** SQL database engine. Unlike most other SQL databases, SQLite does not have a **separate server process**. SQLite reads and writes directly to **ordinary disk files**. A complete SQL database with multiple **tables, indices, triggers, and views**, is contained in a single disk file.

SQLite is a **compact** library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and **compiler optimization settings**. If optional features are omitted, the size of the SQLite library can be **reduced below 300KiB**. SQLite is a popular database engine choice on **memory constrained gadgets** such as **cellphones, PDAs**, and MP3 players. There is a **tradeoff between memory usage** and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in **low-memory environments**.

SQLite is not directly comparable to **client/server SQL database engines** such as MySQL, Oracle, PostgreSQL, or SQL Server since SQLite is trying to **solve** a different problem.

Client/server SQL database engines strive to implement a **shared repository** of enterprise data. They emphasize **scalability, concurrency, centralization, and control**. SQLite strives to provide **local data storage** for individual applications and devices. SQLite emphasizes **economy, efficiency, reliability, independence, and simplicity**.

### SITUATIONS WHERE SQLITE WORKS WELL

- » **Embedded devices** and the **internet of things**
- » Websites
- » Data analysis
- » **Cache** for enterprise data
- » **Server-side database**
- » File archives
- » Replacement for **ad hoc** disk files
- » Internal or temporary databases
- » Stand-in for an enterprise database during demos or testing
- » Experimental **SQL language extensions**
- » Education and Training

### Słownik

in-process – wewnętrzprocesowy  
library – biblioteka  
self-contained – samowystarczalny  
serverless – nie wymagający serwera  
zero-configuration – nie wymagający konfiguracji  
transactional SQL database engine – transakcyjny silnik baz danych SQL  
public domain – domena publiczna  
the most widely deployed – najczęściej wdrażany (używany)  
application – zastosowanie  
embedded – osadzony, wbudowany  
unlike most other – w odróżnieniu od większości innych  
separate – osobny  
server process – proces serwera  
ordinary disk file – zwykły plik dyskowy  
table – tabela  
index – indeks  
trigger – wyzwalacz  
view – widok  
compact – kompaktowy  
compiler optimization settings – ustawienia optymalizacyjne kompilatora  
toreduce – zredukować, ograniczyć  
KiB (kilobyte) – kibibajt ( $2^{10}$  bajtów)  
memory constrained – z mocno ograniczoną ilością pamięci

cellphone – telefon komórkowy  
tradeoff between – kompromis między...  
memory usage – zużycie pamięci  
low-memory environment – środowisko dysponujące małą ilością pamięci  
client/server SQL database engine – silnik baz danych SQL typu klient-serwer  
to solve – rozwiązać  
shared repository – wspólne repozytorium  
scalability – skalowalność  
concurrency – współbieżność  
centralization – centralizacja  
control – kontrola  
local data storage – lokalny magazyn danych  
economy – oszczędność  
efficiency – efektywność  
reliability – niezawodność  
independence – niezależność  
simplicity – prostota  
internet of things – internet rzeczy  
cache – bufor, pamięć podrzeczna  
server-side database – serwerowa baza danych  
ad hoc – tymczasowy, stworzony na poczekaniu  
language extensions – rozszerzenia języka

## ĆWICZENIA

1. Dopasuj słowa lub wyrażenia z lewej kolumny do słów lub wyrażeń z prawej. Potrafisz je wszystkie przetłumaczyć na język polski?

self-	hoc
in-	configuration
zero-	memory environment
public	database engine
shared	repository
ad	domain
language	process
low-	constrained
memory	extension
transactional SQL	contained

2. Odpowiedz na poniższe pytania. Możesz cytować tekst. Powtarzaj, aż będziesz w stanie udzielić odpowiedzi bez patrzenia na tekst.

1. What is SQLite?
2. What license is SQLite under?
3. How popular is SQLite?
4. What type of database engine is SQLite?
5. How does SQLite differ from most other SQL databases?
6. What is contained in a single disk file?
7. How big is SQLite?
8. Where is SQLite a popular database engine choice?
9. When does SQLite generally run faster?
10. What do client/server SQL database engines emphasize?
11. What does SQLite strive to provide?

3. Napisz zdania, używając wyrażenia „Unlike most other...”, jak w poniższym przykładzie.

*no separate server process -> Unlike most other SQL databases, SQLite does not have a separate server process.*

1. in-process library
2. self-contained
3. serverless
4. zero-configuration
5. public domain
6. free for use for any purpose
7. reads and writes directly to ordinary disk files
8. is contained in a single disk file
9. compact library
10. size can be less than 500KiB

## ODPOWIEDZI

Odpowiedzi do ćwiczeń znajdują się na stronie <http://shebang.pl/> programista-minikurs.

źródła tekstu:

- » <https://www.sqlite.org/whentouse.html>,
- » <https://www.sqlite.org/about.html>.



ŁUKASZ PIWKO

piwko.lukas@gmail.com

Tłumacz angielskiej i francuskiej literatury programistycznej z około 70 książkami na koncie, nauczyciel, wykładowca i maniak technologii programistycznych.

reklama

 hopeIT

{ **Developujesz?** }

Podejmij wyzwanie HopeIT hackathon

27–28/10/2017

CENTRUM TECHNOLOGII  
AUDIOWIZUALNYCH / WROCŁAW  
**#hopeithackathon**

**POMAGANIE**

**JEST W KODZIE**

# Indie Games.

## Podręcznik niezależnego twórcy gier



Żyjemy w bardzo interesujących czasach. Dziś praktycznie każdy może zostać twórcą gier. Ba, jego tytuł, o ile jest wystellarzająco innowacyjny i wyróżniający się z wszystkich pozostałych gier, może odnieść gigantyczny sukces komercyjny (dobrym przykładem jest tutaj Minecraft). Wielu developerów wcześniej czy później postanawia napisać swoją własną grę i odnieść choć częściowy sukces na tym jakże niełatwym rynku. Jednak mało który zdaje sobie sprawę, z czym tak naprawdę przyjdzie mu się zmierzyć.

Brakowało na naszym rynku książki, która choćby w częściowy sposób poruszałaby tematykę tworzenia gier kompleksowo. Tak, wiem, są tytuły stricte techniczne, w których opisane są algorytmy, silniki 3D i wszystko to, co dotyczy samego procesu pisania i testowania tytułu, jednak brakowało pozycji obejmującej temat w sposób całosciowy. Lukę tę wypełnia nowe wydawnictwo Merlin Publishing książką pod tytułem *Indie games. Podręcznik niezależnego twórcy gier*.

Na ponad 320 stronach autor opisuje własne doświadczenia związane z tworzeniem niezależnych gier. Książka ta jest takim „kubkiem zimnej wody” na rozpalone głowy początkujących twórców. Jest ciekawie skonstruowana, tak że praktycznie każdy rozdział może być czytany w dowolnym momencie i w dowolnej kolejności. Treść jest zaprezentowana w formie poradnika tworzonego przez kogoś, kto miał rzeczywistą styczność z daną problematyką, a nie tylko teoretyzuje w temacie zupełnie dla siebie nie znany. Fajnym rozwiązaniem są „Ankiety deweloperów Indie”, w których niezależne studia odpowiadają na ten sam zestaw jedenastu pytań, opisując swoją historię, oraz prezentują własne wnioski oparte na swoich doświadczeniach. Wbrew pozorom te ankiety są źródłem cennych informacji.

Sama książka została wydana w ciekowej formie. Sprawia wrażenie brulionu z notatkami, choć nie bez drobnych potknień składowo-redakcyjnych, ale można to zrzuścić na brak doświadczenia – jak by nie patrzeć – najmłodszego wydawcy na naszym rynku (wydawnictwo powstało w tym roku). Zastanawiające dla mnie jest to, że w takiej pozycji nie ma żadnych grafik mimo wyraźnych odniesień do nich w tekście. Z czystej ciekawości sprawdziłem oryginał i niestety tam jest sporo ilustracji, jednak ich brak nie obniża w żaden znaczący sposób wartości pozycji. Dla osób zainteresowanych rozpoczęciem przygody w tej branży ta książka powinna być lekturą obowiązkową. Praktycznie nigdzie indziej nie znajdę tak wyselekcjonowanych informacji!

Wracając do samej książki, zawiera ona dziewięć rozdziałów, które można czytać w dowolnej kolejności:

- » Rozdział 1: *Narzędzia i zasoby do tworzenia gier* – w tym około 70-stronicowym rozdziale opisano praktycznie każde zagadnienie związane z samym procesem tworzenia gier.
- » Rozdział 2: *Self-publishing* – kolejne około 70 stron autor poświęcił opisaniu wydawaniu gier na własny rachunek, jakie są zalety, jakie wady, na co należy zwrócić uwagę na najpopularniejszych platformach.
- » Rozdział 3: *QA, lokalizacja oraz kategorie wiekowe* – ten około 10-stronicowy rozdział został poświęcony procesowi wykry-

wania błędów, tłumaczenia oraz kategoryzacji wiekowej gry.

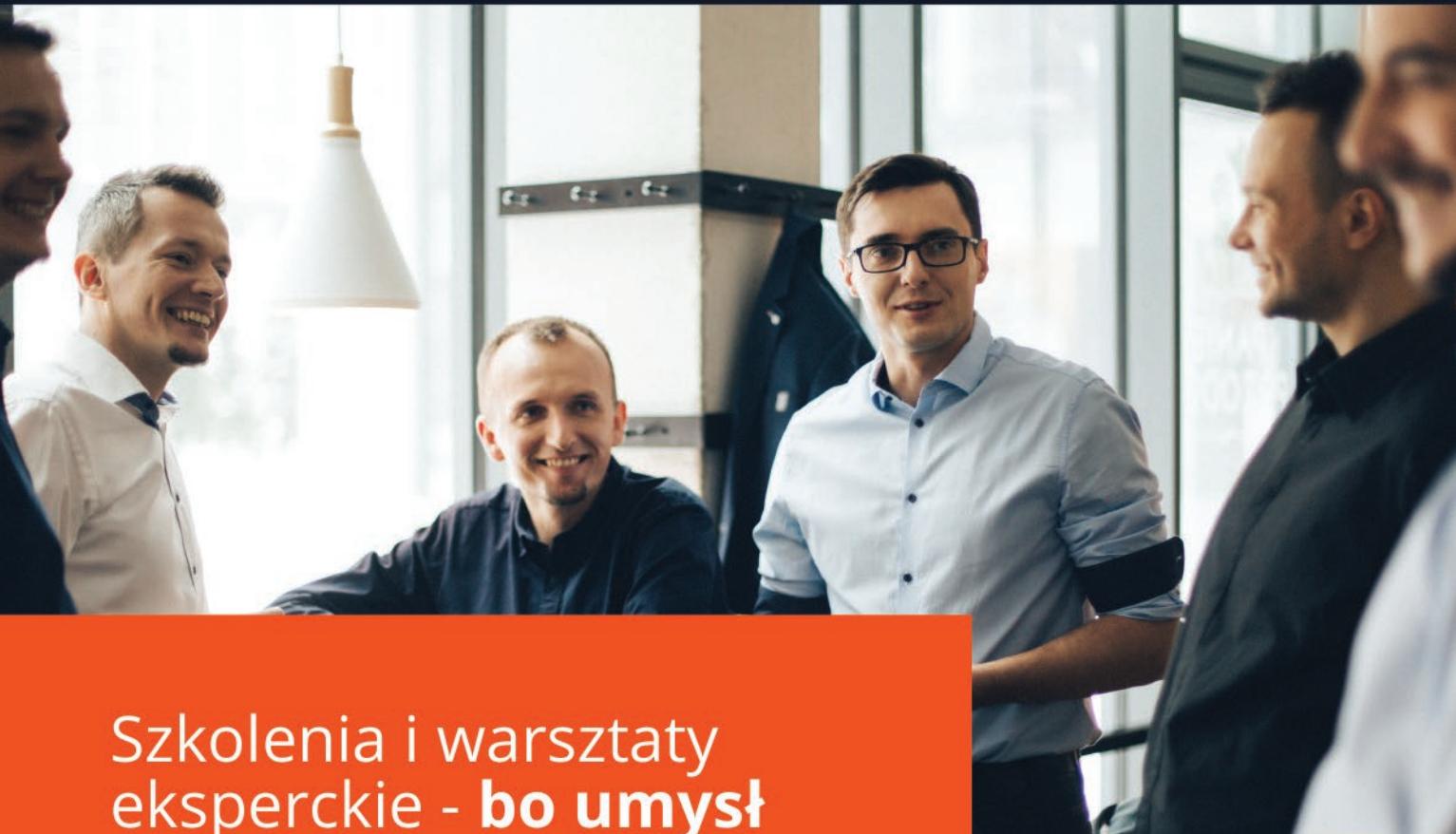
- » Rozdział 4: *PR oraz kontakt z prasą* – na kolejnych stronach autor opisuje kwestie zainteresowania rynku własną grą z wykorzystaniem różnych kanałów promocji.
- » Rozdział 5: *Marketing* – tu autor skupia się na kwestiach marketingu, poznania rynku, prezentuje listę rzeczy do zrobienia oraz opisuje narzędzia marketingowe.
- » Rozdział 6: *Reklama* – chyba wszyscy wiemy, że w dzisiejszych czasach bez reklamy nie istnieje dany produkt w świadomości konsumenckiej, ten rozdział przeznaczony jest w całości tej kwestii.
- » Rozdział 7: *Strony internetowe, fora i kontrola wersji* – autor cały rozdział poświęcił kwestii stworzenia skutecznego narzędzia promocyjno/sprzedażowego, jakim jest własna strona internetowa gry.
- » Rozdział 8: *Finanse* – jest to z jednej strony najważniejszy rozdział dla niezależnych developerów, z drugiej strony niestety ukierunkowany jest na zachodnie rynki (książka jest tylko tłumaczeniem).
- » Rozdział 9: *Kwestie podatkowe, prawne i inne ważne sprawy* – ostatni rozdział autor poświęcił kwestiom podatkowym zależnym od miejsca sprzedaży tytułu oraz umieszczaniu prawnych klauzul w grach.

Lektura książki może przytłoczyć ilością informacji i wiedzy. Jednak należy pamiętać o tym, że jest to jeden z bardziej złożonych tematów. Nie dość, że zajmujemy się samym tworzeniem gry, to musimy stworzyć sobie własny system kontrolowania jakości produktu, wypracować swoje sposoby komunikacji z community, promowania tytułu, pamiętać o wykorzystywaniu nadarzających się okazji do przypomnienia graczom o własnych wydanych i tworzonych obecnie tytułach i zapewnić sobie stałe źródło finansowania. Jeśli, Drogi Czytelniku, żyleś w przekonaniu, że tworzenie niezależnych gier to prosta i przyjemna praca, to ta książka wyprowadzi Cię szybko z błędu i uświadomi, że jest to jeden z cięższych kawałków chleba, przy czym satysfakcja, jaką mają twórcy niezależnych gier, jest praktycznie nie do zmierzenia.

Mówiąc krótko: w mojej opinii książka jest tzw. „must have” dla niezależnych developerów gier. Znajdziesz się w niej wszystko, co potrzebne jest, by wydać grę i kontrolować jej rozwój od samego początku istnienia do pojawienia się jej w sklepach i na różnych platformach cyfrowych. A jakie przygody i trudności miały poszczególne studia? Tego czytelnik dowie się sam po lekturze *Indie Games. Podręcznik niezależnego twórcy gier*, do której z całego serca namawiam.

Mariusz „maryush” Witkowski

Tytuł:	Indie Games. Podręcznik niezależnego twórcy gier
Autor:	Richard Hill-Whittall
Stron:	332
Wydawnictwo:	Merlin Publishing
Data wydania:	2017-05-18



Szkolenia i warsztaty eksperckie - **bo umysł to Twoje najważniejsze narzędzie**



DDD



ARCH



TEST&CRAFT



AGILE&SOFT



JAVA



.NET



C&CPP



WEB



BAZY



MOBILNE



EIP

SPRAWDŹ **200**  
**AUTORSKICH**  
**PROGRAMÓW**  
SZKOLEŃ

{FDD FUTURE  
DEV DAY

1 GRUDNIA 2017, GLIWICE



ORGANIZATOR

Future Processing

WIĘCEJ INFORMACJI:

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

[fb.com/futuredevday](https://fb.com/futuredevday)