

Magazyn programistów i liderów zespołów IT



Cena 23,90 zł (w tym VAT 5%)

PRZETWARZANIE I ANALIZA DANYCH OBSERWACJI ZIEMI W PYTHONIE

TO NIE JEST ZAWÓD DLA
STARYCH LUDZI

ZARZĄDZANIE PAMIĘCIĄ
W C++17

(NIE)BEZPIECZEŃSTWA JWT

CIEMNE STRONY JAVY 8

CAFEOBJ

MODUŁY W JAVIE 9



ISSN 2084-9400



03

9 772084 940800

Źródło: AdobeStock



Join our team of Makers!

career.cybercom.com

**Change
tomorrow
with us**

www.cybercom.pl

CYBERCOM
GROUP

Cybercom Poland Sp. z o.o.

ul. Hrubieszowska 2, 01-209 Warszawa / ul. Składowa 35, 90-127 Łódź / ul. Unii Lubelskiej 4c, 85-059 Bydgoszcz

Jak Python przez satelite Ziemię oglądał

Wiosna praktycznie w pełni. Przed nami słoneczne i ciepłe dni, które możemy spędzać w różny sposób. My proponujemy spędzić czas z naszym magazynem. Jak co miesiąc bowiem przygotowaliśmy zestaw bardzo interesujących artykułów.

Na początek proponujemy nasz temat numeru pt. „Przetwarzanie i analiza danych obserwacji Ziemi w Pythonie” autorstwa Szymona Molińskiego, w którym autor ujawnia tajemnice związane z obserwacjami satelitarnymi dotyczącymi naszej planety. Zastanawialiście się, czy może to być ciekawe zagadnienie i fajna zabawa? Odpowiedź znajdziecie w artykule.

Programujecie w Javie? A znacie jej ciemne strony? Jeśli nie, to zapraszamy do lektury artykułu Grzegorza Piwowarka, w którym autor opisuje najzwycięjsze w świecie wpadki, jakie popełniono przy tworzeniu wersji 8 tego języka. Dla miłośników tego języka przygotowaliśmy również artykuł pt. „Moduły w Javie 9”.

Tworzenie oprogramowania związane jest też z testami. Tylko mało kto zwraca uwagę w wystarczającym stopniu na testy bezpieczeństwa tworzonego kodu. Zagadnienie to opisuje Marcin Święty w swoim artykule pod tytułem „Testować bezpieczeństwo – co to znaczy?”.

Maj, kojarzony najczęściej z maturami, jest też miesiącem zmian w nas w redakcji. Ze swojego stanowiska w związku z innymi licznymi obowiązkami zrezygnował nasz kolega Michał Leszczyński – któremu w tym miejscu chcielibyśmy podziękować za owocną współpracę – a mojej osobie przypadł przywilej zastąpienia go. Tak więc korzystając z możliwości, chciałbym się przywitać z Wami, Drodzy Czytelnicy, i ze swojej strony obiecać, że czasopismo będzie się dalej rozwijało, tak jak dotychczas.

Mariusz „maryush” Witkowski

BIBLIOTEKI I NARZĘDZIA

Przetwarzanie i analiza danych obserwacji Ziemi w Pythonie.....	4
Szymon Moliński	
Moduły w Javie 9.....	12
Kamil Becmer	

JĘZYKI PROGRAMOWANIA

Zarządzanie pamięcią w C++17.....	18
Tomasz "satirev" Jaskólski	
CafeOBJ – maszynowy system dowodzenia.....	24
Marek Sawerwain	

BAZY DANYCH

NoSQL na przykładzie Apache Cassandra® i Scylla.....	32
Piotr Jastrzębski	

TESTOWANIE I ZARZĄDZANIE JAKOŚCIĄ

Testować bezpieczeństwo – co to znaczy?.....	36
Marcin Święty	

BEZPIECZEŃSTWO

(Nie) bezpieczeństwa JWT (JSON Web Token).....	42
Michał Sajdak	

KLUB LIDERA IT

Współczesne architektury aplikacji biznesowych. Warstwy i Domain-Driven Design.....	52
Mariusz Sieraczkiewicz	

LABORATORIUM TEINA

UX dark patterns ciąg dalszy, czyli jak z głową zarezerwować hotel.....	58
Katarzyna Małecka	

LABORATORIUM BOTTEGA

Ciemne strony Javy 8.....	62
Grzegorz Piwowarek	

PLANETA IT

To nie jest zawód dla starych ludzi.....	64
Radek Smilgin	
Tata programistą.....	68
Michał Lewandowski	

redakcja

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.

Prenumerata realizowana jest także przez RUCH S.A.

Zamówienia można składać bezpośrednio na stronie: www.prenumerata.ruch.com.pl

Pytania prosimy kierować na adres e-mail: 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).

Redaktor prowadzący: Mariusz „maryush” Witkowski (mariuszwitkowski@programistamag.pl).

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

Dział reklamy: reklama@programistamag.pl, tel. +48 663 220 102, tel. +48 604 312 716.

Prenumerata: prenumerata@programistamag.pl.

Współpraca: Michał Bartylek, Mariusz Sieraczkiewicz, Dawid Kaliszewski, Marek Sawerwain, Łukasz Mazur, Łukasz Łopuszański, Jacek Matulewski, Sławomir Sobótka, Dawid Borycki, Gynael Coldwind, Bartosz Chrabrski, Rafał Kocisz, Michał Sajdak, Michał Bentkowski, Paweł „KrzaQ” Zakrzewski.

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

Druk: <http://www.moduss.waw.pl/>, Nakład: 4500 egz.

Nota prawa

Redakcja zastrzega sobie prawo do skrótów i opracowań tekstów oraz do zmiany 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 bezpośredni i pośredni, jak również za inne straty i wydatki poniesione w związku z wykorzystaniem informacji prezentowanych na łamach magazynu Programista.

Przetwarzanie i analiza danych obserwacji Ziemi w Pythonie

Mamy szczęście programować w czasach, w których dostęp do wielkich zbiorów danych jest łatwiejszy niż kiedykolwiek wcześniej. A bardzo duże ilości danych otwierają wiele ścięzek, czy to indywidualnej, naukowej kariery, czy biznesowych możliwości. Wśród zbiorów dostępnych dla początkujących analityków danych wyróżniają się dane z sieci społecznościowych, z akcji użytkowników na stronach internetowych albo z systemów Internetu rzeczy, czyli sieci sensorów. Istnieją jednak jeszcze inne źródła informacji, które są godne uwagi. Są ogólnodostępne, darmowe, mają wysoką jakość i na ich podstawie startupy rosną jak grzyby po deszczu. Mowa tutaj o danych z satelitów obserwacji Ziemi.

GDZIE I PO CO WYKORZYSTUJEMY DANE OBSERWACJI ZIEMI?

Oczywistymi beneficjentami zobrazowań satelitarnych są sektory wojskowy i naukowy, więc pominiemy je w dalszych rozważaniach. Skupimy się na komercyjnych aplikacjach, gdzie wyróżnić można kilka nisz rynkowych, które mocno polegają na wykorzystywaniu zdjęć wykonywanych z orbity. Po pierwsze: rolnictwo. Monitoring upraw i chorób roślin, określenie optymalnego poziomu nawożenia roślin, analiza nawodnienia – to podstawowe tematy z kręgu zainteresowań rolników i wszystkie można zrealizować za pomocą zobrazowań satelitarnych (choć aktualnie warto wspomóc się monitoringiem z dronów, ze względu na znacznie lepszą rozdzielcość przestrzenną). Po drugie: zarządzanie kryzysowe, czyli detekcja pożarów, rejonów zagrożonych osunięciami, obszarów zalewowych. Po trzecie: złożone analizy przestrzenne wspomagające decyzje biznesowe w rynkach ubezpieczeniowym, transportowym, wydobywczym, turystycznym i sprzedaży detalicznej, często łączenie informacji z rastrów z innymi typami danych, aby przewidywać ceny ropy albo określić plony zboża i spodziewane ceny produktów roślinnych w następnym sezonie. Po czwarte: aplikacje dla masowego odbiorcy, gdzie zdjęcia satelitarne przedstawiane są w formacie RGB jako „podkładka” pod inne dane (jak na przykład widok satelitarny w *Google Maps*) albo bezpośrednio przedstawiające cenne informacje, np.: w aplikacjach meteorologicznych, aplikacjach turystycznych, aplikacjach przedstawiających zanieczyszczenie powietrza w postaci quasi-ciąglej na dużym obszarze (w przeciwieństwie do danych z pojedynczych czujników pomiarowych, które zbierane są punktowo).

DANE OBSERWACJI ZIEMI – SKĄD JE POZYSKAĆ?

Znając potencjał ukryty w tych danych, możemy przejść już do praktyki, czyli miejsc w sieci, skąd możemy za darmo pobrać dane satelitarne dobrej rozdzielcości. Są dwa główne portale, które wspomagają manualne, jak i zautomatyzowane pobieranie danych. Ze względu na to, że artykuł jest kierowany do osób, które nigdy wcześniej nie miały do czynienia z takimi danymi, nie będziemy zajmować się API. Jest ono potrzebne w sytuacji, kiedy wiemy:

- » jak przetwarzać dane satelitarne,
- » po co je przetwarzamy,
- » jakie mogą być problemy związane z jakością tych danych.

Część odpowiedzi przedstawiono w tym artykule.

Dwa główne miejsca w sieci, z których warto skorzystać w celu pobrania takich danych to *Copernicus Open Access Hub* oraz *EarthExplorer*.

Copernicus Open Access Hub

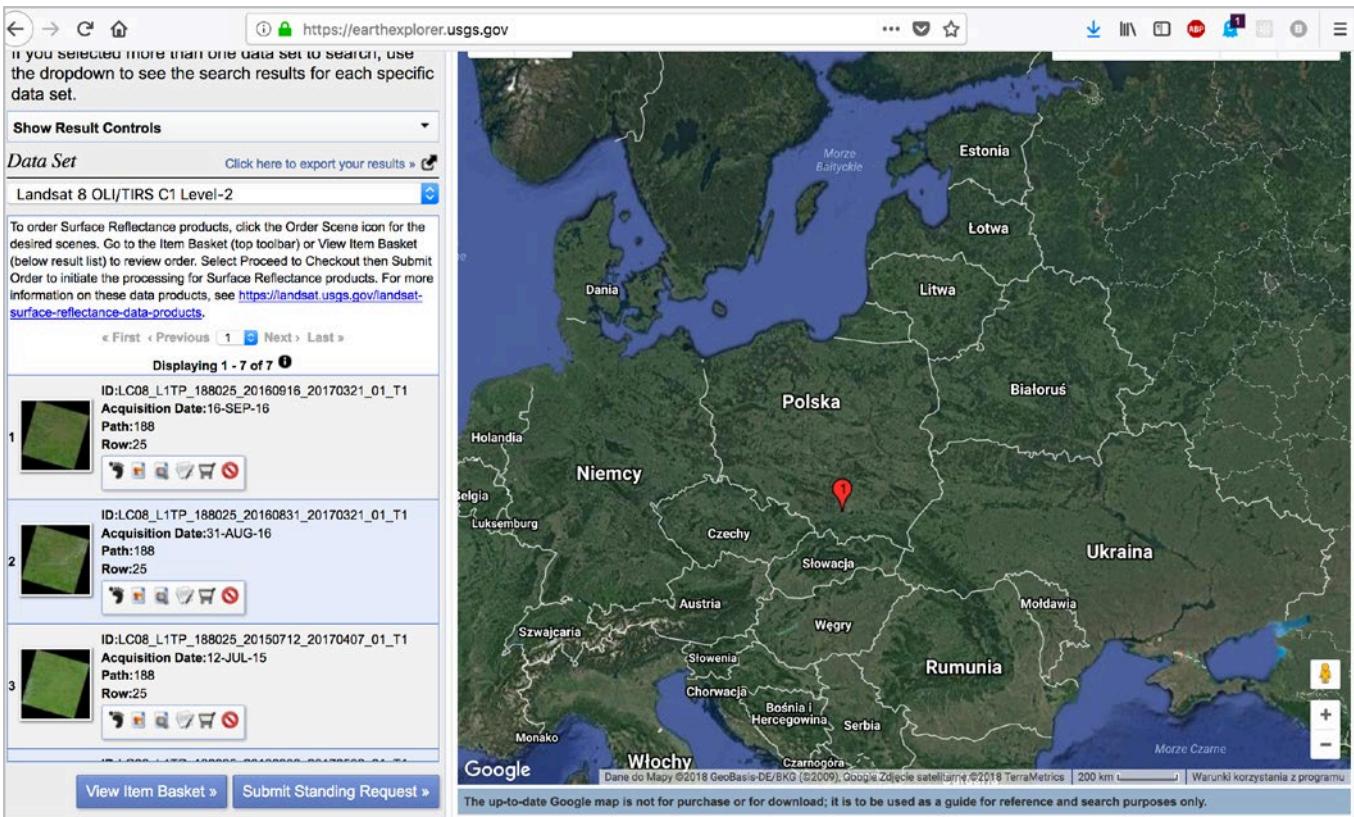
Pod adresem <https://scihub.copernicus.eu/> możemy dostać się do europejskiego systemu udostępniania danych z satelitów Sentinel 1, Sentinel 2 i Sentinel 3. W tym artykule nie będziemy obrabiać zdjęć z satelitów Sentinel, ale sposób pracy z obrazami z satelity Sentinel 2 jest bardzo podobny do przetwarzania danych satelitów z serii Landsat, więc warto wspomnieć o tym zbiorze. Tym bardziej że jest darmowy a jego rozdzielcość przestrzenna spełnia wymagania stawiane przed dużą grupą aplikacji.

EarthExplorer

Poczytając od tego akapitu, proponuję podążyć za artykułem krok po kroku, ponieważ od razu ściągniemy dane potrzebne do dalszej analizy. Na początku przechodzimy pod adres: <https://earthexplorer.usgs.gov/> i rejestrujemy się na portalu za pomocą przycisku *Register* po prawej stronie. Po rejestracji logujemy się w systemie za pomocą przycisku *Login*. Po zalogowaniu znowu jesteśmy na stronie głównej, z mapą po prawej i z polami wyboru danych po lewej stronie. Krok po kroku przejdziemy przez proces pobierania danych, które będą używane w tym artykule:

Zakładka Search Criteria

1. Address/Place -> wpisujemy „Kraków” i klikamy na przycisk „Show”, po czym z tabeli, która wyświetli się poniżej, wybieramy pozycję z polskim Krakowem w roli głównej. Na mapie po prawej powinien pojawić się wskaźnik, a w polu współrzędnych dokładny adres przestrzenny Krakowa.
2. Date Range:
 - » Search months -> zaznaczamy: June, July, August, September



Rysunek 1. Wybór scen Landsat 8

Wybór miejsca, z którego chcemy pozyskać dane satelitarne, może zostać dokonany na kilka sposobów. My idziemy po linii najmniejszego oporu i po prostu wybieramy interesujące nas miasto, ale gdybyśmy chcieli zebrać dane np. dla obszaru Polski, wtedy lepiej zaznaczyć punkty i granice na mapie (klikamy LPM i tworzymy wielokąt obrysowujący obszar zainteresował). Poszukiwania ograniczamy do czterech najlepszych miesięcy w roku, a robimy to z dwóch powodów: będącym analizować współczynnik vegetacji, a w miesiącach zimowych trudno o zielone rośliny, nie licząc drzew iglastych. W miesiącach letnich jest mniej chmur niż w trakcie jesieni, zimą albo wiosną.

Po wybraniu tych współczynników przechodzimy do kolejnej zakładki „Data Sets”.

Zakładka Data Sets

Tutaj możemy się nieco wystraszyć, bo liczba zbiorów danych jest przytaczająca. Na szczęście wiemy, czego chcemy, więc:

1. Przechodzimy do zbioru *Landsat*.
2. Przechodzimy do zbioru *Landsat Collection* oraz do *Level-2 (On-Demand)*.
3. Wybieramy pole *Landsat 8 OLI/TIRS C1 Level-2*.

Po wykonaniu kroku 3 na dole powinny podświetlić się dwa przyciski: *Additional Criteria* oraz *Results*. Za chwilę przejdziemy do zakładki *Additional Criteria*, ale czytelnikowi należy się jeszcze kilka słów wyjaśnienia, dlaczego wybraliśmy właśnie ten konkretny zbiór danych. Czy widzimy, że są tam również inne poziomy tworzenia danych, takie jak *C1 Level 1*? Ich użycie jest kuszące, bo można je pobrać od razu. Haczyk tkwi w obowiązku przetworzenia tych danych, przede wszystkim ich korekcji atmosferycznej. Nie jest to zadanie trywialne, dlatego zaufajmy *USGS* i algorytmom

opracowanym przez specjalistów: wybieramy poziom 2 przetwarzania danych i nie musimy się martwić o dodatkowe, złożone obliczenia. Możemy też przejść od razu do analizy wskaźników ze zdjęcia. Jeśli jesteśmy zainteresowani, co właściwie stało się z danymi, zapraszam do zapoznania się z informacjami na stronie: <https://lta.cr.usgs.gov/L8Level2SR>.

Podstawowe zbiory danych obserwacji Ziemi o stosunkowo dobrej rozdzielczości przestrzennej to zdjęcia wykonywane przez satelity z serii Landsat. Jak zapewne się domyślamy, satelity Landsat 4-5 oraz Landsat 7 to starsze modele (które wciąż dostarczają cennych danych), a Landsat 8 jest najnowszym satelitą USGS. W tym artykule będziemy analizować tylko jedną scenę, ale gdybyśmy chcieli obserwować zmiany w czasie, najpewniej używalibyśmy danych ze wszystkich Landsatów, bo każdy zaczął swoją orbitalną wędrówkę w innym czasie. Ich systemy obrazowania są do siebie podobne, więc analiza wskaźników nie jest obarczona dużym błędem przy porównaniach.

Zakładka Additional Criteria

W zakładce *Additional Criteria* zaznaczamy:

1. *Land Cloud Cover: Less than 10%*
2. *Scene Cloud Cover: Less than 10%*

Nie chcemy, żeby chmury przeszkadzały nam w dalszej analizie, więc będziemy szukać ujęć, na których powierzchnia nie jest zasłaniana przez obłoki, a dane nie są fałszowane przez cienie rzucane na Ziemię przez chmury. Czas na kliknięcie w zakładkę *Results*.

Zakładka Results

Jeśli postępowaliśmy dokładnie jak było opisywane w artykule, powinniśmy zobaczyć widok, jak na Rysunku 1.

Już na miniaturkach widać, że sceny są czyste, pozbawione skupisk chmur. O to nam chodzi! Teraz trzeba zdecydować się na jedną ze scen, na potrzeby artykułu wybierzmy obrazy o następujących parametrach:

1. ID: LC08_L1TP_188025_20130807_20170503_01_T1
2. Acquisition Date: 07-AUG-13
3. Path: 188
4. Row: 25

Klikając czarną ikonkę stopy, możemy zobaczyć, jaki obszar pokrywa nasze zdjęcie. Kraków powinien być w centrum tego obszaru. Po kliknięciu w ikonę obok przedstawiającą miniatułę obrazu, na mapie po prawej wyświetlna zostanie scena (żeby była widoczna, musimy odznaczyć ikonę czarnej stopy). Ikona z lupą nie jest nam na razie potrzebna, za to ikona z notatkami jak najbardziej. Po kliknięciu w nią otworzy się okno, gdzie możemy poznać podstawowe informacje o scenie.

Proponuję przeklikać się w tym oknie przez 3 obrazy z galerii i rzucić okiem na ostatni, na którym przedstawiono białe chmury z niebieskimi cieniami. To bardzo cenna informacja, bo pokazuje, jaka część obrazu jest uszkodzona z przyczyn naturalnych, i analiza tych obszarów nie jest możliwa albo obarczona jest dużym błędem. Zwyczajowo piksele z chmurami i ich cieniami zastępuje się wartościami NaN.

Zamknijmy okno podglądu i przejdźmy do procesu pobierania. Wielkość produktów *USGS* można pobrać od razu, ale w przypadku produktów drugiego poziomu musimy wysłać zapytanie o przetworzenie danych. Klikamy więc w ikonę koszyka. Powinna zostać podświetlona. Na dole klikamy w przycisk *View Item Basket*. Powinniśmy mieć tam jedną scenę. Klikamy przycisk *Proceed To Checkout*, a później *Submit Order* – dane są darmowe, więc nie musimy się martwić nazewnictwem przycisków.

Teraz pozostało nam zająć się czymś innym – na przykład przeczytaniem artykułu. Scena jest przetwarzana. Dostaniemy dwa maile, jeden z informacją o rozpoczęciu przetwarzania i drugi po jej przetworzeniu. Pierwszy powinien przyjść po kilku minutach, a drugi po kilkunastu-kilkudziesięciu minutach, ja zwykle czekam 15 minut na jedną scenę. Wchodząc w link podany w drugim mailu i klikając na odnośnik do naszego produktu, możemy rozpocząć pobieranie po kliknięciu na *Download*. Zapisujemy archiwum na dysku (na razie w folderze pobranych) i przechodzimy do właściwej analizy zdjęcia.

PRZYGOTOWANIE ŚRODOWISKA PRACY

Pracować będziemy w środowisku Anaconda, które znaczco ułatwia zarządzanie wirtualnymi środowiskami do zadań związanych z eksploracją i analizą danych. Dlatego pierwszym krokiem jest pobranie i instalacja aktualnej wersji Anacondy dla Pythona 3.6 ze strony: <https://www.anaconda.com/download/>. Szczegółowy opis instalacji opisany jest na stronie: <https://docs.anaconda.com/anaconda/install/>. Kod prezentowany niżej przetestowany został na systemach macOS i Linux. Używanie Anacondy nie jest obowiązkowe, wirtualne środowisko ze wszystkimi potrzebnymi paczkami można stworzyć ręcznie albo z poziomu PyCharm'a.

Po instalacji i konfiguracji Anacondy zapraszam pod adres: <https://github.com/szymon-datalions/l8p>, skąd można ściągnąć albo sklonować całe repozytorium z plikami potrzebnymi do dalszej pracy, oprócz plików rasterowych pobieranych wcześniej ze

strony *USGS*. Pliki z *USGS* rozpakowujemy i przerzucamy do folderu ściągniętego z repozytorium Githuba.

Dwa kroki za nami (czyli instalacja Anacondy i ściągnięcie repozytorium), czas na uruchomienie środowiska.

1. Otwieramy terminal i przechodzimy do katalogu z repozytorium.
2. W terminalu wpisujemy komendę `conda env create`. Conda powinna w tym momencie odczytać plik `enviornment.yml` i zainstalować w środowisku odpowiednie biblioteki.

Po zakończeniu konfiguracji środowiska możemy przejść do kolejnego etapu.

PODSTAWOWE OPERACJE

W terminalu przechodzimy do katalogu `l8p` (o ile już w nim nie jesteśmy), uruchamiamy środowisko Condy iłączamy notatnik Jupyter:

```
>> source activate l8p  
>> jupyter notebook
```

Po wydaniu ostatniej komendy, w przeglądarce, którą zazwyczaj używamy, powinna otworzyć się strona Jupytera, ewentualnie w terminalu znajduje się link do środowiska lokalnego – kopujemy go i wklejamy do okna przeglądarki. Zobaczmy następującą listę folderów i plików:

```
f/ clipped  
f/ LC081880252013080701T1-SC20180329040242  
f/ vector  
p/ cwiczenie.ipynb  
p/ environment.yml  
p/ podstawy_przetwarzania_E0.ipynb  
p/ rozwiazanie.ipynb
```

Folder `LC081880252013080701T1-SC20180329040242` podmieniający własnym folderem ze ściągniętymi i rozpakowanymi danymi z Landsata. W `gitignore` uwzględniliśmy pliki `.tiff`, dlatego folder jest „pusty” i zachowane są tylko metadane. Żeby poprawnie wykonać ćwiczenie, musimy pamiętać o tym, że pliki `.tiff` są niezbędne do jego ukończenia. W folderze `vector` znajdują się pliki wektorowe w formacie `shapefile (.shp)`. Przygotowałem je specjalnie dla tego ćwiczenia i przedstawiają one granice administracyjne Krakowa i powiatu krakowskiego. Ich rzutowanie jest zgodne z projekcją danych rasterowych. Nie musimy się przejmować potencjalnymi błędami spowodowanymi przez różne reprezentacje przestrzenne.

Będziemy pracować z plikiem `cwiczenie.ipynb`. W pliku `podstawy_przetwarzania_E0.ipynb` mamy zebrane wszystkie funkcje z ćwiczenia i dodatkowe wizualizacje, które możemy uznać za istotne. Plik `rozwiazanie.ipynb` to wypełnione ćwiczenie, dla porównania. Uruchommy plik `cwiczenie.ipynb` i zaczynajmy!

Oto lista kroków w zadaniu:

1. Import potrzebnych bibliotek.
2. Funkcja wczytująca adresy plików do przetwarzania.
3. Funkcja do wyświetlania obrazów.
4. Wczytanie listy dostępnych kanałów i wyświetlenie wybranego obrazu.
5. Usunięcie wartości oznaczających brak danych, poprawa funkcji do wyświetlania obrazów.
6. Funkcja do wycięcia obszaru zainteresowania z rastra.
7. Przetworzenie wszystkich rasterów do postaci obejmującej tylko obszar zainteresowań.
8. Wskaźniki, które możemy pozyskać z danych satelitarnych.

1. Import potrzebnych bibliotek

W toku pracy potrzebować będziemy paczek odpowiedzialnych za wczytywanie plików (a właściwie przeglądanie katalogów), obliczenia, pracę z danymi rastrowymi i wektorowymi oraz wyświetlanie danych. Wczytujemy więc wszystkie potrzebne biblioteki i pamiętamy o poleceniu `%matplotlib inline` albo `%matplotlib notebook` na początku notatnika. Pozwalają one na wyświetlanie wykresów i obrazów w notatniku (Listing 1):

Listing 1. Import bibliotek

```
%matplotlib notebook
# Import podstawowych bibliotek

import os
import numpy as np
import rasterio as rio
import rasterio.mask as rmask
import fiona as fio
import matplotlib.pyplot as plt
```

2. Funkcja wczytująca adresy plików do przetworzenia

Piszemy pomocniczą funkcję do przygotowania naszych obrazów do dalszej pracy. Wykorzystujemy do tego metodę `os.listdir` i sprawdzamy, czy w folderze znajdują się pliki o początku „LC” i końcu „.tif”. Sprawdzamy też, czy w nazwie pliku znajduje się słowo „band”, po czym dodajemy go do listy, którą sortujemy, i tworzymy z niej słownik. Funkcja zadziała tylko w przypadku, jeśli w folderze mamy 7 podstawowych kanałów, w innym przypadku należy dostosować zmienną `channel_numbers`, albo podawać ją jako parametr (Listing 2):

Listing 2. Przygotowanie do analizy danych. Wczytywanie plików

```
# Funkcja do wczytywania Listy kanałów i porządkowania jej
def read_landsat_images(folder_name):
    """Funkcja zwraca słownik (dict) z parami NUMER KANAŁU:
    ścieżka do pliku
    return: {numer_kanalu: ścieżka_do_pliku}"""
    file_list = os.listdir(folder_name)
    channel_list = []
    for f in file_list:
        if (f.startswith('LC') and f.endswith('.tif')):
            if 'band' in f:
                channel_list.append(folder_name + f)
    channel_list.sort()
    channel_numbers = np.arange(1, 8)
    bands_dictionary = dict(zip(channel_numbers, channel_list))
    return bands_dictionary

# Test
satellite_images =
read_landsat_images('LC081880252013080701T1-SC20180329040242/')
for band in satellite_images:
    print(band, satellite_images[band])
```

3. Funkcja do wyświetlania obrazów

Zadaniem funkcji jest tak naprawdę „upakowanie” metod oferowanych w pakiecie `matplotlib.pyplot` do jednego obiektu. Funkcja tworzy płótno, wyświetla na nim piksele i odpowiednio je koloruje – w zależności od wybranej mapy kolorów. Dodatkowo tworzy legendę. Mapy kolorów zaprezentowane są pod adresem: https://matplotlib.org/examples/color/colormaps_reference.html. Funkcję zapisano w Listingu 3.

Listing 3. Przygotowanie do analizy danych. Wyświetlanie obrazów

```
# Funkcja do wyświetlania poszczególnych obrazów
def show_band(band, color_map='gray'):
```

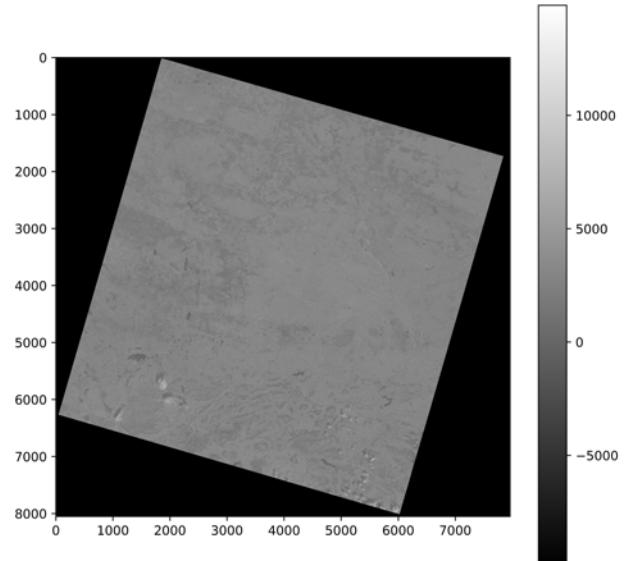
```
fig = plt.figure(figsize=(8,8))
image_layer = plt.imshow(band)
image_layer.set_cmap(color_map)
plt.colorbar()
plt.show()

# Test
test = np.random.randint(low=0, high=255, size=(200, 200))
show_band(test, color_map='winter')
```

4. Wczytanie listy dostępnych kanałów i wyświetlenie przykładowego obrazu

W tym kroku przechodzimy do prawdziwych danych i zajmujemy się ich eksploracją, na początku wizualną. Kod przedstawiono w Listingu 4.

Wykorzystamy naszą funkcję `read_landsat_images(folder_name)` i przygotujemy słownik, w którym kluczami będą liczby całkowite oznaczające poszczególne kanały, a wartościami będą adresy do plików z obrazami. Następnie, z pomocą biblioteki rasterio, otworzymy dostęp do pliku za pomocą metody `open` i utworzymy zmienną typu `numpy array` za pomocą metody `read`. W macierzy zapisany zostanie obraz bez dodatkowych informacji przestrzennych. W wyniku powinniśmy otrzymać obraz przedstawiony na Rysunku 2:



Rysunek 2. Widok na Polskę z kanału piątego satelity Landsat 8

Listing 4. Testy funkcji wczytującej i wyświetlającej dane

```
# Wczytywanie plików obrazów i sporządzenie ich listy
band_list = read_landsat_images('LC081880252013080701T1-SC20180329040242/')

# Wczytywanie obrazu i jego wyświetlenie
with rio.open(band_list[5], 'r') as src:
    band_matrix = src.read(1)

show_band(band_matrix)
```

Więcej informacji o znaczeniu poszczególnych kanałów można znaleźć tutaj: <https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands/>. W tym ćwiczeniu będziemy używać kanałów 3-6. Zapewne zauważylismy, że obraz jest niewyraźny, a wartości zamkają się w przedziale od około -10 000 do około 15 000. Wartości ujemne oznaczają w tym przypadku brak danych i wypadałoby się ich pozbyć, przynajmniej przy wyświetlaniu danych.

5. Usunięcie wartości oznaczających brak danych, poprawa funkcji do wyświetlania obrazów

Poprawne obsłużenie danych wymaga od nas kilku zmian w funkcji `show_band` (Listing 5). Po pierwsze, dodajemy parametr `remove_negative`, domyślnie będący `True`. Wewnątrz metody wartości pikseli macierzy są porównywane do zera. Jeśli są mniejsze lub równe zero, wtedy zmieniamy je na `NaN`. Takie dane nie są wyświetlane na płótnie.

Listing 5. Poprawiona funkcja do wyświetlania przetworzonych zdjęć

```
# Funkcja do wyświetlania poszczególnych obrazów - v 1.1
def show_band(band, color_map='gray', remove_negative=True):
    matrix = band.astype(float)
    if remove_negative:
        matrix[matrix <= 0] = np.nan
    fig = plt.figure(figsize=(8,8))
    image_layer = plt.imshow(matrix)
    image_layer.set_cmap(color_map)
    plt.colorbar()
    plt.show()
```

Proponuję kolejny raz wczytać kanał 5 (będzie najbardziej wyraźny) i tym razem wyświetlić go za pomocą poprawionej funkcji `show_band`.

6. Funkcja do wycięcia obszaru zainteresowania z rastra

Pliki, które dotychczas przetwarzaliśmy, są wielkie. O ile nie posiadamy zestawu monitorów do wyświetlenia ich w bardzo dobrej rozdzielcości, to tak naprawdę tracimy możliwość wizualnej interpretacji. Nie wiadomo, co właściwie znajduje się na obrazie. To nie wszystko. O ile nie zajmujemy się wielkoskalowymi studiami ekologicznymi albo powiązaniami między odległymi obszarami, zużywamy bardzo dużo zasobów komputera na przetwarzanie niepotrzebnych danych. Dlatego zanim przejdziemy do analizy obrazów, został nam ostatni krok związany z ich przetwarzaniem: ograniczenie rastrów do obszaru zainteresowania.

Dane przestrzenne różnią się pod tym względem od zwykłych obrazów, nie możemy po prostu wyciąć fragmentu macierzy, bo stracimy informacje na temat lokalizacji pikseli w przestrzeni. Można temu zaradzić, przeprowadzając odpowiednie obliczenia geometryczne. Dla prostych wielokątów nie stanowi to problemu, za to dla granic administracyjnych, dla których często przeprowadza się analizy i symulacje, sprawa nieco się komplikuje. Na szczęście wykorzystując dwie biblioteki Pythona: `fiona` i `rasterio`, możemy to zrobić za pomocą kilku linijek kodu.

Wspomnę tutaj jeszcze o projekcji, która jest nieodzownym elementem pracy z danymi przestrzennymi. Dane w rzeczywistych zastosowaniach często różnią się projekcją i na przykład granice administracyjne regionów Polski są przedstawiane w projekcji EPSG:2180 specjalnie przygotowanej przez naukowców po to, by znieskażniać przestrzeń dla obszaru naszego kraju w rzuconych obrazach były jak najmniejsze. Z drugiej strony dane Landsat są prezentowane w zupełnie innym układzie odniesienia: UTM, który obejmuje całą Ziemię. Adres miasta w Polsce będzie inny w projekcji EPSG:2180 i UTM. Należy zdecydować się na jeden z układów i według niego przeprowadzać przekształcenia danych oraz ich analizę. Najbezpieczniej jest przekształcać pliki wektorowe (np. granice administracyjne albo punkty) między różnymi projekcjami, ponieważ wektor sam w sobie nie reprezentuje żadnych wartości fizycznych. Przy reprojekcji rastrów narażamy się na utratę danych, szczególnie wartości ekstremalnych, ponieważ operacje zmiany rzutowania działają jak filtr uśredniający. Temat projekcji

jest tak rozległy, że mógłby zająć kilka artykułów, więc w tej chwili musimy pamiętać o tym, że wszystkie dane przestrzenne powinny mieć tę samą projekcję, żebyśmy mogli je przetworzyć.

Wcześniej wspomniałem o folderze `vector` i pliku wektorowym obejmującym granice administracyjne Krakowa i powiatu krakowskiego. Właśnie teraz z niego skorzystamy (Listing 6). Wykonanie funkcji składa się z dwóch etapów: najpierw następuje przycięcie rastra do granic wektora. Następnie zapisanie wyciętego pliku do nowego folderu. Fiona otwiera plik wektorowy i zapisuje adresy wszystkich punktów tworzących granicę administracyjną do zmiennej `geometry`. Następnie rasterio wczytuje obraz, nakłada na niego maskę utworzoną przez fiona i nadpisuje odpowiednio metadane przestrzenne obrazu. Na koniec raster, przedstawiający obszar zainteresowania, jest zapisywany do nowego pliku.

Listing 6. Przygotowanie do danych. Funkcja do wycięcia granic do analizy

```
# Funkcja do wycinania fragmentów rastra na podstawie szablonu
# z pliku wektorowego
def clip_area(vector_file, raster_file, save_image_to):
    with fio.open(vector_file, 'r') as clipper:
        geometry = [feature['geometry'] for feature in clipper]
    with rio.open(raster_file, 'r') as raster_source:
        clipped_image, transform = rmask.mask(raster_source,
                                                geometry, crop=True)
        metadata = raster_source.meta.copy()
    metadata.update({'driver': "GTiff",
                    "height": clipped_image.shape[1],
                    "width": clipped_image.shape[2],
                    "transform": transform})
    with rio.open(save_image_to, "w", **metadata) as g_tiff:
        g_tiff.write(clipped_image)
```

7. Przetworzenie wszystkich rastrów do postaci obejmującej tylko obszar zainteresowań

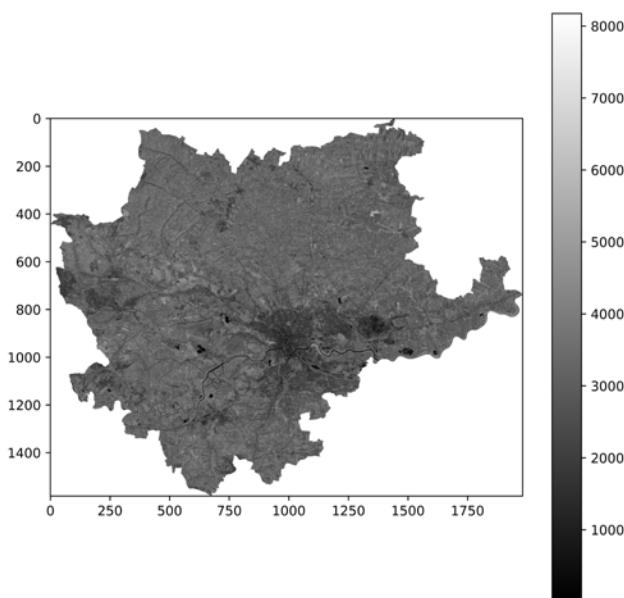
Teraz pozostało nam przetestować naszą funkcję do wycinania rastrów i wyświetlenie przykładowego obrazu. Po tym kroku przejdziemy do ostatniego etapu – będącego rzeczywistą analizą.

Połączmy wymienione wcześniej kroki w całość do momentu otrzymania listy wyciętych rastrów (Listing 7). Ważne jest nazewnictwo wyciętych plików. W nazwie musi się znaleźć ciąg znaków „LC” na początku i „tif” na końcu oraz „band” we wnętrzu. Zachęcam do zmiany tych sztywnych ustawień wedle własnych potrzeb i gustów!

Listing 7. Przygotowanie danych do analizy

```
# 1. Wczytanie plików obrazów i sporządzenie ich listy
bands = read_landsat_images('LC081880252013080701T1-
SC20180329040242/')
# 2. Wycięcie obszaru zainteresowania z każdego pliku
vector = 'vector/krakow_krakowskie.shp'
clipped_folder = 'clipped/' # Pamiętaj o utworzeniu tego
# folderu wcześniej!
for band in bands:
    destination = clipped_folder + 'LC_clipped_band' + str(band)
    + '.tif'
    clip_area(vector, bands[band], destination)
# 3. Wczytanie listy przetworzonych plików
clipped_bands = read_landsat_images('clipped/')
for band in clipped_bands:
    print(clipped_bands[band])
# 4. Wyświetlenie przykładowego pliku
with rio.open(clipped_bands[5], 'r') as src:
    band_matrix = src.read(1)
show_band(band_matrix)
```

Oto obraz, który powinniśmy zobaczyć:



Rysunek 3. Powiat krakowski i miasto Kraków, zdjęcie wykonane przez satelitę Landsat 8

8. Wskaźniki, które możemy pozyskać z danych satelitarnych

Na chwilę odłożymy na bok kod. Musimy zrozumieć, czym są wskaźniki pozyskiwane z danych satelitarnych. Satelity obserwacji Ziemi to złożone instrumenty badawcze, które operują na wielu kanałach spektralnych. Niektóre satelity mają dziesiątki tych kanałów, dlatego są nazywane aparatami hiperspektralnymi. Każdy z kanałów, a właściwie każdy przedział długości fal elektromagnetycznych rejestrowanych przez satelitę ma związek z pewnymi procesami fizycznymi i biologicznymi: wilgotnością, zawartością chlorofilu w liściach, zawartością soli morskiej w wodzie, temperaturą powierzchni itp. W trakcie eksperymentów wyznaczono kilka różnicowych wskaźników. Wskaźniki te podkreślają charakterystyczne obiekty i zjawiska, przy tym wyłumiając inne: na przykład różnicowy wskaźnik wegetacji wskazuje obszary o dużym zagęszczeniu roślin jako jaśniejsze niż obszary zabudowane. Dodatkowo zdrowsze, poprawnie nawodzone i nawodnione rośliny „jaśniają” na obrazach tego wskaźnika. Umożliwia on również klasyfikację powierzchni i może przysłużyć się do zbudowania nienadzorowanego modelu klasyfikacji.

Przygotujemy funkcję do obliczania tych wskaźników. Mają one swoje nazwy, wymienione poniżej:

- » NDBI – znormalizowany różnicowy wskaźnik zabudowy.
- » NDVI – znormalizowany różnicowy wskaźnik wegetacji.
- » NDWI – znormalizowany różnicowy wskaźnik wody.

Wartości tych wskaźników będą zawierać się w przedziałach -1 do 1. Zwykle można pominąć wartości ujemne, dlatego będziemy je wycinać z wynikowych obrazów.

Wskaźnik *NDBI* sam w sobie powinien przedstawać zabudowę, jednak jest on „słabym” wskaźnikiem i często myli roślinność z zabudową. Dlatego różnica między *NDBI* a *NDVI* zwróci znacznie lepsze wskaźniki zabudowy, bo obszary, gdzie rośliny były wartością maksymalną, zostaną wyzerowane.

Wskaźnik *NDVI* jest jednym z najpopularniejszych wskaźników. Wykorzystywany przede wszystkim w rolnictwie, klimatologii i ekologii. Jest na tyle istotny, że kanały w multispektralnych kamerych dronów wykorzystywanych w rolnictwie dobierane są tak, by móc łatwo obliczyć *NDVI*. Wysokie wartości *NDVI*, bliskie 1, pokazują obszary o gęstej i dobrze nawodnionej roślinności. Niskie wartości odpowiadają miastom i zbiornikom wodnym.

Ostatni ze wskaźników, *NDWI*, przedstawia zbiorniki i cieki wodne. Generalnie im wyższe wartości, tym woda czystsza. Za jego pomocą można monitorować zakwity sinic albo wyszukiwać zakłady komunalne/przemysłowe, które „trują” środowisko.

Wskaźniki są bardzo proste do obliczenia, dzielimy różnicę dwóch kanałów przez ich sumę. Trzeba tylko wiedzieć, które kanały wykorzystać. Przy kanałach z Landsata 8 wybieramy następujące obrazy, gdzie B (band) oznacza numer kanału:

- » $NDBI = (B6 - B5) / (B6 + B5)$
- » $NDVI = (B5 - B4) / (B5 + B4)$
- » $NDWI = (B3 - B6) / (B3 + B6)$

Ostatecznie nasza funkcja do obliczeń przedstawiona jest w Listingu 8.

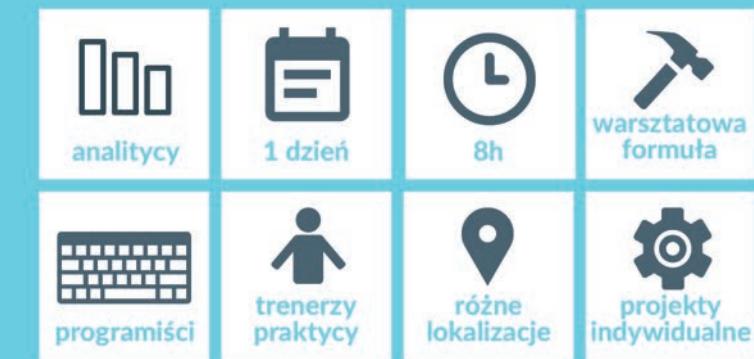
Listing 8. Analiza danych. Obliczenia wskaźników różnicowych

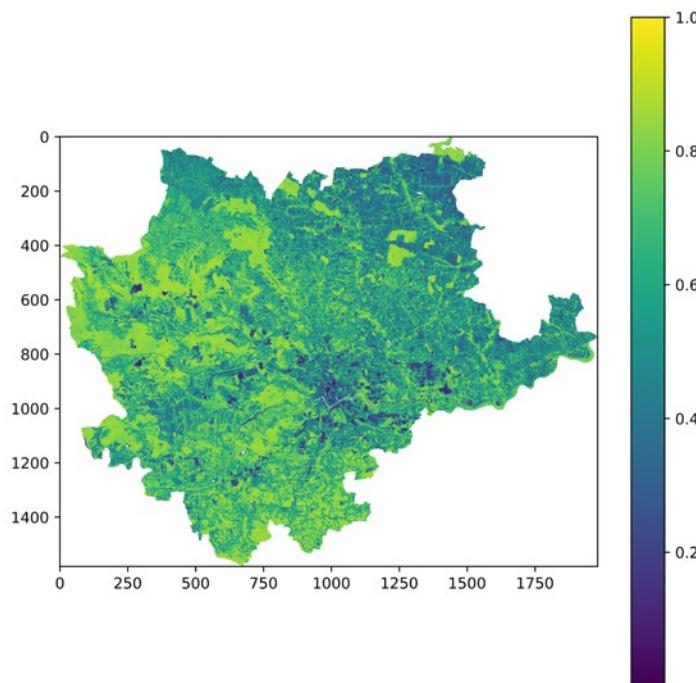
```
def calculate_index(index_name, landsat_8_bands):
    indexes = {
        'ndvi': (5, 4),
        'ndbi': (6, 5),
        'ndwi': (3, 6),
    }
    # Magiczne 10000, przez które dzielone są piksele poszczególnych
    # obrazów, to maksymalna wartość pikseli w produktach poziomu 2
    # satelity Landsat 8
```

Szkolenie dla Ciebie lub Twojego zespołu

Warsztat analityka danych w języku Python

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages ważnej przy zamówieniach złożonych do końca czerwca 2018 r. Hasło: PROGRAMISTAMAG





Rysunek 4. Znormalizowany wskaźnik wegetacji dla Krakowa i powiatu krakowskiego

```
if index_name in indexes:  
    bands = indexes[index_name]  
  
    with rio.open(landsat_8_bands[bands[0]]) as a:  
        band_a = (a.read()[0]/10000).astype(np.float)  
    with rio.open(landsat_8_bands[bands[1]]) as b:  
        band_b = (b.read()[0]/10000).astype(np.float)  
  
    numerator = band_a - band_b  
    denominator = band_a + band_b  
  
    idx = numerator / denominator  
    idx[idx > 1] = 1  
    idx[idx < -1] = -1  
  
    return idx  
else:  
    raise ValueError('Brak wskaźnika do wyboru, dostępne  
    wskaźniki to ndbi, ndvi i ndwi')
```

```
# 6. Oblicz NDWI  
ndwi = calculate_index('ndwi', clipped_bands)  
ndwi[ndwi == 0] = np.nan  
show_band(ndwi, color_map='viridis', remove_negative=False)  
  
# 7. Oblicz NDBI  
ndbi = calculate_index('ndbi', clipped_bands)  
ndbi[ndbi == 0] = np.nan  
show_band(ndbi, color_map='viridis', remove_negative=False)  
  
# 8. Różnica NDBI-NDVI  
show_band(ndbi-ndvi, color_map='viridis', remove_negative=False)
```

Przykładowy obraz znajduje się na Rysunku 4 (NDVI).

Posiadając te dane, możemy już przystępować do analizy przestrzennej danych. Jeśli mamy takie ambicje, możemy również porównywać serie czasowe na kilkanaście lat wstecz – i tworzyć przidykle przyszłości.

PODSUMOWANIE

Informacje, które przedstawiłem czytelnikowi w tym artykule, są wystarczające do rozpoczęcia własnej pracy z danymi satelitarnymi. Jednakże temat ich analiz jest bardzo rozległy. Algorytmy i metody statystyczne związane z informacją przestrzenną stanowią oddzielną klasę problemów, z którymi może mierzyć się analityk danych, naukowiec albo programista. Zachęcam do eksploracji tematu, ponieważ źródła danych są łatwo dostępne, a dziedzina analizy przestrzennej potrzebuje świeżego spojrzenia ludzi „spoza branży GIS”, szczególnie osób zainteresowanych uczeniem maszynowym.



SZYMON MOLIŃSKI

Inżynier danych i kierownik techniczny w firmie Data Lions. Od studiów chciał łączyć swoje zainteresowanie inżynierią kosmiczną oraz medycyną techniczną i udało się – w dużej mierze dzięki Pythonowi. Jest członkiem trójmiejskiej społeczności Hackerspace. Natok myśli programisty ucisza, tańcząc breakdance.

Nowe trendy i innowacje dla programistów – wywiad z Siddarthą Agarwalem, wiceprezesem ds. strategii i zarządzania produktem w Oracle

Termin DevOps to połącznie dwóch słów: development oraz operations. W ostatnich czasach to bardzo popularna metodyka, ale niewiele osób wie, co się faktycznie kryje za tym terminem. Co powiedziałby Pan wszystkim tym ludziom, którzy uważają, że to tylko moda, która szybko przeminie?

Wszyscy zgadzamy się co do tego, że DevOps to metodyka pozwalająca programistom tworzyć nowe aplikacje i narzędzia szybciej, przy jednoczesnym zachowaniu wysokiego poziomu wydajności oraz jakości. Ta metodyka kładzie nacisk na ścisłą komunikację, a przy tym wzajemne zaangażowanie zarówno programistów, jak i specjalistów odpowiedzialnych za utrzymanie IT. Chodzi tutaj o to, by poprawić zarówno sam proces tworzenia aplikacji, jak i jakość samego produktu. Istnieje dziś jednak bardzo poważny problem w tym devopowym równaniu. Programiści i developerzy aplikacji pracujący w zgodzie z metodyką DevOps bardzo często ponad 50 proc. swojego czasu poświęcają na prace po stronie operacyjnej, zamiast tworzyć i doskonalici samo oprogramowanie. W tym samym czasie firmy wywierają na nich presję, by dostarczali oni nowe rozwiązania szybciej i sprawniej. Nie ma wątpliwości co do tego, że DevOps to innowacyjne podejście, ale aby programiści mogli realizować cele biznesowe, metodyka musi ulec pewnym modyfikacjom. Pozytywne zmiany może przynieść automatyzacja pewnych procesów. Możemy być pewni, że w tym roku programiści będą coraz częściej korzystać z rozwiązań opartych na chmurze.

Jakie technologie chmurowe ma Pan na myśli? Długo przyjdzie nam na nie czekać?

Zmiana jest na wyciągnięcie ręki dzięki trzem wyróżnikom chmurowych platform, dzięki którym możliwe jest stworzenie autonomicznych, samonaprawiających się systemów. Mam na myśli wykorzystanie automatyczne narzędzia, analitykę wykorzystującą uczenie maszynowe i zintegrowane systemy naprawcze. Dzięki gromadzeniu dużych ilości danych w jednym konkretnym magazynie, wykorzystaniu dużej mocy obliczeniowej, uczenia maszynowego oraz algorytmów stworzonych pod konkretne zadania, bazujące na chmurze rozwiązań znaczco ułatwia pracę. Zaimplementowane zaawansowane procedury wykrywania anomalii pozwolą na automatyczne informowanie programistów o tym, że pewne procesy zachodzą niezgodnie z normami. Dzięki technologii machine learningu (pol. uczenie maszynowe) mogą one także znaleźć przyczynę problemu, a w pewnych sytuacjach podjąć właściwe działania naprawcze. To prowadzi nas do stworzenia samonaprawiających i samozarządzających się systemów.

Brzmi jak magia albo fantastyka naukowa. Czy stworzenie takiej technologii rodem z filmu jest możliwe już teraz?

Wspomniane przeze mnie funkcje to nie jest technologia poza naszym zasięgiem. To się już dzieje i jest efektem zintegrowania różnych funkcji, systemów oraz procesów, w formę platformy, która najpierw generuje odpowiednie mechanizmy, następnie podłącza je pod oparty na uczeniu maszynowym silnik analityczny. Uzyskane dane są następnie wykorzystywane po to, by podjąć odpowiednie działania usprawniające lub naprawcze, które proponuje sama platforma. Dostawcy tacy jak Oracle bardzo intensywnie pracują nad tym, by tego typu mechanizmy nie były tylko ciekawostką, ale realnie działającym instrumentem dla współczesnego, nowoczesnego developera aplikacji. Potrzebuję on narzędzi, który uczynią jego produkt nie tylko wydajnym, ale i bezpiecznym dla biznesu. Warto odnotować, że tradycyjne podejście do IT zakładało do tej pory, by oddzielnie monitorować poziomy bezpieczeństwa i wydajność aplikacji. Dzisiejsze technologie zmieniają się jednak bardzo szybko i nie da się już tych dwóch kwestii od siebie oddzielić. W ramach usługi Oracle Management Cloud wszystkie dane wydajnościowe oraz logi bezpieczeństwa są przechowywane w chmurze, która umożliwia wdrożenie systemów do analityki Big Data oraz algorytmy uczenia maszynowego, usprawniające pracę DevOpów. Wszystkie procesy analityczne czy procedury bezpieczeństwa zachodzą tu i teraz, au-

tomatycznie gwarantując zarówno zespołowi developerskiemu, jak i analitykom odpowiednie dane i narzędzia.

Wspomniane uczenie maszynowe stanowi w takim przypadku nie tylko szansę, ale także ogromne wyzwanie dla programistów?

Technologia oparta na uczeniu maszynowym, aby była wydajna i spełniała swoje zadanie, nadal musi być najpierw umiejętnie zaprogramowana w danym środowisku informatycznym, z uwzględnieniem danych, jakie będą analizowane, i rozwiązań, jakich użytkownik będzie oczekwać. To nie jest rozwiązanie uniwersalne. Inaczej napiszemy aplikację do monitorowania bezpieczeństwa, a zupełnie odmiennie podejdziemy do pracy nad systemem do optymalizacji robotów w fabryce. Programiści i developerzy aplikacji będą musieli w najbliższych latach poświęcić wiele czasu na zdobycie dodatkowej wiedzy na temat konkretnych przypadków zastosowania takiego czy innego rozwiązania opartego na uczeniu maszynowym. Projektując daną aplikację, muszą już na wstępnie wiedzieć, jakie dane będą gromadzić, jakie algorytmy zastosować i jakie pytania będą kluczowe w danym przypadku. Przygotowując swoje własne środowisko pracy, jednocześnie pojawi się potrzeba oceny, czy przy stosowaniu olbrzymich ilości danych lepiej pracować na własnych systemach, czy w oparciu o model usługowy SaaS (Software as a Service), w którym dane aplikacje są przechowywane na komputerach dostawcy usługi, a następnie udostępniane użytkownikom przez Internet.

Wspomina Pan o automatyzacji i uczeniu maszynowym jako przyszłości programowania. Ostatnio dużo się mówi o nowych technologiach i ich wykorzystaniu w metodyce DevOps. Jakie zatem nowinki czekają programistów w 2018 roku?

W nadchodzącym roku programiści będą mogli poeksperymentować z wieloma nowymi narzędziami i technologiami takimi jak np. blockchain, chatboty czy infrastruktura bezserwerowa. Chatboty to gorący temat wśród developerów aplikacji ze względu na to, że mogą one zaoferować ciągły, nieprzerwany dostęp do danych, zarówno systemowych, jak i konsumenckich. Programiści stosując coraz to bardziej zaawansowane, oparte na technologiach chmurowych aplikacje, są w stanie stworzyć boty, które rozumieją potrzeby klienta, potrafią podtrzymywać rozmowę w taki sposób, że rozmówca ma wrażenie interakcji z prawdziwą osobą. W tym samym czasie boty współpracują z systemami wsparcia i dostarczają danych do analizy. W 2018 r. branża będzie debatować także, czy architektura bezserwerowa to rewolucja, czy ewolucja dla środowisk IT. To obecnie bardzo pożądana technologia, ale trudno określić, czy stanie się powszechnym standardem, czy metodą działania zarezerwowaną dla niewielkiej grupy niszowych rozwiązań. Myślę jednak, że dla samych programistów kluczowe będzie umiejętnie łączenie różnych bezserwerowych funkcjonalności ze standardowymi rozwiązaniami, tak by możliwe było sprawne przeprowadzanie skomplikowanych operacji. To oczywiście rodzi potrzebę stworzenia warunków progowych, które zdefiniują, jak te funkcje łączyć oraz jak zmniejszyć negatywny wpływ na cały proces, gdy jedno z naszych bezserwerowych narzędzi ulegnie awarii.

A co sędzi Pan o blockchainie? Dojdzie do rewolucji na rynku i technologia ta rozwinię skrzydła także w regulowanych branżach?

Wydaje mi się, że programiści muszą być przygotowani na moment, w którym będą musieli wdrożyć technologię blockchain w klasycznych usługach finansowych czy w systemach do zarządzaniałańcuchem dostaw. Biznes coraz bardziej stawia na takie cechy, jak bezpieczeństwo, niezawodność i wydajność. Decydenci rozumieją, że ich najwyższy poziom można uzyskać dzięki działałom opartym na blockchain. Technologia ta zapewni wielu branżom pełną skalowalność, odporność na zagrożenia oraz bezpieczeństwo i proste mechanizmy integracji z innymi firmowymi systemami korporacyjnymi. To samo w sobie znacznie ułatwi programistom pracę nad odpowiednim użyciem technologii dla celów biznesowych.

Moduły w Javie 9

Java Platform Module System, znany wcześniej jako Project Jigsaw, znacząco wpłynął na organizację maszyny wirtualnej, zaś wdrożenie w bibliotekach i aplikacjach staje się faktem. W tym artykule opisuję, jakie problemy stały przed systemem JPMS i jak je rozwiązano, oddając w nasze ręce kompletny system modułarnego środowiska Javy.

PIEKŁO WYBRUKOWANE KLASAMI

Dotychczas, by zlokalizować definicję klasy, Java używała *class path*, tj. podanej (w manifeście lub poprzez argument) listy ścieżek przeszukiwania. Każda z klas ładowanych do JVM musiała znajdować się w pliku dostępnym z tej listy. Zbiór wszystkich dostępnych tą drogą klas spłaszczały jest do sekwencji, z której – w kolejności wystąpienia – wybierana jest żądana nazwa definicja. Oczywiście, mówimy tu o domyślnym *classloaderze* aplikacji, zaś ambitny programista mógłby napisać własną implementację dobierającą klasy z większą precyzją. Koniec końców gubimy dodatkową charakterystykę, potencjalnie szerszą niż grupowanie pakietami.

JVM ładuje klasy na żądanie. Jeśli żaden *classloader* nie znajdzie klasy, maszyna rzuci wyjątkiem *NoClassDefFoundError*. Kompletność środowiska uruchomieniowego w tym sensie nie jest i nie może zostać zweryfikowana, przez co – w aplikacjach o złożonych zależnościach – trudno przewidzieć scenariusze prowadzące do katastrofy. W myśl praw Murphy'ego taki błąd objawi się podczas ważnej prezentacji, w ramach przetwarzania krytycznego *payloadu*, albo wyklika go strategiczny klient w zaciszu swojego biura piątkowego wieczoru.

Kolejnym problemem jest sytuacja, w której istnieją pod tą samą kwalifikowaną nazwą różne klasy, niekoniecznie ze sobą powiązane. Zawsze zostanie wybrana „pierwsza z brzegu” implementacja. Jeśli to nie ta, której oczekujemy, uruchomienie może się skończyć serią błędów zaciemniających źródło problemu.

Wszystko to definiuje stan zwany *classpath hell* i jest błędem konfiguracji. Odpowiedzią Javy 9 na ten problem jest spójny mechanizm zależności – system modułów, których powiązania i interfejsy są jawnie zadeklarowane w deskryptorze modułu. Teraz JVM już podczas startu weryfikuje środowisko, zapewniając jego spójność, zaś w przypadku niespełnionej zależności zgłasza wyjątek *java.lang.module.FindException* zawierający wszystkie informacje niezbędne do skorygowania deskryptora.

ŚCISŁA HERMETYZACJA

Hermetyzacja przed Javą 9 sprowadzała się do kombinacji pakietów i modyfikatorów dostępu (*private*, *protected*, *package-private*, *public*). Praktyka wykazała, że istnieje model współdzielenia typów, w którym to nie wystarcza. Zwłaszcza w obliczu technik refleksji.

Dotychczasowy mechanizm nie pozwalał na ukrycie klasy przed światem i jednocześnie udostępnienie tej samej klasy do użytku we wszystkich pakietach biblioteki. Takie połączenie *protected* dla kodu z zewnątrz i *public* dla kodu z wewnątrz biblioteki. Między innymi dlatego, że nie było mechanizmu pozwalającego jasno określić granice tejże biblioteki. Pamiętajmy przy tym, że to, co nazywamy tutaj biblioteką w sensie logicznym, może składać się z wielu plików JAR.

Wszystkie pakiety *impl* czy *internal*, jakie spotykamy, są emanacją słabości hermetyzacji przed JPMS. Projektant biblioteki daje nam jasno do zrozumienia, że choć te klasy są publiczne, to nie należy ich używać, chociażby dlatego, że rości sobie prawo do ignorowania kompatybilności pomiędzy wersjami lub żeby ograniczyć możliwość przypadkowej ingerencji w danych.

Całą platformę podzielono na około 100 modułów, dzięki czemu wraz z aplikacją możemy dostarczyć w pełni sprawną maszynę wirtualną skrojoną na miarę, co jest istotne zwłaszcza na urządzeniach o mocno ograniczonych zasobach. Pełna modularyzacja JVM i aplikacji wpływa pozytywnie na jej wydajność, ponieważ graf zależności jasno precyzuje, które komponenty wchodzą w skład dystrybucji i warte są dalszego przetwarzania.

W celu dostosowania JVM, Java 9 wprowadza proces zwany linkowaniem i zupełnie nowe narzędzie: *jlink*. Znajdziemy je w katalogu *bin* instalacji JDK. Dzięki temu poleceniu przygotujemy obraz maszyny wirtualnej zintegrowany z wymaganymi przez aplikację modułami, a ponadto narzędzie wygeneruje skrypty uruchomieniowe dla różnych systemów. Linkowanie jest opcjonalnym procesem zaliczanym do zaawansowanych sztuczek, dlatego wykracza poza ramy artykułu. Polecam lekturę dokumentacji oraz publikacji z odnośników na końcu artykułu.

DESKRYPTOR MODUŁU

Długo wyczekiwanym rozwiązaniem wymienionych problemów jest Java Platform Module System, czyli system modułów. Moduł grupuje logicznie powiązane pakiety typów i ewentualne zasoby, takie jak obrazy czy pliki XML. Pakowany jest w dobrze znany format JAR, ale odróżnia go obecność w katalogu głównym archiwum skompilowanego pliku *module-info.java* czyli deskryptora modułu. Plik ten określa charakterystykę modułu: jego unikalną nazwę, zależności od innych modułów oraz jasno sprecyzowane API udostępniane przez moduł.

Ponieważ w przestrzeni nazw modułów aplikacji pod jedną nazwą może występować jeden moduł, nazwa ta musi być unikalna. Ponadto jeśli projektowany moduł będzie dostępny szerzej, np. jako fragment biblioteki, nazwa powinna być unikalna globalnie. Przyjętą praktyką jest konwencja nazewnica *reverse DNS*, którą dobrze znamy choćby z nazewnictwa pakietów, np. *org.apache.logging*. Przestrzeń nazw modułów jest niezależna od pozostałych przestrzeni nazw, więc technicznie rzecz ujmując, moduł może nosić tę samą nazwę, co dostarczana przez niego klasa czy pakiet. Zdecydowanie lepiej jednak zachować globalną unikalność między tymi przestrzeniami, aby uniknąć wieloznaczności i pomyłek.

Przyjrzyjmy się deskryptoriowi jednego z modułów platformy – modułu *java.prefs* dostarczającego API do obsługi prostej konfiguracji. Plik deskryptora wygląda tak:

Listing 1. Deskryptor modułu java.prefs

```
module java.prefs {
    requires java.xml;
    exports java.util.prefs;
}
```

Ten moduł wymaga API do obsługi plików XML albo inaczej – moduł `java.prefs` zależy od modułu `java.xml`. Bez tej deklaracji kompilator nie odnajdzie wszystkich używanych wewnętrznie typów, a moduł nie zostanie skompilowany lub uruchomienie zakończy się błędem, jeśli skorzystamy z techniki niejawnego ładowania klas.

Moduł używa co prawda API dla XML, ale nie dostarcza go. Oznacza to, że nawet jeśli aplikacja deklaruje zależność od modułu `java.prefs`, to nie ma dostępu do modułu `java.xml` – jest on używany wewnętrznie, ale nie jest dostarczany „w zestawie”. Aby uzyskać dostęp, trzeba jawnie zadeklarować taką zależność.

Co jeśli typy zdefiniowane w jednym module są używane w API drugiego modułu? Czy zawsze trzeba będzie zadeklarować obydwie zależności, aby uniknąć błędów komplikacji? Nie, jest na to sposób. Jeżeli wsparcie jednego modułu jest nierozerlaczne z perspektywy drugiego modułu, to deklaracja zależności jako przechodniej (ang. *transitive*) sprawia, że moduł dostarczany jest wraz z tą zależnością. Taka sytuacja ma miejsce w przypadku modułu `java.sql`, który przechodnio dostarcza m.in. moduł `java.xml`:

Listing 2. Deskryptor modułu java.sql

```
module java.sql {
    requires transitive java.logging;
    requires transitive java.xml;

    exports java.sql;
    exports javax.sql;
    exports javax.transaction.xa;
}
```

Zależność nieprzechodnia (ang. *nontransitive dependency*) służy wyłącznie wewnętrznej implementacji modułu. Zależność przechodnia (ang. *transitive dependency*) jest niezbędna do obsługi dostarczanego API. Domyślnie zależności są nieprzechodnie.

Ponadto mechanizm zależności przechodnich pozwala tworzyć moduły agregujące, tj. zawierające jedynie deskryptor, w którym wyszczególniono wiele zależności poprzez `requires transitive`. Przykładem takiego modułu dostarczanego w ramach platformy jest `java.se`, którego deskryptor z grubsza wygląda tak:

Listing 3. Wyciąg z deskryptora modułu java.se

```
module java.se {
    requires transitive java.desktop;
    requires transitive java.logging;
    requires transitive java.prefs;
    requires transitive java.sql;
    requires transitive java.xml;
    // ... i wiele innych modułów
}
```

Istnieje również dyrektywa `requires static` wskazująca moduł, którego obecność wymagana jest jedynie w czasie komplikacji, co pozwala zadeklarować opcjonalne zależności nie wymagane podczas uruchomienia. Weźmy za przykład sytuację, gdy implementujemy w naszej aplikacji opcjonalne wsparcie dla skryptów, używając silnika Nashorn – możemy ustalić taką zależność:

Listing 4. Przykład deklaracji zależności opcjonalnej

```
module com.example.client {
    ...
    requires static jdk.scripting.nashorn;
    ...
}
```

Podczas uruchomienia obecność danego modułu w systemie możemy zweryfikować kodem zaczerpniętym z dokumentacji JDK:

Listing 5. Przykład odpytywania o moduł

```
Path dir1, dir2, dir3;
ModuleFinder finder = ModuleFinder.of(dir1, dir2, dir3);
Optional<ModuleReference> omref = finder.find("jdk.foo");
omref.ifPresent(mref -> ... );
```

Temat dynamicznego odwoływania się do modułów wykracza poza ten artykuł. Po więcej szczegółów odsyłam do dokumentacji dla klasy `java.lang.ModuleLayer` i do publikacji wspomnianych w odnośnikach na końcu artykułu.

Wymienione dotychczas deskryptory pomijają pewną zależność – z całą pewnością `java.prefs`, `java.sql`, czy każdy z modułów przechodnich `java.se`, wymaga również podstawowych typów języka Java, choćby tych z pakietu `java.lang` czy – rzadziej – `java.net`. Dostarcza je moduł `java.base` – podstawowy moduł platformy. Choć deskryptory pomijają tę zależność, to jest ona dorozumiana, oczywista dla każdego modułu. Można jawnie zapisać deklarację `requires java.base`; nie będzie to błędem, ale też nie jest potrzebne, ponieważ wszystkie moduły mają zagwarantowany dostęp do modułu bazowego.

Do tej pory opisałem jedną stronę medalu dostępności zawartości modułu, czyli deklarowanie zależności pomiędzy modułami. W materiałach anglojęzycznych czasem odnosi się do niej per *readability*, co można przetłumaczyć jako „czytelność”, niemniej uważam, że termin ten jest mało precyzyjny, zaś forma tego artykułu nie wymaga takiego rozróżnienia.

Drugą stroną medalu, tą związaną z hermetyzacją kodu, a – w odniesieniu do *readability* – zwaną *accessibility*, jest eksportowanie pakietów. Na nic nam zależność od modułu, który nie eksportuje nic, choćby przechodnio. Pakiety staną się dostępne dla innych dopiero wówczas, gdy w deskryptorze jawnie zaznaczymy deklarację eksportu. Pakiety podrzędne nie są eksportowane automatycznie – każdy wymaga osobnej deklaracji.

Gwoli ścisłości wspomnieć muszę, że istnieje możliwość wyeksportowania całego drzewa pakietów, używając zapisu `exports mypackage.*;` – w praktyce jednak zaciemnia on deskryptor modułu i może być źródłem wielu komplikacji, np. konfliktu nazw pakietów, ponieważ JPMS wymusza unikalność nazwy pakietów – nie jest zatem możliwe współdzielienie tego samego pakietu przez wiele modułów.

Tylko publiczne klasy eksportowanych pakietów będą dostępne dla innych modułów. Jeżeli w pakiecie istnieje typ o modyfikatorze dostępu innym niż `public`, tj. jeśli pozostawimy go jako `package-private` lub ograniczymy dostęp jeszcze bardziej rygorystycznym modyfikatorem `protected` lub `private`, to nie będzie on już osiągalny w żadnym innym module. Innymi słowy – jeśli klasa eksportowana go pakietu nie jest jawnie publiczna, to dostęp z zewnątrz ograniczony zostanie przez tradycyjny mechanizm kontroli dostępu. Kla-

sy, których pakietów nie wyeksportowano, pozostają niedostępne poza modułem niezależnie od innych czynników. Naruszenie tych zasad wykryte w czasie komplikacji zakończy się jej błędem, zaś podczas uruchomienia wyrzucony zostanie wyjątek `IllegalAccessError`. Hermetyzacji tej nie złamiemy refleksją, ponieważ nie można oczekiwać dłużej nieskrepowanych niczym efektów wywołania `setAccessible(true)` jak dotychczas. Należy dodać, że JPMS wymusza, aby graf zależności był acykliczny, tzn. że nie jest możliwe zapętlenie zależności poprzez ich wzajemną deklarację, np. moduł A wymaga modułu B, który wymaga modułu A.

Spójrzmy jeszcze raz na deskryptor modułu `java.sql` – eksportuje on publiczną zawartość trzech pakietów:

Listing 6. Wyciąg z deskryptora modułu `java.sql`

```
module java.sql {  
    ...  
    exports java.sql;  
    exports javax.sql;  
    exports javax.transaction.xa;  
}
```

W skrajnych przypadkach może zajść potrzeba eksportu wewnętrznej implementacji wybranym modułom. Taki rodzaj precyzyjnego eksportu nazywa się eksportem kwalifikowanym (ang. *qualified export*). Wskazany pakiet pozostanie niedostępny dla modułów innych niż wyszczególnione w deklaracji (wiele modułów rozdzielimy przecinkiem), nawet jeśli zadeklarujemy zależność względem takiego modułu. Deskryptor modułu `java.xml` dostarcza przykładową deklarację:

Listing 7. Wyciąg z deskryptora modułu `java.xml`

```
module java.xml {  
    ...  
    exports com.sun.xml.internal.stream.writers to java.xml.ws;  
    ...  
}
```

Należy unikać stosowania deklaracji tego typu. Mechanizm ten łamie elastyczność wzorca *producer-consumer*, ponieważ odniesienia do konsumentów stają się częścią deskryptora modułu producenta. Został wprowadzony na potrzeby modularizacji dotychczasowego monolitu, jakim była biblioteka `rt.jar` w poprzednich wydaniach Javy, gdyż okazało się, że wiele modułów platformy zmuszonych jest do używania szczegółów implementacyjnych innych modułów.

Silna hermetyzacja wprowadzona wraz z JPMS, jak już wspomniałem, odcięnęła się również na dostępie przez refleksję. Gdyby pozostawić taką sytuację, byłoby to dość surowe rozwiązanie dla wielu frameworków pokroju Hibernate, Spring i innych, dla których badanie i operowanie refleksją jest podstawą pracy. Wprowadzono zatem dyrektywę `opens`, w której podajemy nazwę otwartego pakietu. Powoduje ona, że dany pakiet pozostaje otwarty na dostęp poprzez refleksję (również do typów i składowych niepublicznych) podczas uruchomienia, ale nie jest dostępny w czasie komplikacji, ponieważ zwyczajnie nie jest to eksport. Alternatywnie można otworzyć cały moduł, poprzedzając jego deklarację odpowiednim modyfikatorem: `open module xyz { ... }`.

Rozważna mieszanka zależności, eksportów i dotychczasowych modyfikatorów dostępu zwiększa bezpieczeństwo platformy, gdyż ogranicza liczbę klas wystawionych na potencjalne ataki. Ponadto silna hermetyzacja pomaga w uzyskaniu i utrzymaniu czystszego

i bardziej logicznego projektu. Granice modułów i złożonych z nich bibliotek pozostają ściśle określone, zaś szczegóły implementacyjne zachowują wewnętrzny charakter i poziom dostępu.

USŁUGI

Java 9 nadaje nowego życia konceptowi usług, tj. `services`. Nie jest on co prawda całkiem nowym konceptem – w starszych wersjach istniał mechanizm `ServiceLoader` i był całkiem szeroko stosowany – jednak w ujęciu modularnym nabiera pełni.

Usługa składa się z dwóch elementów: interfejsu i co najmniej jednej implementacji. Interfejs usługi jest zwykle dystrybuowany w dedykowanym module, który zawiera jedynie tenże interfejs oraz wszelkie inne typy i zależności przechodnie niezbędne do jego obsługi z zewnątrz. Każda implementacja usługi jest zaś dostarczana w osobnym, dedykowanym module.

Aplikacja czy dowolny inny moduł mogą zależeć od modułu zawierającego interfejs usługi, nie znając właściwego modułu, który dostarcza jej implementacji. Platforma na podstawie `module path` znajduje i rozwiązuje taki moduł podczas uruchomienia.

Moduł interfejsu usługi nie jest deklarowany w szczególny sposób – ot, zwykła deklaracja eksportu, np.:

Listing 8. Deskryptor przykładowego modułu interfejsu usługi

```
module com.example.service {  
    exports com.example.service;  
}
```

Poza nazwą modułu, która nie stanowi nic więcej poza czytelnym identyfikatorem, nie znajdziemy tu żadnego odniesienia do konceptu usług. Eksportujemy jedynie API i nie wskazujemy, gdzie usługa została zaimplementowana. Sam interfejs też nie jest szczególnym, dlatego pomijam przytaczanie przykładu.

Cała rzecz dzieje się w module implementującym usługę. Po pierwsze, taki moduł musi oczywiście zależeć od modułu udostępniającego interfejs. Po drugie, co równie oczywiste, moduł musi zawierać klasę implementującą. Po trzecie, mamy do dyspozycji specjalną dyrektywę, która informuje system modułów, że tu oto znajduje się implementacja interfejsu:

Listing 9. Deskryptor przykładowego modułu implementacji usługi

```
module com.example.service.dummy {  
    requires com.example.service;  
  
    provides com.example.service.MyService  
        with com.example.service.dummy.MyServiceImpl;  
}
```

Aby skorzystać z takiej usługi, moduł kliencki również musi zatrzymać specjalną dyrektywę wskazującą w swoim deskryptorze na interfejs usługi, z której będzie korzystał:

Listing 10. Deskryptor przykładowego modułu klienckiego usługi

```
module com.example.client {  
    requires com.example.service;  
  
    uses com.example.service.MyService;  
}
```

Tak naprawdę jawne deklarowanie zależności od modułu interfejsu nie jest tu wymagane – dodałem je dla jasności. Sama dyrektywa `uses` zawiera wystarczająco informacji, by zlokalizować inter-

fejs podczas uruchomienia. Dla dopełnienia obrazu całości samo użycie takiej usługi wygląda tak:

Listing 11. Przykład pozyskania instancji implementujących usługę

```
Iterable<MyService> services = ServiceLoader.load(MyService.class);
```

Zwrócony iterator zawiera instancje wszystkich implementacji danej usługi, jakie odnaleziono w obrębie *module path* – to od nas zależy, której implementacji użyjemy. Na przykład możemy wybrać pierwszą z brzegu, ale jeśli zależy nam na konkretnej charakterystyce implementacji, to – zgodnie ze sztuką – powinniśmy uwzględnić tę charakterystykę w samym interfejsie usługi, tak aby możliwa była selekcja instancji, odpowiadając jej o atrybuty i filtrując te pozostałe w naszym zainteresowaniu.

MODUŁY A WERSJONOWANIE

System modułów JPMS rozwiązuje inne problemy niż np. Maven i na tym etapie jego implementacji nie było potrzeby wprowadzenia analogicznego wersjonowania modułów. Systemy te nie konkurują ze sobą bezpośrednio, a raczej się uzupełniają. To, co nazываемy wersjonowaniem modułów w JPMS, odnosi się do techniki dostarczania modułu wspierającego więcej niż jedno wydanie Javy.

Począwszy od Javy 9, możliwe jest utworzenie pojedynczego pliku JAR zawierającego wersje dedykowane różnym wydaniom Javy. Dzięki temu możemy dostarczyć kompletny moduł czerpiący benefity z nowszych wydań, nie wymuszając jednocześnie od jego użytkowników przesiadki. Tak wygląda struktura przykładowego pliku JAR dla wielu wydań:

Listing 12. Przykładowa struktura pliku JAR dla wielu wydań

```
META-INF/
 MANIFEST.MF
versions/
 10/
 example
 9/
 example
example
```

Pakiet `example` w katalogu głównym pliku JAR dostarcza klasy skompilowane dla wydań starszych niż Java 9, a powód jest prosty – wcześniejsze wersje nie znają takiej struktury, więc kod należy dostarczyć z dotychczasową strukturą katalogów. Oczywiście oznacza to, że możemy jednocześnie dostarczyć kod dla jednej tylko wersji sprzed wydania 9.

Plik manifestu `MANIFEST.MF` wymaga specjalnego wpisu wskazującego nowszym maszynom wirtualnym, że mają do czynienia z dystrybucją dla wielu wydań:

Listing 13. Wpis w manifeście informujący o wielowydaniowej naturze pliku JAR

```
Multi-Release: true
```

Katalog `versions` w katalogu `META-INF` zawiera podkatalogi, których nazwa musi odpowiadać numerowi wydania, dla którego przeznaczona jest zawartość.

NA POGRANICZU DWÓCH ŚWIATÓW

Oczywiście nie wszystkie biblioteki zostały już zmodularyzowane, choć od czasu wydania stabilnego, publicznie dostępnego JDK 9 minęło już ponad pół roku, zaś wersje *General Availability* były dostępne wcześniej. Niektóre z tych bibliotek tak bardzo zależą od szczegółów implementacyjnych platformy, że konieczne staje się ich przeprojektowanie. Te aspekty dotyczą głównie użycia klasy `sun.misc.Unsafe`, zaś szczegółowy opis sytuacji i możliwych rozwiązań wykracza poza rama tego artykułu. Twórcy niektórych bibliotek nie dostrzegają problemu, gdy ich kod przypadkiem działa pod Javą 9 dzięki wsparciu mechanizmów opisanych w dalszych akapitach. Takie podejście jest właśnie weryfikowane przez wydanie JDK 10, które w bardziej rygorystyczny sposób podchodzi do obsługi modułów. To, co nas w tym miejscu interesuje, to podejście, jakie możemy zastosować dziś, by używać kodu, który nie był pisany z myślą o modułach.

W dalszym ciągu w Javie 9 możemy używać mechanizmu *class path*, ale zaszły tu pewne zmiany. Zawartość wszystkich plików JAR – niezależnie, czy są one modularne, czy nie – znajdujących się w zasięgu *class path* ląduje w specjalnym module bez nazwy, tj. *unnamed module*. Eksportuje on wszystkie pakiety i posiada dostęp do wszystkich innych modułów aplikacji. Nie posiada nazwy, przez co żadne moduły nie mogą ustanawiać zależności względem niego.

Jeżeli klasa lub cały moduł jest dostępny zarówno poprzez mechanizm *module path*, jak i *class path*, to pierwotnie ma oczywiście źródło zmodularyzowane. Każdy plik JAR bez deskryptora modułu znaleziony w ramach *module path* staje się tzw. modułem automatycznym, tj. *automatic module*. Pozwala to na swego rodzaju automatyczną modularyzację niemodularnych dotychczas bibliotek. Automatyczny moduł w swoich zależnościach posiada wszystkie nazwane, w tym inne automatyczne moduły dostępne w aplikacji, a także – w przeciwnieństwie do modułów nazwanych nieautomatycznych, czyli tych prawdziwych – posiada dostęp do klas składających się na wspomniany wyżej nienazwany moduł *unnamed module*. Moduły automatyczne również eksportują wszystkie zawarte w nich pakiety, dzięki czemu inne moduły mogą się do nich odnosić poprzez zależności, używając automatycznie wygenerowanej nazwy. Taka nazwa modułu automatycznego powstaje na bazie nazwy pliku JAR – jeśli będzie to np. `com-example-jpms.jar`, to nazwę modułu będzie `com.example.jpms`. Jeśli nazwa pliku zawiera numer wersji, to jest on obcinany. Znaki takie jak myślnik nie mogą być częścią nazwy modułu, więc są konwertowane na kropki.

KOMPILACJA, PAKOWANIE I URUCHOMIENIE

No dobrze, przedstawiłem cechy modułów i szczegóły ich deklaracji, ale jak tak naprawdę użyć modułów?

Analogicznie do *class path*, które precyzowało obszar wyszukiwania klas, *module path* dostarcza lokalizacji, w których mogą występować moduły (lub katalogi zawierające moduły) potrzebne do komplikacji i uruchomienia. Oczywiście ścieżki listujemy w zależności od systemu operacyjnego, rozdzielając lokalizacje separatorem : (Unix) lub ; (Windows). Wszystkim polecaniom linii komend, które wspierają obsługę modułów – w tym `javac` i `java` – taką listę wyszukiwania modułów podajemy parametrem `--module-path` lub `-p`. Wszystkie artefakty znalezione w *module path* zostaną prze-

konwertowane na moduły wspomnianym we wcześniejszej części artykułu mechanizmem – nawet jeśli są to tradycyjne pliki JAR.

Dalej przedstawię pokrótkie polecenia środowiska Java. Nie będę opisywać tu szczegółowo ich składni, którą można znaleźć bez trudu w Internecie. Środowiska IDE też już całkiem dobrze wspierają JPMS, więc być może ta wiedza nie będzie potrzebna czytelnikowi, ale cóż to za programista, który nie potrafi skompilować kodu własnoręcznie.

Zakładając, że nasz moduł nosi nazwę com.example, poleceniem komplikacji jest stare, dobre javac:

Listing 14. Kompilacja źródeł modułu

```
$ javac  
--module-path katalog_modulow  
-d target/com.example  
${source-files}
```

Nowością jest tu oczywiście wspomniany parametr --module-path. Drugim argumentem jest docelowa lokalizacja komplikacji, a trzecim jest seria plików źródłowych.

Pakowanie odbywa się poprzez polecenie jar:

Listing 15. Pakowanie skompilowanego modułu

```
$ jar --create  
--file mymodules/com-example.jar  
--main-class com.example.MyApplication  
-C target/com.example
```

Tutaj ciekawostką jest parametr --main-class, który pozwala na zadeklarowanie głównej klasy aplikacji bez użycia pliku manifestu. Mając tak zadeklarowaną klasę główną, możemy uruchomić aplikację prostym poleceniem:

Listing 16. Uruchomienie modułu posiadającego klasę główną

```
$ java  
--module-path mymodules  
--module com.example
```

Ścieżek komplikacji i kombinacji środowiskowych istnieje oczywiście tyle, ile projektów, dlatego uważam, że na tym poziomie wykorzystałem powyższym opisem temat. Zintegrowane edytory załatwiają sprawę za nas, niemniej zachęcam do lektury dokumentacji wymienionych polecen zwłaszcza w kontekście JPMS.

PODSUMOWANIE

Java Platform Module System był długo wyczekiwany. Wiązano z nim wielkie nadzieje, m.in. to, że połączy świat OSGi i Mavena w jedno standardowe rozwiązanie. Tak się nie stało, choć pozostałe oczekiwania zostały spełnione z nawiązką. OSGi i Maven rozwiązują zgoła odmienne problemy, zresztą wcale nie jest powiedziane, że JPMS nie będzie ewoluował, czerpiąc z nich, co najlepsze. Póki co, zrobiono miły krok w zarządzaniu tak dużym projektem jak JDK, a te narzędzia i techniki udostępniono pozostałym programistom. Trudno jest na chwilę obecną przewidywać, gdzie nas to zaprowadzi, ale z całą pewnością tendencja jest słuszna – odchudzone środowisko, ścisła hermetyzacja, poprawione bezpieczeństwo i wydajność.

Starałem się zawrzeć w artykule jak najwięcej szczegółów odnośnie tworzenia modularnych aplikacji w już nie tak nowej Javie 9. Być może przeoczyłem coś lub pewne aspekty dałoby się wyjaśnić lepiej. Dlatego zachęcam do odwiedzenia w pierwszej kolejności wymienionych odnośników i dalszego przeszukiwania Internetu.

W sieci

- ▶ Strona główna projektu Jigsaw na OpenJDK: <http://openjdk.java.net/projects/jigsaw/spec/>
- ▶ Dokumentacja modułu java.base: <https://docs.oracle.com/javase/9/docs/api/java.base-summary.html>
- ▶ Dokumentacja modułu java.se: <https://docs.oracle.com/javase/9/docs/api/java.se-summary.html>
- ▶ Świetna książka Sandera Maka i Paula Bakkeria wyczerpująca tematykę: <https://javamodularity.com/>
- ▶ Kolejna wyczerpująca publikacja, tym razem Nicolaia Parloga: <https://www.manning.com/books/the-java-module-system>
- ▶ Artykuł Paula Deitela dla Oracle poruszający też historię implementacji: <https://www.oracle.com/corporate/features/understanding-java-9-modules.html>
- ▶ Artykuł Jakuba Jenkova na jego blogu: <http://tutorials.jenkov.com/java/modules.html>
- ▶ Artykuł Nicolaia Parloga na jego blogu: <https://blog.codefx.org/java/java-module-system-tutorial/>
- ▶ Krótkie, tabelaryczne podsumowanie tematu: <https://labs.consol.de/development/2017/02/13/getting-started-with-java9-modules.html>



KAMIL BECMER
kamil.becmer@gmail.com

Pracuje w firmie Sii Sp. z o.o., w projekcie dla PZU S.A. w zespole utrzymania systemu rozliczeń. Programuje od dzieciństwa, od 8 lat w Javie, od kilku również w Gosu. W wolnych chwilach praktykuje C++ i poznaje elektronikę. Pasjonat systemów „inteligentnego domu”, technologii druku 3D i wszelkich gadżetów rozwiązujących problemy pierwszego świata.

SZKOLENIA Z JĘZYKA **C++**

POZNAŃ 11.06.2018

KRAKÓW 13.06.2018

WARSZAWA 18.06.2018

KATOWICE 27.06.2018

SZKOLENIA Z JĘZYKA **PYTHON**

WARSZAWA 04.06.2018

ŁÓDŹ 12.06.2018

GLIWICE 20.06.2018

WROCŁAW 25.06.2018

- PROGRAMOWANIE PROCEDURALNE
- PROGRAMOWANIE OBIEKTOWE
- WZORCE PROJEKTOWE

- PODSTAWY PYTHONA
- PROGRAMOWANIE I TWORZENIE APLIKACJI



KOMPLEKSOWO SZKOLIMY NOWOCZESNY BIZNES



SPRAWDŹ SZKOLENIA NA WWW.HELIONSZKOlenia.PL

Zarządzanie pamięcią w C++17

Pamięć jest jednym z ważniejszych zasobów każdego programu komputerowego. Reprezentacja danych programu i ich układ w pamięci ma znaczący wpływ na szybkość wykonywania obliczeń [0].

Język C++ od samego początku swojego istnienia oferował programiście rozbudowany zestaw narzędzi do automatycznego, jak i ręcznego zarządzania pamięcią. Początkowo, udostępniając C API do dynamicznego zarządzania pamięcią (`malloc/realloc/free`) oraz definiując operatory `new/delete`, a także umożliwiając formę automatycznego zarządzania pamięcią dzięki stosowaniu RAII. Wraz z kolejnymi wersjami C++ usprawniono mechanizm pozykiwania i zwalniania pamięci, dodając tzw. intelligentne wskaźniki, które *de facto* pozwalają na całkowite wyeliminowanie manualnego zarządzania pamięcią. Naprawiono także część problemów odziedziczonych po C, dla przykładu zdefiniowano operator `delete` z informacją o rozmiarze zaalokowanej pamięci, dzięki czemu można zaimplementować alokator z lepszym układem pamięci.

Listing 1. Deklaracje globalnych operatorów `delete` z dodatkowym parametrem informującym o rozmiarze zaalokowanej pamięci

```
void operator delete( void* ptr, std::size_t sz );
void operator delete[]( void* ptr, std::size_t sz );
```

W ramach poprawek dodano także wsparcie dla alokatorów ze statem, a także umożliwiono zdefiniowanie wymagań wyrównania pamięci dla konkretnego typu oraz obiektu dzięki słowi kluczowemu `alignas`.

Pomimo tych wszystkich zmian zarządzanie pamięcią w C++14 nadal było dalekie od ideału. Wymieńmy tylko kilka z nierożwiązańskich problemów:

- » dynamiczna alokacja z zadanym wyrównaniem pamięci,
- » niezależność alokatora od typu obiektu, który go używa,
- » wsparcie dla realokacji,
- » operatory `new` i `delete` działające w ramach ograniczonego zakresu (niezależne od typu),
- » narzędzia debugowe, jak choćby oznaczanie pamięci, metryki.

Wraz z wydaniem najnowszej odsłony języka C++17 część z powyższych problemów została w pełni, bądź częściowo, rozwiązana.

W niniejszym artykule zostaną opisane najważniejsze zmiany w ramach zarządzania pamięcią, które wprowadzono w C++17. W dalszej kolejności zostanie porównane to, co oferuje język C++ z wymaganiami, które stawia się przed systemami wymagającymi dużej kontroli nad pamięcią. Za przykład takiego systemu posłuży system zarządzania pamięcią w grze komputerowej.

W tym miejscu warto także odpowiedzieć na pytanie, dlaczego w ogóle temat zarządzania pamięcią w C++ jest nadal istotny. Zwłaszcza biorąc pod uwagę to, że obecnie ręczne zarządzanie pamięcią w C++ uznaje się za antywzorzec, a za idiomatyczny kod przyjmuje się taki, który dynamiczną alokację sprowadza do użycia `std::make_unique` oraz rzadziej `std::make_shared`¹.

1. Należy pamiętać, że `std::make_shared` pomimo optymalizacji pamięciowej (pojedyncza alokacja na obiekt i blok kontrolny) nadal jest kosztowną operacją. Wynika to ze specyfikacji `std::shared_ptr` i wiąże się to głównie z kosztem atomowego liczenia referencji. Dlatego też w wielu przypadkach dużo lepszym rozwiązaniem jest użycie `intrusive_ptr`.

Warto zwrócić uwagę na to, że poszczególne alokacje (podobnie jak sama pamięć) różnią się od siebie, m.in. kontekstem wywołania, rozmiarem czy też czasem życia. Dlatego też stosowanie dla wszystkich przypadków jednego, generycznego alokatora pamięci nie jest efektywne, ponieważ taki alokator musi między innymi:

- » obsługiwać alokacje dla dowolnego rozmiaru,
- » zapewniać bezpieczny dostęp do alokacji i dealokacji z wielu wątków,
- » radzić sobie z fragmentacją pamięci.

Ponadto programista pisząc program, powinien znać charakterystykę pamięci², której w danej chwili potrzebuje. Zatem powinien być w stanie dobrać odpowiedni sposób alokacji dla konkretnego przypadku. Innymi powodami przemawiającymi za ręcznym zarządzaniem pamięcią mogą być wymagania *strict* systemowe. Dana platforma może zabraniać dynamicznej alokacji, bądź może stawać różnego rodzaju wymagania odnośnie wyrównania (ang. *alignment*), lub rozmiaru pamięci.

DYNAMICZNA ALOKACJA Z ZADANYM WYRÓWNANIEM

Do niedawna język C++ nie pozwalał na dynamiczną alokację pamięci z zadanym wyrównaniem. C++11 umożliwiał jedynie zdefiniowanie wymagania wyrównania dla typu bądź obiektu poprzez użycie słowa kluczowego `alignas`. Tym niemniej język nie oferował mechanizmu, który pozwalałby na użycie wyrównania podczas dynamicznej alokacji pamięci.

Listing 2. Przykład problemu z dynamiczną alokacją z zadanym wyrównaniem

```
class alignas(16) float4
{
    float f[4];
};

// nie ma gwarancji, że tablica zostanie poprawnie wyrównana
float4 *p = new float4[1000];
```

Problem ten obchodziło poprzez wołanie funkcji z zewnętrznego API, np. `_aligned_malloc`, `_aligned_realloc`, `posix_memalign`. Sytuacja zmieniła się w C++17, gdzie zaimplementowano rozwiązanie z C11, jak i zdefiniowano nowe wersje operatorów `new` oraz `delete`.

`std::aligned_alloc`

Znana z C11 funkcja do dynamicznej alokacji pamięci z podanym wyrównaniem została dodana do nagłówka `<cstdlib>`.

2. Jeśli nie zna charakterystyki, to najprawdopodobniej nie rozumie danych, na których działa, zatem nie rozumie problemu, który stara się rozwiązać. Wówczas powinien więc korzystać z popularnych idiomów.

Listing 3. Deklaracja funkcji aligned_alloc

```
void* aligned_alloc( std::size_t alignment, std::size_t size );
```

Funkcja ta w przypadku powodzenia zwraca wskaźnik do zaallokowanej, niezainicjowanej pamięci, w przeciwnym wypadku zwraca pusty wskaźnik.

Standard definiuje, że powyższa funkcja może być bezpiecznie wołana z wielu wątków. Warto dodać, że wymagania odnośnie wartości wyrównania pamięci są zdeterminowane przez wymagania implementacyjne. Dla przykładu implementacja POSIXowa wymaga, by alignment był potęgą liczby dwa oraz wielokrotnością sizeof(void*). Dodatkowo zadany rozmiar alokacji musi być wielokrotnością wyrównania.

Operatory new/delete z wyrównaniem pamięci

Najnowszy standard C++ zdefiniował nowe operatory do dynamicznej alokacji pamięci z wyrównaniem pamięci [1].

Listing 4. Deklaracja operatorów new i delete z dodatkowym parametrem wyrównania

```
void* operator new( std::size_t count, std::align_val_t al );
void* operator new[]( std::size_t count, std::align_val_t al );
void* operator new( std::size_t count, std::align_val_t al,
    const std::nothrow_t& );
void* operator new[]( std::size_t count, std::align_val_t al,
    const std::nothrow_t& );
void* operator new( std::size_t count, std::align_val_t al,
    user-defined-args... );
void* operator new[]( std::size_t count, std::align_val_t al,
    user-defined-args... );
void* T::operator new( std::size_t count, std::align_val_t al );
void* T::operator new[]( std::size_t count, std::align_val_t al );
void* T::operator new( std::size_t count, std::align_val_t al,
    user-defined-args... );
void* T::operator new[]( std::size_t count, std::align_val_t al,
    user-defined-args... );

void operator delete( void* ptr, std::align_val_t al );
void operator delete[]( void* ptr, std::align_val_t al );
void operator delete( void* ptr, std::size_t sz,
    std::align_val_t al );
void operator delete[]( void* ptr, std::size_t sz,
    std::align_val_t al );
void operator delete( void* ptr, std::align_val_t al,
    const std::nothrow_t& tag );
void operator delete[]( void* ptr, std::align_val_t al,
    const std::nothrow_t& tag );
void T::operator delete( void* ptr, std::align_val_t al );
void T::operator delete[]( void* ptr, std::align_val_t al );
void T::operator delete( void* ptr, std::size_t sz,
    std::align_val_t al );
void T::operator delete[]( void* ptr, std::size_t sz,
    std::align_val_t al );
```

Wyrównanie pamięci zostało zdefiniowane jako scoped enum o rozmiarze std::size_t, tak by nie kolidować z istniejącymi już przeładowaniami placement new/delete, które przyjmują dodatkowy parametr w postaci liczby całkowitej.

Listing 5. Definicja std::align_val_t

```
enum class align_val_t : std::size_t {};
```

Należy zauważyć, że powyższe przeładowania zostaną wywołane tylko wtedy, gdy wyrównanie dla danego typu będzie większe niż domyślne wyrównanie zdefiniowane przez makro __STDCPP_DEFAULT_NEW_ALIGNMENT__. Makro to wyraża zależną od implementacji, mieszącą się w std::size_t, domyślną wartość wyrówna-

nia dla wywołań operator new(std::size_t) oraz operator new[](std::size_t). Z reguły wartość ta odpowiada domyślnemu wyrównaniu std::malloc, tj. 8 bajtów w przypadku architektury x86 i 16 bajtów dla x64. Niektóre kompilatory pozwalają na nadpisanie domyślnego wyrównania, dla przykładu w clangu można to zrobić, podając parametr komplikacji -fnew-alignment=N, gdzie N to nowa wartość dla domyślnego wyrównania³.

Prześledźmy kilka przykładów z nowymi operatorami new i delete.

W Listingach 6-8 zostały przedstawione przykłady dla globalnych operatorów new i delete.

Listing 6. Definicje wybranych, globalnych operatorów new i delete

```
void* operator new(std::size_t size)
{
    std::printf("operator new with default alignment '%zu'\n",
        static_cast<std::size_t>(__STDCPP_DEFAULT_NEW_ALIGNMENT__));
    auto ptr = std::aligned_alloc(
        static_cast<std::size_t>(__STDCPP_DEFAULT_NEW_ALIGNMENT),
        size);
    return ptr ? ptr : throw std::bad_alloc{};
}

void* operator new(std::size_t size, std::align_val_t align)
{
    std::printf("operator new with custom alignment '%zu'\n",
        static_cast<std::size_t>(align));
    auto ptr = std::aligned_alloc(
        static_cast<std::size_t>(align),
        size);
    return ptr ? ptr : throw std::bad_alloc{};
}

void operator delete(void* ptr) noexcept
{
    std::printf("operator delete with default alignment '%zu'\n",
        static_cast<std::size_t>(__STDCPP_DEFAULT_NEW_ALIGNMENT));
    std::free(ptr);
}

void operator delete(void* ptr, std::align_val_t align) noexcept
{
    std::printf("operator delete with custom alignment '%zu'\n",
        static_cast<std::size_t>(align));
    std::free(ptr);
}
```

Listing 7. Definicje kilku struktur pomocniczych

```
template<typename T>
bool is_aligned(T* ptr, std::size_t alignment)
{
    assert(ptr != nullptr);
    assert(alignment > 0 && (alignment & (alignment - 1)) == 0);
    return reinterpret_cast<std::uintptr_t>(ptr) &
        (alignment - 1) == 0;
}

struct foo {};
struct alignas(32) foo_aligned_as_32 {};
struct alignas(4) foo_aligned_as_4 {};
```

Listing 8. Przykłady dynamicznej alokacji dla wcześniej zdefiniowanych typów

```
{
    auto ptr = new foo;
    assert(is_aligned(ptr, __STDCPP_DEFAULT_NEW_ALIGNMENT__));
    delete ptr;
}

{
    auto ptr = new foo_aligned_as_32;
    assert(is_aligned(ptr, 32));
    delete ptr;
}
```

3. Autor przestrzega przed nadpisywaniem domyślnego wyrównania dla dynamicznej alokacji, gdyż może to pociągać ze sobą nieoczekiwane i negatywne skutki. Zwłaszcza gdy używany jest domyślny aloktor.

```

{
    auto ptr = new foo_aligned_as_4;
    assert(is_aligned(ptr, __STDCPP_DEFAULT_NEW_ALIGNMENT__));
    delete ptr;
}

{
    auto ptr = ::operator new(
        sizeof(foo_aligned_as_4),
        static_cast<std::align_val_t>(alignof(foo_aligned_as_4)));
    auto typed_ptr = ::new (ptr) foo_aligned_as_4;
    assert(is_aligned(typed_ptr, alignof(foo_aligned_as_4)));
    typed_ptr->~foo_aligned_as_4();
    ::operator delete(
        ptr,
        static_cast<std::align_val_t>(alignof(foo_aligned_as_4)));
}

```

Zacznijmy od pierwszego, najprostszego przypadku z typem `foo`. Zauważmy, że typ ten ma 1-bajtowe wyrównanie, więc mniejsze od `__STDCPP_DEFAULT_NEW_ALIGNMENT__`. Zatem wyrażenie `new foo` w konsekwencji spowoduje wywołanie `void* operator new(std::size_t size)`. Na podobnej zasadzie zadziała wyrażenie `delete ptr` dla tego przypadku.

W następnej kolejności mamy alokację pamięci dla typu `foo_aligned_as_32`. W tym miejscu warto dodać, że standard C++ określa wsparcie dla wyrównania większego od maksymalnej wartości wyrównania `alignof(std::max_align_t)` dla największego typu skalarnego jako zależne od implementacji. Na potrzeby tego przypadku założymy, że działamy na platformie, która na to pozwala. `foo_aligned_as_32` ma wyrównanie 32-bajtowe, zatem większe od `__STDCPP_DEFAULT_NEW_ALIGNMENT__`, dlatego też wyrażenie `new foo_aligned_as_32` wywoła `void* operator new(std::size_t size, std::align_val_t align)`. Analogicznie dla `delete ptr` zostanie wybrany operator `delete` z wyrównaniem.

Zauważmy, że dla wszystkich typów z mniejszym wyrównaniem niż `__STDCPP_DEFAULT_NEW_ALIGNMENT__` wyrażenie `new T` spowoduje wywołanie operatora `new` bez wyrównania. W każdej chwili jednak jesteśmy w stanie wywołać jawnie odpowiedni operator `new/delete`. Należy jednak pamiętać o tym, że musimy wtedy także wywołać `placement new` oraz destruktor, jeśli nie mamy do czynienia z obiektem typu POD. Przykład takiego użycia został pokazany, jako ostatni, w Listingu 8.

Rozpatrzmy jeszcze kilka przykładów dla operatorów `new/delete` zdefiniowanych w obrębie typu.

Listing 9. Definicje przykładowych struktur z różnymi operatorami new/delete

```

struct foo_with_default_aligned_memory_operators
{
    void* operator new(std::size_t size);
    void operator delete(void* ptr);
};

struct alignas(32)
foo_aligned_as_32_with_default_aligned_memory_operators
{
    void* operator new(std::size_t size);
    void operator delete(void* ptr);
};

struct alignas(32)
foo_aligned_as_32_with_over_aligned_memory_operators
{
    void* operator new(std::size_t size, std::align_val_t align);
    void operator delete(void* ptr, std::align_val_t align);
};

struct foo_with_over_aligned_memory_operators
{
    void* operator new(std::size_t size, std::align_val_t align);
    void operator delete(void* ptr, std::align_val_t align);
};

```

Listing 10. Przykłady dynamicznej alokacji dla wcześniej zdefiniowanych typów

```

{
    auto ptr = new foo_with_default_aligned_memory_operators;
    assert(is_aligned(ptr, __STDCPP_DEFAULT_NEW_ALIGNMENT__));
    delete ptr;
}

{
    auto ptr =
        new foo_aligned_as_32_with_default_aligned_memory_operators;
    assert(is_aligned(ptr, __STDCPP_DEFAULT_NEW_ALIGNMENT__));
    delete ptr;
}

{
    auto ptr =
        new foo_aligned_as_32_with_over_aligned_memory_operators;
    assert(is_aligned(
        ptr,
        alignof(
            foo_aligned_as_32_with_over_aligned_memory_operators)));
    delete ptr;
}

{
    // error function does not take 1 argument
    auto ptr = new foo_with_over_aligned_memory_operators;
    delete ptr;
}

{
    auto ptr = ::operator new(
        sizeof(foo_with_over_aligned_memory_operators));
    auto typed_ptr =
        ::new (ptr) foo_with_over_aligned_memory_operators;
    assert(is_aligned(typed_ptr, __STDCPP_DEFAULT_NEW_ALIGNMENT__));
    typed_ptr->~foo_with_over_aligned_memory_operators();
    ::operator delete(ptr);
}

```

W pierwszym przypadku dla typu `foo_with_default_aligned_memory_operators` mamy definicje operatorów bez wyrównania, dodatkowo sam typ ma wyrównanie mniejsze od domyślnego, dlatego też zostaną wywołane te operatory, które zostały zdefiniowane dla tego typu.

Typ `foo_aligned_as_32_with_default_aligned_memory_operators` również definiuje operatory `new/delete` bez wyrównania, tym niemniej określa on też wyrównanie większe od domyślnego. Zauważmy jednak, że przeładowanie działa w taki sposób, że jeśli nie uda się znaleźć odpowiednich operatorów z wyrównaniem, to nastąpi próba użycia operatorów z domyślnym wyrównaniem, co też ma miejsce w tym przypadku.

Trzeci przypadek z `foo_aligned_as_32_with_over_aligned_memory_operators` jest analogiczny do wcześniejszego, z tą różnicą, że teraz typ definiuje operatory `new/delete` z wyrównaniem. Dlatego też w ramach przeładowania zostają wywołane te definicje.

Czwarty typ `foo_with_over_aligned_memory_operators` ma 1-bajtowe wyrównanie i jednocześnie definiuje operatory `new/delete` z wyrównaniem. W konsekwencji próba zaalokowania takiego typu za pomocą wyrażenia `new foo_with_over_aligned_memory_operators` spowoduje błąd komilacji. Wykonika to z faktu, że operatory z wyrównaniem są brane pod uwagę tylko w momencie, gdy wyrównanie alokowanego typu jest większe od domyślnego. Warto przypomnieć tutaj także o starej regule przeszukiwania nazw w C++ (ang. *name lookup rule*). Mianowicie dowolna deklaracja funkcji do alokacji wewnątrz klasy powoduje ukrycie wszystkich globalnych funkcji alokacyjnych dla tego typu w ramach wywołania.

W ostatnim przykładzie zostało pokazane, w jaki sposób można ręcznie odwołać się do globalnych operatorów bez wyrównania w celu alokacji typu `foo_with_over_aligned_memory_operators`.

POLIMORFICZNE ALOKATORY

Właściwie od samego początku istnienia biblioteki standardowej C++ alokatory borykały się z problemami. Zaczynając od decyzji o zdefiniowaniu alokatorów bezstanowych, poprzez brak wsparcia dla relokacji, czy alokacji z wyrównaniem pamięci, kończąc na uzależnianiu typu zmiennej od rodzaju użytego alokatora. Czytelników zainteresowanych tym tematem odsyłam do prezentacji Andrei Alexandrescu z CppCon 2015 [2].

W C++11 oraz C++14 naprawiono część z tych problemów, natomiast w C++17 zaadresowano problem z zależnością typu od rodzaju alokatora [3].

Rozpatrzmy następujący kod (Listing 11). Występuje tutaj problem z komplikacją i jest on spowodowany tym, że `vector_a` oraz `vector_b` to różne typy, ponieważ każdy z nich jest statycznie związany z innym rodzajem alokatora.

Listing 11. Przykład problemu z wiązaniem typu alokatora z typem zmiennej

```
template<typename T>
using vector_a = std::vector<T, allocator_a<T>>;
template<typename T>
using vector_b = std::vector<T, allocator_b<T>>;
template<typename T>
void foo(vector_a<T> const& v){}
void bar()
{
    vector_b<int> vec;
    foo(vec); // error
}
```

W C++17 rozwiązano ten problem, wprowadzając polimorficzne alokatory. Polimorficzność została uzyskana poprzez użycie wirtualnego interfejsu z zasobu pamięci, tzw. `std::pmr::memory_resource`. Polimorficzny alokator może przyjąć dowolny zasób pamięci i na jego podstawie będzie dokonywać alokacji. Spójrzmy na poniższy przykład.

Listing 12. Przykład użycia polimorficznych alokatorów

```
constexpr std::size_t tmp_buffer_size = 1 << 10;
std::byte tmp_buffer[tmp_buffer_size] = {};
game_engine game;
while (game.is_running())
{
    std::pmr::monotonic_buffer_resource frame_buffer
    { tmp_buffer, tmp_buffer_size, std::pmr::null_memory_resource{} };
    game.process_input(frame_buffer);
    game.process_tick(frame_buffer);
    game.render(frame_buffer);
}
void game_engine::process_input(
    std::pmr::monotonic_buffer_resource& frame_buffer)
{
    std::pmr::polymorphic_allocator<std::int32_t> poly_allocator
    { &frame_buffer };

    std::pmr::vector<std::int32_t> v{ poly_allocator };
    // ...
}

void game_engine::process_tick(
    std::pmr::monotonic_buffer_resource& frame_buffer)
{
    std::pmr::polymorphic_allocator<foo> poly_allocator
    { &frame_buffer };

    std::pmr::vector<foo> v{ poly_allocator };
    // ...
}
```

Przeanalizujmy główną pętlę gry. Na samym początku utworzony zostaje 1kB blok pamięci. Kolejno, na początku pętli, definiowany jest `std::pmr::monotonic_buffer_resource`. Jest to prosty zasób pamięci, który działa podobnie do alokatora liniowego, mianowicie otrzymuje na starcie zewnętrzną pulę pamięci, na której będzie dokonywać alokacji, natomiast dealokacja całego zaalokowanego obszaru następuje tylko w momencie niszczenia zasobu. W ramach powyższego przykładu chcielibyśmy uzyskać prosty alokator ramkowy do tymczasowych alokacji w obrębie 1 ramki gry⁴. Ponadto chcielibyśmy, żeby nasz alokator ramkowy alokował tylko z prealokowanej puli pamięci, dlatego też jako ostatni parametr konstruktora zasobu pamięci podajemy `std::pmr::null_memory_resource`. Jest to najprostszy zasób pamięci, który nie dokonuje żadnej alokacji, a zamiast tego rzuca wyjątkiem `std::bad_alloc`, jeśli zostanie poproszony o wykonanie alokacji. W dalszej kolejności uprzednio utworzony zasób pamięci zostaje przekazany do kolejnych funkcji silnika gry. W każdej z funkcji utworzony zostaje różny, polimorficzny alokator, a także polimorficzna wersja wektora, która tego alokatora używa.

Standard C++ definiuje następujące zasoby pamięci:

- » `std::pmr::null_memory_resource` – w przypadku próby zaalokowania pamięci rzuca wyjątek `std::bad_alloc`.
- » `std::pmr::new_delete_resource` – domyślny zasób pamięci, który używa operator `new` do alokacji i operator `delete` do dealokacji.
- » `std::pmr::monotonic_buffer_resource` – prosty, liniowy zasób pamięci, który dealokuje tylko w momencie usunięcia zasobu.
- » `std::pmr::synchronized_pool_resource` – generyczny, zapewniający bezpieczny dostęp z wielu wątków zasób, który sam sobie alokuje pulę pamięci. Zasób ten składa się z wielu pul pamięci, które obsługują alokacje o różnych rozmiarach.
- » `std::pmr::unsynchronized_pool_resource` – podobnie jak `synchronized_pool_resource`, przy czym ten zasób nie synchronizuje dostępu z wielu wątków równocześnie.

Warto też wspomnieć o tym, że przez swoją dynamiczność alokatory polimorficzne nie są propagowane do przypisywanego obiektu podczas wykonywania kopiującego/przesuwającego przypisania, czy wołania zamiany (ang. *swap*) dla kontenerów. Oznacza to, że operator przypisania z polimorficznym alokatorem może rzucić wyjątkiem w trakcie wykonywania tych operacji.

Listing 13. Przykład braku propagacji polimorficznego alokatora podczas wołania operatora przesuwającego przypisania

```
std::vector<int> a = {1};
std::vector<int> b;
b = std::move(a); // no copy, no throw

std::pmr::monotonic_buffer_resource mr(64);
std::pmr::vector<int> a({1}, &mr);
std::pmr::vector<int> b;
b = std::move(a); // copy, may throw
```

Problem ten można rozwiązać, przypisując zawczasu alokator.

Listing 14. Rozwiążanie problemu z brakiem propagacji alokatora

```
std::pmr::monotonic_buffer_resource mr(64);
std::pmr::vector<int> a({1}, &mr);
std::pmr::vector<int> b(a.get_allocator());
b = std::move(a); // no copy, no throw
```

4. Na potrzeby przykładu zakładamy, że 1 ramka odpowiada 1 przebiegowi głównej pętli gry.

Polimorficzne alokatory pomimo swoich ewidentnych zalet niosą ze sobą też trochę problemów. Tym, co przede wszystkim rzuca się w oczy, jest niekompatybilność typów używających stacycznych alokatorów z tymi, które używają alokatorów polimorficznych. Powoduje to duży problem w momencie, gdy programista, który zdecydował się na użycie jednego rodzaju alokatorów, musi nagle zintegrować się z biblioteką, która korzysta z drugiego rodzaju alokatorów. W dalszej kolejności można wspomnieć o tym, że polimorficzne alokatory niosą ze sobą narzut czasowy związany z wywoływaniem funkcji wirtualnych, a także narzut pamięciowy spowodowany koniecznością przechowywania wskaźnika na za-sób pamięci w alokatorze.

FUNKCJE POMOCNICZE

W standardzie C++17 dodano także kilka funkcji pomocniczych do zarządzania inicjalizacją oraz destrukcją obiektów w obrębie zarządzanej puli pamięci [4]. Do nagłówka <memory> dodano następujące funkcje:

- » `uninitialized_value_construct` – inicjalizuje obiekty w zadany zakresie za pomocą inicjalizacji wartościowej,
- » `uninitialized_default_construct` – inicjalizuje obiekty w zadany zakresie za pomocą inicjalizacji domyślnej,
- » `uninitialized_copy` – kopiuje obiekty w zadany zakresie do niezainicjalizowanej puli pamięci,
- » `uninitialized_move` – przenosi obiekty w zadany zakresie do niezainicjalizowanej puli pamięci,
- » `uninitialized_fill` – kopiuje zadany obiekt do niezainicjalizowanej puli pamięci w ramach danego zakresu,
- » `destroy` – niszczy obiekty w zadany zakresie,
- » `destroy_at` – niszczy obiekt pod zadanym adresem.

SYSTEM ZARZĄDZANIA PAMIĘCIĄ W GRZE

Po omówieniu najważniejszych zmian w zarządzaniu pamięcią, które dodano w C++ 17, zastanówmy się, czy oferowany przez język zestaw narzędzi odpowiada potrzebom bardziej złożonych systemów. Systemem, który poddamy analizie, będzie manager pamięci w grze komputerowej.

Zaczniemy od zdefiniowania podstawowych wymagań stawianych przed systemem zarządzania pamięcią w grze [5][6][7]:

- » dedykowany zestaw alokatorów definiowanych pod konkretne potrzeby podsystemów gry,
- » ścisła kontrola nad tym, ile i jakiej pamięci jest dedykowanej danemu podsystemowi,
- » gwarancja czasowa dla poszczególnych alokacji (budżet czasowy wywołania),
- » ograniczenie fragmentacji do minimum,
- » wsparcie dla alokacji z dowolnym wyrównaniem,
- » narzędzia debugowe (oznaczanie pamięci, śledzenie pamięci, debugowy alokator, metryki itd.).
- » obsługa `out of memory`,
- » preferowany brak wyjątków (także RTTI na poziomie języka),
- » preferowany statyczny, nieszablonowy interfejs.

Większość z wymagań stawianych przed takim systemem może zostać pokrytych przez to, co oferuje język C++, jak choćby ze-

staw alokatorów może zostać utworzony poprzez konkretne implementacje dziedziczące po `std::pmr::memory_resource`. Tym niemniej jest kilka wymagań, których C++, bez dużego wysiłku ze strony programisty, nie oferuje. Przede wszystkim wsparcie dla alokacji z dowolnym wyrównaniem pamięci działa w C++ automatycznie, tylko wtedy gdy wyrównanie jest większe od domyślnego. Jest to o tyle problematyczne, że prowadzi do sporego narzutu pamięciowego, gdy mamy do czynienia z alokatorem, który jest dedykowany bardzo małym obiektom. Ponadto biblioteka standardowa C++ *de facto* zakłada, że kod, który jej używa, będzie komplikowany ze wsparciem dla wyjątków. Nie inaczej jest w przypadku `std::pmr::memory_resource`, które w sytuacji braku pamięci rzucają `std::bad_alloc`. Zauważmy też, że chcąc używać operatorów `new/delete` z wyrównaniem pamięci, a także mając na uwadze konieczność przypisywania konkretnego budżetu pamięci do danego podsystemu, programista musiałby *de facto* używać tylko `placement new/delete`, ponieważ tylko one pozwalają na przekazanie dodatkowych argumentów z informacją o samej alokacji. Problem, który się tutaj objawia, to fakt, że C++17 nie definiuje operatorów `placement delete` z wyrównaniem. Ten sam problem występuje dla operatora `delete` z rozmiarem alokacji.

Na koniec warto się zastanowić, czy stosowanie mocno szablonowego interfejsu dla alokatorów jest dobrym rozwiązaniem, gdyż może to prowadzić do dużego wzrostu rozmiaru plików obiektowych, a przede wszystkim do znacznego wydłużenia czasu linkowania.

PODSUMOWANIE

Jak widać, oferowane przez C++17 narzędzia do zarządzania pamięcią niekoniecznie sprawdzają się w systemach, które wymagają ścisłej kontroli nad układem pamięci. Choć istnieją systemy [8], które podążając za wytycznymi przez bibliotekę standardową rozwiązaniami, starają się dostosować system zarządzania pamięcią z C++ do swoich potrzeb, to nie jest to powszechną praktyką. Dużo częściej stosowanym rozwiązaniem jest pisanie dedykowanych systemów do zarządzania pamięcią.

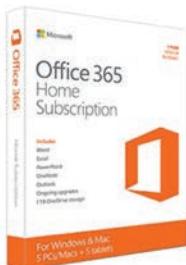
Bibliografia

- [0]: <https://www.youtube.com/watch?v=rX0ItVEVjHc>
- [1]: <https://goo.gl/EMZRSt>
- [2]: <https://www.youtube.com/watch?v=Lb3L4vKZ7U>
- [3]: <https://goo.gl/HMZrsV>
- [4]: <https://goo.gl/4Y3Fud>
- [5]: <http://gameenginebook.com>
- [6]: <https://www.youtube.com/watch?v=f8XdvlO8JxE>
- [7]: <https://goo.gl/HwUzHr>
- [8]: <https://github.com/electronicarts/EASTL>

TOMASZ „SATIREV” JASKÓLSKI

Absolwent Informatyki na Politechnice Poznańskiej. Pracuje jako Engine Programmer w CD Projekt RED. Miłośnik programowania, gier i anime.

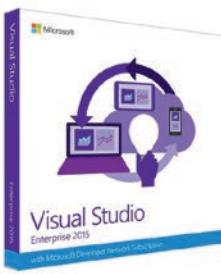
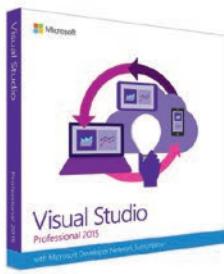
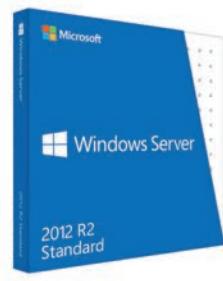
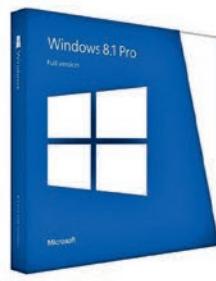
TTS Company rekomenduje oprogramowanie Microsoft ®



Microsoft Partner



Gold Datacenter
Gold Small and Midmarket Cloud Solutions
Silver Data Platform
Silver Data Analytics
Silver Cloud Productivity



www.OprogramowanieKomputerowe.pl

Microsoft Azure

Office 365

OneDrive

Więcej informacji: ☎ (22) 868 40 42 ✉ sales@tts.com.pl

Sprzedaż



Dystrybucja



Import na zamówienie

CafeOBJ

– maszynowy system dowodzenia

CafeOBJ to specjalny język programowania przeznaczony do tworzenia formalnych i zarazem matematycznych specyfikacji modeli opisujących różnego typu systemy z otaczającego nas świata. Zarówno tego prawdziwego, ale także wirtualnego. A w szczególności za pomocą CafeOBJ możemy opisywać tworzone przez nas programy. O CafeOBJ można też powiedzieć, iż jest to system wspomagający przeprowadzanie dowodów matematycznych dla systemów formalnych. Inaczej mówiąc, możemy przeprowadzić weryfikację, czy badany system spełnia postulowane przez nas założenia. Lecz zamiast pracować z kartką i ołówkiem, CafeOBJ przeprowadzi dowód matematyczny za nas.

Nie oznacza to, iż CafeOBJ (CO) to mityczny system przeprowadzający dowolne typy dowodów matematycznych, bowiem aby przeprowadzić dowód, CafeOBJ potrzebuje wprowadzenia dodatkowych informacji. Ogólnie CO korzysta z logiki równościowej oraz systemu przepisywania termów, aby przeprowadzić określony dowód. Wymaga to dostarczenia tzw. modelu. W modelu opisujemy obiekty w postaci encji, danych czy też stanów, a nawet agentów. Należy także opisać operacje realizowane na obiektach, które są reprezentowane tym razem przez funkcje, akcje czy ogólnie zdarzenia.

Jak widać, CafeOBJ to system do zastosowań specjalnych, ale mimo jego matematycznego charakteru jest to język programowania. W artykule na podstawie kilkunastu prostych przykładów pokażemy, jak go zastosować. np. aby przeprowadzić dowód łączności operacji dodawania. System ten może przeprowadzać samodzielnie dowody indukcyjne, w tym także dla programów komputerowych, ale my na początek zaprezentujemy tylko podstawowe przykłady dotyczące liczb oraz typu listowego.

JĘZYK CAFE OBJ

Podstawowym elementem języka CafeOBJ jest moduł – deklarowany słowem `module` lub krócej `mod`. Ogólna składnia modułu przedstawia się następująco:

```
module module name {
    module element*
}
```

W module umieszczamy potrzebne nam definicje, które są podzielone na trzy główne sekcje:

```
imports { module element* }
signature { module element* }
axioms { module element* }
```

Trzeba też dodać, iż są dostępne trzy rodzaje modułów: tzw. ścisłe moduły `mod!`, luźne `mod*` oraz zwykłe `mod`.

Celem pojedynczego modułu jest implementacja tzw. sortu (ang. `sort`), tj. zbioru elementów klasyfikowanych. Definicja sortu, czyli niejako typu, sprowadza się do podania następującego wyra-

żenia: [`NAT`], gdzie `NAT` to nazwa sortu reprezentującego liczby naturalne. CafeOBJ jest dość elastyczny i możemy deklarować kilka sortów, np.: [`Node Edge`]. Język CafeOBJ pozwala wprowadzać porządek na definicjach (nie jest to rodzaj dziedziczenia, bowiem technika dziedziczenia też jest dostępna), wskazując, który element ma poprzedzać drugi:

```
[ <subsort-name> < <supersort-name> ]
```

W przykładzie [`NzInt < Int`] typ `Int` jest niejako szerszy niż `NzInt`, ponieważ sort `NzInt` nie zawiera zera.

Krótko o języku i systemie CafeOBJ

Język CafeOBJ pozwala na formalny zapis specyfikacji modelu opisującego działanie określonego pojęcia nie tylko dla systemów formalnych, ale także rzeczywistego sprzętu, oprogramowania i weryfikację ich własności. W tym celu wykorzystywana jest logika równościowa (ang. *equational logic*) oraz system przepisywania (ang. *term rewriting system*) do wyprawdzania dowodów. Przeprowadzanie dowodu jest możliwe za pomocą tzw. rezultatów dowodowych (ang. *proof scores*), których realizacja przez system CafeOBJ pozwala na dowodzenie określonych własności.

Sam system CafeOBJ został opracowany w języku Lisp i wymaga interpretera, kompilatora tego języka. Mimo tego jest bardzo wydajny i CafeOBJ pozwala na przeprowadzanie dowodów nawet dla dość rozbudowanych systemów w bardzo krótkim (tj. w trakcie kilku sekund) czasie.

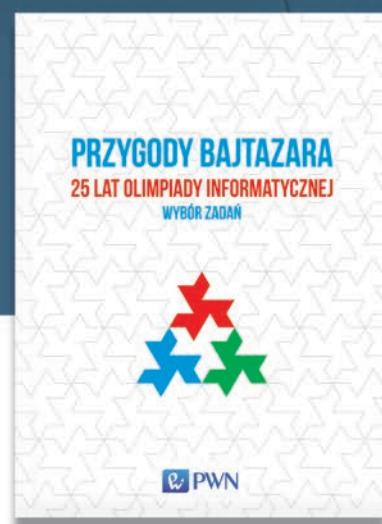
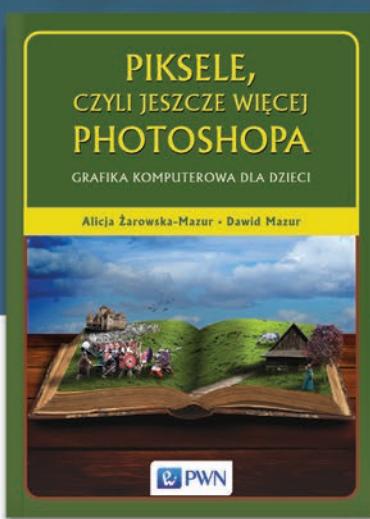
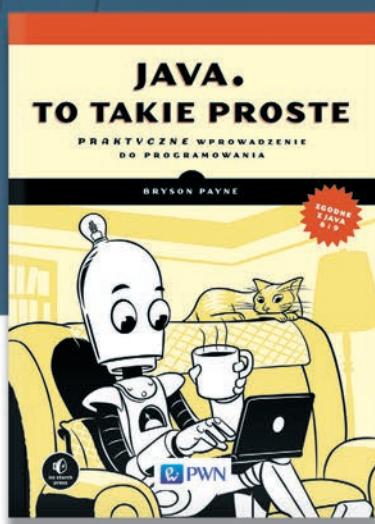
CafeOBJ ma także przydatną dokumentację oraz przykłady, które także zostały wykorzystane w tym artykule. Zatem czytelnikom zainteresowanym dalszym poznaniem środowiska CafeOBJ polecamy przejrzenie dokumentacji, a także przykładów dołączonych do środowiska CO.

Zanim podam przykładowe definicje modułów, warto jeszcze dodać, iż sorty mogą dziedziczyć z innych sortów. Wzorem tradycyjnego dziedziczenia mamy trzy rodzaje dziedziczenia: pierwszy typ to tzw. dziedziczenie chronione (słowo `protecting`, w skrócie `pr`). W tym typie dziedziczenia nie można dodawać ani usuwać istniejących definicji, pojęć etc. W drugim typie dziedziczenia opisany przez słowo `extending` (ex) nowe elementy mogą być dodawane, lecz redefinicja istniejących nie jest dopuszczalna. Najbardziej liberalny jest trzeci sposób dziedziczenia realizowany przez słowo

Najlepsze od



KOMPUTERY SĄ GOTOWE
DO UWOLNIENIA NASZYCH
UMYSŁÓW



ODWIEDŹ NAS NA



IT.PWN.PL i zapisz się na newsletter!



KSIEGARNIA.PWN.PL

using (us), które pozwala na pełną modyfikację pojęć, operacji, aksjomatów.

```

C:\WINDOWS\system32\cmd.exe
-- defining module! PNAT
[Warning]: Redefining module PNAT
[Error]: declaring operator, no such sort Bool
-- failed to evaluate the form:
Operator declaration:
  symbol = (_ + _)
  arity = Nat Nat
  coarity = Bool
attributes :
- theory (comm)
CafeOBJ> modl PNAT+
pr(PNAT)
op _+ : Nat Nat -> Nat .
vars N1 N2 : Nat .
-- equations
eq 0 + N2 = N2 .
eq (s N1) + N2 = s(N1 + N2) .
}

-- defining module! PNAT+ done.
CafeOBJ> select PNAT+
PNAT+ red s s 0 + s 0 .
-- reduce in PNAT+ : ((s (s 0)) + (s 0)):Nat
(s (s (s 0))):Nat
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 5 matches)
PNAT+> red s s 0 + s s 0 .
-- reduce in PNAT+ : ((s (s 0)) + (s (s 0))):Nat
(s (s (s (s 0)))):Nat
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 5 matches)
PNAT+>

```

Rysunek 1. Środowisko CafeOBJ w konsoli podczas pracy z jednym z przykładów

OPERACJE I RÓWNANIA

Na modułach – a raczej powinniśmy powiedzieć, że na sortach – możemy wykonywać operacje realizujące przetwarzanie danych opisanych w module. Składnia operacji jest następująca:

`op <op-name> : <arity> -> <sort-of-op> { <operator-attribute> }`

Dostępna jest odmiana wprowadzająca możliwość definicji kilku operacji:

`ops <op-name> ... <op-name> : <arity> -> <sort-of-op> { <operator-attribute> }`

Podajemy nazwę operacji, argumenty, tj. opis, na jakich sortach działamy. Po strzałce podajemy typ rezultatu, a także dodatkowe atrybuty.

Samo pojęcie operacji jest tu dość elastyczne, bowiem możemy tworzyć statyczne obiekty, np.:

`ops true false : -> Bool`

W ten sposób utworzono dwie operacje reprezentujące wartości logiczne. Pominieły zostały argumenty, a podano tylko typ wyniku. Typowym przykładem operacji jest np. operator plus:

`op _+_ : Nat Nat -> Nat`

Znaki podkreślenia reprezentują odpowiednio lewy i prawy argument. Po dwukropku opisaliśmy argumenty operatora plus jako liczby naturalne, a także rezultat, który również jest liczbą naturalną.

CafeOBJ oferuje też dostęp do konstrukcji if then else, zatem jeśli potrzebna jest nam konstrukcja warunkowa, to operację tego typu zapiszemy w następujący sposób:

`op if_then_else_if : Bool Exp Exp -> Exp`

Znaki podkreślenia pozwalają na określenie, na jaki typ oczekujemy w danym miejscu konstrukcji warunkowej.

Istotne są też atrybuty np. dla przykładu:

`op _+_ : Nat Nat -> Nat { assoc comm }`

definiujemy operator dodawania, który jest łączny (operator `assoc`) oraz przemienny (operator `comm`), operatorów dodatkowych jest więcej, poniżej je tylko wymieniamy:

`assoc, comm, idem, idr: <op-name> : identity, r-assoc, l-assoc, prec: <precedence>, strat: (<e-strategy>)`

Po opisaniu, jakie operacje są związane z naszym modelem, możemy przystąpić do ich implementacji. W tym celu stosowane są równania o następującej syntaktyce:

`eq <lhs> = <rhs> .`

Powyższą konstrukcję należy rozumieć, iż lewa strona wyrażenia powinna być równa prawej stronie wyrażenia. Toteż jeśli CafeOBJ znajdzie lewe wyrażenie, to może je zastąpić prawym wyrażeniem. Łatwo to pokazać na przykładzie:

`eq 0 + M:Nat = M .`

Równanie opisuje fakt, iż zero nie zmienia wartości wyrażenia M. Dlatego prawa strona równania to ponownie ta sama wartość M. Należy także zwrócić uwagę na końcową kropkę, nie może ona zostać dołączona do ostatniego wyrażenia i zawsze należy ją dodać po spacji. Język CafeOBJ w tym przypadku jest bardzo rygorystyczny, zatem trzeba pamiętać, aby kropkę na końcu stawiać po spacji.

Równania mogą być również warunkowe, składnia wymaga dodania sekcji `if`, gdzie opiszemy warunki, przy jakich należy zastosować równanie:

`ceq <lhs> = <rhs> if <condition> .`

Przeznaczenie tego typu konstrukcji to optymalizacja późniejszych dowodów, konstrukcja pozwala bezpośrednio wskazać, w jakim przypadku należy zastosować określone równanie.

W module mogą także występować klasyczne zmienne, szczególnie mogą one być pomocne w trakcie realizacji równań. Składnia deklaracji zmiennych również przyjmuje dwie postaci, dla pojedynczej zmiennej:

`var <var-name> : <sort-name>`

a także dla wielu zmiennych:

`vars <var-name> ... <var-name> : <sort-name>`

W Listingu 1 przedstawiono definicję modułu implementującego sort reprezentujący liczby naturalne. Jest to wersja posługująca się pełnym zapisem syntaktycznym z podziałem na poszczególne sekcje. W pierwszej sekcji opisujemy porządek na sortach, następnie wymieniamy dopuszczalne operacje oraz opisujemy operację sumy dla dwóch wielkości SIMPLE-NAT. Następnie w sekcji aksjomatów podajemy równanie odpowiedzialne za realizację sumy, gdzie, jak widać, używamy dwóch zmiennych zdefiniowanych we wcześniejszej sekcji.

Listing 1. Definicja przykładowego modułu implementującego liczby naturalne

```
module SIMPLE-NAT {
  signature {
    [ Zero NzNat < Nat ]
  }
  signature {
    op 0 : -> Zero
    op s : Nat -> NzNat
  }
  signature {
    op _+_ : Nat Nat -> Nat
  }
  axioms {
    vars N N' : Nat
  }
  axioms {
    eq 0 + N = N .
    eq N + s(N') = s(N + N') .
  }
}
```

Nie musimy jednak tak dokładnie opisywać wszystkich szczegółów z punktu widzenia syntaktycznego. W Listingu 2 przedstawiono ten sam moduł zapisany na dwa sposoby. W pierwszym mamy jasny podział na sekcje `signature` oraz `axioms`, ale nie jest to wymagane i możemy moduł SIMPLE-NAT zapisać bez ich jawniej specyfikacji. Pokazano to w drugiej części Listingu 2. Nie mamy tam już jawnego podziału na sekcje i jeśli będziemy zaglądać do dokumentacji, czy też przykładów, to najczęściej będziemy spotykać taki jak w Listingu 2 uproszczony zapis.

Listing 2. Dwie uproszczone postaci modułu SIMPLE-NAT

```
module SIMPLE-NAT {
  signature {
    [ Zero NzNat < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _+_ : Nat Nat -> Nat
  }
  axioms {
    vars N N' : Nat
    eq 0 + N = N .
    eq s(N) + N' = s(N + N') .
  }
}
```

Wersja druga:

```
module SIMPLE-NAT {
  [ Zero NzNat < Nat ]
  op 0 : -> Zero
  op s : Nat -> NzNat
  op _+_ : Nat Nat -> Nat
  vars N N' : Nat
  eq 0 + N = N .
  eq N + s(N') = s(N + N') .
}
```

SYSTEM PEANO NA ROZGRZEWKĘ

Poznanie języka CafeOBJ wymaga jak widać nieco innego podejścia do przykładów niż w przypadku typowych języków programowania jak Java czy np. Scala. Zamiast kolejnego przykładu „Hello World!”, utworzymy sort dla liczb naturalnych w oparciu o system Peano. Wykorzystamy wbudowany w CafeOBJ dostępny moduł NAT. Ogólnie system Peano jest zbudowany przy użyciu trzech reguł:

- » (I) zero jest liczbą naturalną,
- » (II) jeśli n jest liczbą naturalną, to $(s\ n)$ także jest liczbą naturalną,
- » (III) obiekty tworzone za pomocą reguł (I) oraz (II) są liczbami

Instalacja środowiska CafeOBJ

Środowisko CafeOBJ z racji tego, iż jest napisane w Lisp, wykorzystuje środowisko tego języka. Autorzy CO polecają kilka odmian języka Lisp, w tym także wersje komercyjne. Najlepiej jednak wybrać ogólnie dostępne implementacje języka Lisp, takie jak Steel Bank Common Lisp lub po prostu Common Lisp. Znakomita większość dystrybucji systemu Linux oferuje odpowiednie pakiety, w tym także polecane dla CafeOBJ. Zatem najwygodniej będzie zainstalować język Lisp z pakietów danej dystrybucji. Sam pakiet CafeOBJ także jest już dojrzałym narzędziem, zatem i jego również znajdziemy w pakietach dystrybucji np. Debian czy Ubuntu. Instalację możemy przeprowadzić, wydając polecenie:

```
apt-get install cafeobj
```

CafeOBJ jest także dostępny dla innych systemów niż Linux, np. dla macOS czy Windows. W przypadku popularnych okienek CO oferują pakiet binarny dla systemu Windows (tu warto zwrócić uwagę, iż stosowana jest komercyjna wersja języka Lisp, zatem jeśli planujemy komercyjnie korzystać z CafeOBJ, lepiej będzie samodzielnie skompilować CO na swoje potrzeby). Jest to kompletna instalacja razem z odpowiednim środowiskiem języka Lisp, zatem nie trzeba instalować i kompilować pakietów samodzielnie.

Jeśli zależy nam na instalacji z kodu źródłowego, to najwygodniej ją zrealizować w systemie Linux albo macOS. Należy w pierwszej kolejności posiadać odpowiednią instalację języka Lisp. Kompilację z kodu źródłowego rozpoczęmy od pobrania źródeł z repozytorium:

```
git clone https://github.com/CafeOBJ/cafeobj.git
```

Następnie przechodzimy do katalogu ze źródłami:

```
cd cafeobj
```

I wykorzystując typowy skrypt `configure`, przeprowadzamy konfigurację pakietu:

```
./configure --with-lisp=LISP-TYPE
```

gdzie w miejsce LISP-TYPE możemy podać `clisp`, jeśli chcemy zastosować Common Lispa, albo `sbc1`, gdy mamy w systemie zainstalowany Steel Bank Common Lisp. Następnie typowymi poleceniami:

```
make  
sudo make install
```

przeprowadzamy komplikację oraz instalację systemu CafeOBJ.

naturalnymi.

Wyrażenie $(s\ n)$ to tzw. następnik, czyli z zera tworzymy jedność, z jedności budujemy dwójkę, czyli mamy $(s\ (s\ 0))$. Możemy to pokazać za pomocą poniższych przykładów:

```
0, 0+1, 0+1+1, 0+1+1+1, 0+1+1+1+1, i etc.,  
0, s(0), s(s(0)), s(s(s(0))), s(s(s(s(0)))), i etc.
```

Analizując przykłady budowy składników w systemie Peano, można wyróżnić dwa elementy: obiekt reprezentowany przez wielkość naturalną oraz operacje. Przy czym są to dwie operacje, jedna zwracająca zero oraz s , która dla danego obiektu n zwraca następny obiekt następujący po n : $(s\ n) = n + 1$.

W Listingu 3 przedstawiono definicję modułu PNAT. Znajdują się w nim potrzebne definicje w postaci zera, następnika, tj. operacji s , oraz operatora równości. Implementację poszczególnych operacji realizujemy za pomocą trzech równań. Pierwsze równanie:

```
eq (N:Nat = N) = true .
```

oznacza równość elementu z samym sobą, wbrew pozorom musimy to jawnie określić. Podobnie w przypadku zera, które nie będzie równe w przypadku porównania go z innymi elementami, co zapisujemy w postaci poniższego równania:

```
eq (0 = s(N2:Nat)) = false .
```

Ważną rolę pełni trzecie równanie, które wskazuje, że wartości następników będą równe wtedy, gdy równe są argumenty następników:

```
eq (s(N1:Nat) = s(N2:Nat)) = (N1 = N2) .
```

Rolę uzupełniającą pełni moduł PNAT+, ale choć napisaliśmy, iż rola jest pomocnicza, to moduł ten jest niezbędny, aby przeprowadzić przykładowe dowody opisane w dalszej części artykułu. W module PNAT+ wprowadzamy operację sumy i dwa równania, które ją implementują. Pierwsze z nich dotyczy zera, gdzie zapisujemy operację, że dodanie do wartości np. N2 zera nie zmienia wartości N2:

```
eq 0 + N2 = N2 .
```

Drugie równie odpowiada za operację sumy. Opisujemy przypadek, że do wartości następnika dodajemy nową wartość N2, co jest równe sumie argumentów następnika:

```
eq (s N1) + N2 = s(N1 + N2) .
```

Listing 3. Definicje potrzebnych modułów PNAT oraz PNAT+

```
mod! PNAT {
    [ Nat ]
    op 0 : -> Nat {constr} .
    op s_ : Nat -> Nat {constr} .
    op _=_ : Nat Nat -> Bool {comm} .
    eq (N:Nat = N) = true .
    eq (0 = s(N2:Nat)) = false .
    eq (s(N1:Nat) = s(N2:Nat)) = (N1 = N2) .
}
```

Drugi moduł:

```
mod! PNAT+ {
    pr(PNAT)
    op _+_ : Nat Nat -> Nat .
    vars N1 N2 : Nat .
    eq 0 + N2 = N2 .
    eq (s N1) + N2 = s(N1 + N2) .
}
```

Zanim zrealizujemy przykładowe dowody, warto sprawdzić, czy nasze moduły poprawnie działają. Po włączeniu środowiska CafeOBJ możemy zwyczajnie przekopiować treść modułów np. PNAT+ i wkleić do konsoli treść modułu. Po naciśnięciu klawisza enter, jeśli nie było błędów syntaktycznych, moduły zostaną wprowadzone do systemu.

Sprawdzenie, czy nasze moduły funkcjonują poprawnie, rozpoznamy, wydając następujące polecenie:

```
select PNAT+
```

Uaktywnimy w ten sposób moduł o podanej nazwie. Możemy teraz wydać polecenie redukcji (wyrażenie PNAT+ to znak zachęty):

```
PNAT+> red s s 0 + s 0 .
```

Pamiętajmy, że kropkę na końcu podajemy po spacji. Polecenie redukcji (reduce, lub krótko red) pozwala dokonać ewaluacji podanego wyrażenia; mówiąc inaczej, obliczymy wartość podanego wyrażenia. Stosujemy też skrócony zapis syntaktyczny bez nawiasów. Wydanie polecenia red skutkuje tym, iż zobaczymy następującą odpowiedź:

```
-- reduce in PNAT+ : ((s (s 0)) + (s 0)):Nat
(s (s (s 0))):Nat
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 5 matches)
```

W odpowiedzi dostaliśmy trójkę zapisaną w postaci wyrażenia z funkcją s. Można wykorzystując polecenie trace, poprosić, aby system pokazał, jakie reguły zostały zastosowane, aby prowadzić operację redukcji. Odpowiedź zobaczymy w Listingu 4.

Listing 4. Wykonanie operacji redukcji wraz z listą zastosowanych reguł w przeprowadzanej ewaluacji wyrażenia s s 0 + s 0

```
PNAT+> set trace on
PNAT+> red s s 0 + s 0 .
-- reduce in PNAT+ : ((s (s 0)) + (s 0)):Nat
1>[1] rule: eq ((s (s 0)) + (s 0)) = (s (N1 + N2))
{ N1 |-> (s 0), N2 |-> (s 0) }
1<[1] ((s (s 0)) + (s 0)):Nat --> (s ((s 0) + (s 0))):Nat
1>[2] rule: eq ((s N1) + N2) = (s (N1 + N2))
{ N1 |-> 0, N2 |-> (s 0) }
1<[2] ((s 0) + (s 0)):Nat --> (s (0 + (s 0))):Nat
1>[3] rule: eq (0 + N2) = N2
{ N2 |-> (s 0) }
1<[3] (0 + (s 0)):Nat --> (s 0):Nat
(s (s (s 0))):Nat
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 5 matches)
```

PNAT+>

ŁĄCZNOŚĆ I PRZEMIENNOŚĆ DODAWANIA

Po sprawdzeniu poprawności działania modułu PNAT+ możemy przeprowadzić dowód łączności dodawania za pomocą indukcji strukturalnej. Własność tą możemy zapisać jako:

$$\forall_{i,j,k \in \mathbb{N}} (i + j) + k = i + (j + k)$$

W CafeOBJ dowód możemy wprowadzić, podając następujące wyrażenia: w pierwszej kolejności otwieramy potrzebne moduły (korzystamy z dodatkowego modułu EQL):

```
open NAT+ + EQL
```

Deklarujemy operacje reprezentujące zmienne:

```
ops i j k : -> Nat .
```

Opisujemy krok początkowy indukcji:

```
red ((0 + j) + k) = (0 + (j + k)) .
```

A następnie krok indukcyjny, wraz z jego redukcją:

```
eq (i + j) + k = i + (j + k) .
```

```
red ((s(i) + j) + k) = (s(i) + (j + k)) .
```

Ostatnie polecenie, jakie należy wpisać, to `close`, aby zamknąć ciąg polecień i wykonać dowód. Na Rysunku 2 pokazano, jak ten proces przebiega w konsoli CafeOBJ.

```
C:\dev\cafeobj-1.5-sbcl\cafeobj.exe
-- done reading in file: eq1
done.
processing input : C:/dev/cafeobj-1.5-sbcl/lib/sys_bool.cafe
-- defining module! BOOL done. done.

CafeOBJ>
mod! NAT+ {
  pr(SIMPLE-NAT)
  op _+_ : Nat Nat -> Nat
  eq 0 + M:Nat = M .
  eq s (N:Nat) + M:Nat = s (N + M) .
}

-- defining module! NAT+ done.

CafeOBJ>
open NAT+
.

-- opening module NAT+ done.

%NAT+> ops i j k : -> Nat .
%NAT+> red ((0 + j) + k) = (0 + (j + k)) .
-- reduce in %NAT+ : (((0 + j) + k) = (0 + (j + k))):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 11 matches)

%NAT+> eq (i + j) + k = i + (j + k) .

%NAT+> red ((s(i) + j) + k) = (s(i) + (j + k)) .
-- reduce in %NAT+ : (((s(i) + j) + k) = (s(i) + (j + k))):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 5 rewrites + 32 matches)

%NAT+> close
CafeOBJ>
```

Rysunek 2. Środowisko CafeOBJ w konsoli podczas pracy z dowodem łączności dodawania

Drugi dowód, jaki chcemy podać, dotyczy przemienności dodawania:

$$\forall_{i,j \in N} i + j = j + i$$

Dowód przebiega podobnie jak poprzedni, zatem na początku sprawdzamy krok bazowy. Jednakże, co oczywiście, pierwszą czynność, jaką wykonamy, to uzyskanie dostępu do modułu:

```
open PNAT+ .
```

Następnie tworzymy odpowiednie definicje:

```
eq X:Nat + 0 = X .
op y : -> Nat .
```

I możemy przeprowadzić redukcję:

```
red 0 + y = y + 0 .
```

Dalej zostaje nam już tylko zamknąć moduł `close`. W ten sposób przeprowadzony został dowód dla kroku bazowego indukcji. Możemy teraz przejść do hipotezy indukcyjnej i jej dowodu. Moduł PNAT+ otwieramy i zamykamy tak jak poprzednio, więc operacją istotną dla dowodu jest definicja wartości n reprezentowana przez operację:

```
op n : -> Nat .
```

A jej implementacja w postaci równania jest następująca:

```
eq n + Y:Nat = Y + n .
```

I podajemy drugie równanie:

```
eq X:Nat + s Y:Nat = s (X + Y) .
```

Po czym możemy wprowadzić kolejną wartość y:

```
op y : -> Nat .
```

Pozostało już tylko wykonać operację redukcji, aby zakończyć dowód:

```
red s n + y = y + s n .
```

```
C:\dev\cafeobj-1.5-sbcl\cafeobj.exe
vars N1 N2 : Nat .
-- equations
eq 0 + N2 = N2 .
eq (s N1) + N2 = s(N1 + N2) .
}

-- defining module! PNAT+ done.

CafeOBJ> open PNAT+ .
-- opening module PNAT+ done.

%PNAT+> eq X:Nat + 0 = X .
%PNAT+> op y : -> Nat .
%PNAT+> red 0 + y = y + 0 .
-- reduce in %PNAT+ : (((0 + y) = (y + 0))):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 4 matches)

%PNAT+> close
CafeOBJ> open PNAT+ .
-- opening module PNAT+ done.

%PNAT+> op n : -> Nat .
%PNAT+> eq n + Y:Nat = Y + n .
%PNAT+> eq X:Nat + s Y:Nat = s (X + Y) .
%PNAT+> op y : -> Nat .
%PNAT+> red s n + y = y + s n .
-- reduce in %PNAT+ : (((s n) + y) = (y + (s n))):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 4 rewrites + 16 matches)

%PNAT+> close
CafeOBJ>
```

Rysunek 3. Dowód przemienności dodawania w konsoli środowiska CafeOBJ

ŁĄCZNOŚĆ ŁĄCZENIA LIST

Ostatni przykład, jaki chcemy zaprezentować, będzie już bardziej przypominał zagadnienie dla programistów, bowiem chcemy pokazać, iż operacja łączenia dwóch list także jest łączna.

W Listingu 5 zawarto definicję modułów LIST-NAT oraz LISTA, które opisują klasyczny typ listowy, jaki spotkamy w językach Lisp czy Scheme. Mamy wartość `nil`, tj. pustą listę, operację łączenia list za pomocą znaku plus. Natomiast za pomocą kropki dodajemy element typu NAT do listy. Dostępne są też dwie podstawowe operacje na listach, to znaczy tzw. car listy, czyli pierwszy element, jaki znajduje się na liście, oraz cdr listy, czyli ogon listy, a inaczej mówiąc, pozostałe elementy poza pierwszym.

Przeprowadzenie dowodu składać się będzie z czterech części. W pierwszej otwieramy moduł LISTA oraz definiujemy listy oraz elementy reprezentowane przez operacje:

```
open LISTA .
ops l l' l'' : -> List .
ops m n p : -> Nat .
```

Na początku kroku indukcyjnego sprawdzamy, jak wartość `nil` wpływa na zawartość listy:

```
red nil + (l + l') == (nil + l) + l' .
```

Następnie tworzymy hipotezę:

```
eq l + (l' + l'') = (l + l') + l'' .
```

```

C:\dev\cafeobj-1.5-sbcl\CafeOBJ.exe

-- done reading in file: truth
done.
-- defining module* BASE-BOOL
-- reading in file : eql
processing input : C:/dev/cafeobj-1.5-sbcl/lib/eql.cafe
-- defining module! EQL done.
-- done reading in file: eql
done.
processing input : C:/dev/cafeobj-1.5-sbcl/lib/sys_bool.cafe
-- defining module! BOOL done.
-- done reading in file: bool
done.
-- done reading in file: nznat
done. done.

CafeOBJ>
CafeOBJ> open LISTA .

[Error]: incorrect module expression or unknown module LISTA
CafeOBJ> mod! LISTA { protecting(LIST-NAT) op _+_ : List List -> List eq nil + L:List = L . eq (N:Nat , L:List) +
L':List = (N . (L + L')) . }

-- defining module! LISTA done.

CafeOBJ> open LISTA .
-- opening module LISTA done.

%LISTA> ops l l' l'' : -> List . ops m n p : -> Nat .

%LISTA>
%LISTA> red nil + (l + l') == (nil + l) + l'
-- reduce in %LISTA : ((nil + (l + l')) == ((nil + l) + l')):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 3 rewrites + 11 matches)

%LISTA> eq l + (l' + l'') = (l + l') + l'' .
%LISTA> red (n . l) + (l' + l'') == ((n . l) + l') + l'' .
-- reduce in %LISTA : (((n . l) + (l' + l'')) == (((n . l) + l') + l'')):Bool
(true):Bool
(0.0000 sec for parse, 0.0000 sec for 5 rewrites + 32 matches)

%LISTA> -

```

Rysunek 4. Własność łączności operacji łączenia list jako dowód w środowisku CafeOBJ

Redukcja następującego wyrażenia:

```
red (n . 1) + (l' + l'') == ((n . 1) + l') + l'' .
```

powinna zwrócić wartość logiczną True, co oznacza, iż łączenie dwóch list jest również łączne, tak jak operacja sumy na liczbach całkowitych. Przedstawiono to także na Rysunku 4.

Listing 5. Definicja modułu LIST-NAT oraz LISTA

```

mod! LIST-NAT {
protecting(NAT)

[ NList < List ]
op nil : -> List
op _-_ : Nat List -> NList
op car_ : NList -> Nat
op cdr_ : NList -> List

var L : List
var N : Nat

eq car (N . L) = N .
eq cdr (N . L) = L .
}
```

Moduł LISTA:

```

mod! LISTA {
protecting(LIST-NAT)

op _+_ : List List -> List
eq nil + L:List = L .
eq (N:Nat . L:List) + L':List = (N . (L + L')) .
}
```

PODSUMOWANIE

CafeOBJ to język programowania – i jak mamy nadzieję, że udało się to pokazać – z przeznaczeniem do innych zadań niż typowe języki, jakie dobrze znamy. Nasze przykłady dotyczyły podstawowych własności systemów liczbowych oraz list. Naszym głównym zadaniem było pokazanie, w jaki sposób można przeprowadzać podstawowe przykłady dowodów.

CafeOBJ to znacznie silniejsze narzędzie, niż sugerują to nasze przykłady. Warto zajrzeć do dokumentacji dołączonej do dystrybucji CafeOBJ. Znajdziemy tam opis działania bankomatu. Choć może to wydawać się dziwne, a nawet nieco śmieszne, to CO umożliwia modelowanie sposobu funkcjonowania bankomatu i możemy formalnie weryfikować przyjęte założenia co do algorytmu sterującego bankomatem. Pozwoli to sprawdzić, czy potencjalny program obsługujący urządzenie tego typu nie będzie się blokował np. w przypadku źle wybieranych opcji lub z powodu błędnych danych odczytanych z karty płatniczej.

W sieci:

- » Oficjalna strona projektu CafeOBJ: <https://cafeobj.org/>
- » Implementacja Common Lispa o nazwie Steel Bank stosowana przez CafeOBJ: <http://www.sbcl.org/>

MAREK SAWERWAIN



Autor, pracownik naukowy Uniwersytetu Zielonogórskiego, na co dzień zajmuje się teorią kwantowych języków programowania, ale także tworzeniem oprogramowania dla systemów Windows oraz Linux. Zainteresowania: teoria języków programowania oraz dobra literatura.

NAJWIĘKSZEJ KONFERENCJI

DEDYKOWANEJ PLATFORMIE .NET!

**Nowa formuła:
aż 4 równoległe sesje
tematyczne!**

Pre-cons

17 września 2018

**Konferencja
18-19 września**

Neal Ford:
Building Evolutionary
Architectures



Daniel Marbach:
Workshop Async/Await and the
Task Parallel Library



Sasha Goldshtein:
Making .NET Applications
Faster



Shawn Wildermuth:
Writing and Securing APIs
with ASP.NET Core 2



**Warszawa
EXPO XXI**
net.developerdays.pl

NoSQL na przykładzie Apache Cassandra® i Scylla

Apache Cassandra® to uznana baza danych NoSQL, która jest bardzo popularna wśród firm przechowujących i przetwarzających duże ilości danych (Apple, Netflix), głównie z powodu jej liniowej skalowalności horyzontalnej. Projekt powstał 10 lat temu w firmie Facebook i po pewnym czasie został opublikowany na licencji open-source i przekazany pod kontrolę Apache Foundation®. Decyzje projektowe zaczerpnięte z Amazon Dynamo (model rozproszenia) i Google Big Table (sposób zapisu danych na dysku) są na tyle dobre, że pozwalają skalować klastry Cassandra nawet do kilku tysięcy serwerów.

Mimo wielu zalet Apache Cassandra® nie jest wolna od pewnych problemów, związanych z faktem, że została zaimplementowana w technologii Java. Działanie na wirtualnej maszynie, która zarządza pamięcią przy użyciu *Garbage Collectora*, uniemożliwia, a przynajmniej znaczco utrudnia zapewnienie odpowiednio krótkich i stabilnych czasów obsługi żądań. Dodatkowo język Java powstał w celu zwiększenia produktywności programistów i zdecydowanie nie traktuje wydajności jako obywatała pierwszej kategorii. W większości zastosowań jest to sensowny wybór, ale w przypadku baz danych, które przetwarzają olbrzymie ilości informacji, spadek wydajności jest niestety mocno odczuwalny.

Kolejny problem Cassandra® jest związany z faktem, że powstała ona w 2008 roku. W tamtym czasie przeciętny serwer posiadał tylko 4 rdzenie procesora. Przez ostatnie 10 lat sytuacja zmieniła się diametralnie. Przyspieszanie taktowania pojedynczego rdzenia przestało mieć ekonomiczny sens i producenci sprzętu zbudowali nową strategię rozwoju na pakowaniu jak największej liczby rdzeni na jednej płytce. Klasyczny model współprzeźności oparty na wątkach nie jest w stanie wykorzystać nowej architektury sprzętowej w zadowalającym stopniu.

W odpowiedzi na wydajnościowe problemy Cassandra® powstała Scylla, która jest zaimplementowana w języku C++ i zaprojektowana zgodnie z ideą share-nothing-architecture, która polega na uruchamianiu jedynie tylu wątków, ile dostępnych jest wątków sprzętowych (zazwyczaj dwa razy więcej niż liczba rdzeni procesora), oraz na dzieleniu zasobów (np. pamięci operacyjnej), tak by każdy wątek miał własną część niedostępna dla pozostałych wątków. Komunikacja między wątkami odbywa się za pomocą specjalnego mechanizmu, a wszystkie pozostałe operacje wykorzystują jedynie zasoby należące do wykonującego je wątku. Dzięki temu nie ma potrzeby wykonywać kosztownej synchronizacji. W efekcie Scylla jest nawet 10x wydajniejsza od

Cassandra®, zachowując jednocześnie kompatybilność na poziomie formatu danych, protokołów komunikacyjnych oraz narzędzi administracyjnych.

Budowa i działanie rozproszonych baz danych to skomplikowana materia, dlatego w tym artykule przedstawiony jest jedynie wysokopoziomowy opis Cassandra® i Scylli. Pewne mechanizmy, które nie są niezbędne, ale usprawniają działanie bazy, zostały pominięte. Zainteresowanych większą liczbą szczegółów zapraszam do dokumentacji produktowej.

MODEL DANYCH

Po tym nieco przydłużym wstępnie przyjrzyjmy się sposobowi, w jaki modelowane są dane przechowywane w Cassandrze® i Scylli. W tym celu używany jest język zapytań CQL (Cassandra Query Language), który jest bardzo podobny do języka SQL. Dane organizowane są w tabelach o wcześniej zdefiniowanym schemacie. Opisywane bazy nie są jednak bazami relacyjnymi. W związku z tym CQL jest mocno ograniczony w porównaniu do SQLa. Na przykład nie ma możliwości łączenia tabel za pomocą operacji JOIN, a klauzula WHERE jest ograniczona tylko do kolumn, które należą do klucza identyfikującego wiersz. W Listingu 1 przedstawiono przykładową definicję schematu tabeli o nazwie *odczyty_temperatury*, która posiada 5 pól: *data_odczytu*, *godzina_odczytu*, *identyfikator_sensora*, *czas_odczytu* i *wartosc_odczytu*.

Listing 1. Przykładowa definicja schematu tabeli w CQL

```
CREATE TABLE odczyty_temperatury (
    data_odczytu DATE,
    godzina_odczytu INT,
    identyfikator_sensora INT,
    czas_odczytu TIME,
    wartosc_odczytu DOUBLE,
    PRIMARY KEY((data_odczytu, godzina_odczytu), identyfikator_sensora, czas_odczytu)
);
```

Jest to tabela przechowująca dane odbierane od sensorów meteorologicznych w regularnych odstępach czasu. Ważnym elementem definicji jest sekcja PRIMARY_KEY, która definiuje, z jakich pól składa się klucz każdego z wierszy. W Cassandrze® i Scylli taki klucz składa się z dwóch komponentów. Pierwsza część, zamknięta w dodatkowe nawiasy, definiuje klucz partycji (partition key), który jest używany do rozproszenia danych między wiele serwerów.Więcej o tym w dalszej części artykułu. W naszym przykładzie na klucz partycji składają się pola `data_odczytu` i `godzina_odczytu`. Druga część klucza, złożona z pozostałych pól wymienionych w definicji klucza, określa klucz lokalny (clustering key), który służy do organizacji danych wewnątrz partycji. W naszym przykładzie klucz lokalny to pola `identyfikator_sensora` i `czas_odczytu`. Pole `wartosc_odczytu` nie należy do żadnego z kluczy i jest po prostu wartością, która może być obecna w wierszu. Może, ale nie musi. Ewentualny brak wartości tego pola jest przez bazy akceptowany. Wszystkie pola należące do kluczy są obowiązkowe i muszą być obecne.

MODEL ROZPROSZENIA

Zobaczmy, jak Cassandra® i Scylla wyglądają z punktu widzenia systemów rozproszonych. Sposób, w jaki te bazy korzystają z wielu serwerów, jest ewolucją pomysłów pierwszy raz opublikowanych przy okazji prac nad bazą Amazon Dynamo. Fundamentem jest tak zwana „masterless architecture”, w której każdy serwer jest traktowany tak samo i nie ma żadnych uprzywilejowanych (specjalnych) serwerów, które miałyby większe uprawnienia lub obowiązki od innych. Sprawia to, że system nie posiada tak zwanego „single point of failure”, czyli słabego ogniska, którego awaria powoduje, że cały system staje się niefunkcjonalny lub wymaga czasochłonnej naprawy zanim zacznie znów funkcjonować poprawnie. W przypadku naszych baz sterownik używany przez klienta łączy się z jednym (dowolnym) z dostępnych serwerów, na których działa baza, i wykorzystuje ten serwer jako koordynatora swoich żądań. W razie awarii tego serwera lub jego przeciążenia sterownik może wybrać

dowolny inny serwer, na którym działa baza, i mianować go swoim nowym koordynatorem. Tak naprawdę każde żądanie może mieć innego koordynatora.

Partycjonowanie

Jak to w rozproszonych systemach bywa, dane trzeba jakoś podzielić między dostępne serwery. Proces ten nazywa się partycjonowaniem, ponieważ dane dzielone są na części (partycje) i każda z tych części zostaje przypisana do jednego konkretnego serwera, na którym będzie przechowywana. Służy do tego klucz partycji, na podstawie którego, w deterministyczny sposób, wyliczany jest hash definiujący, do którego serwera należy partycja.

Replikacja

W celu zapewnienia bezpieczeństwa przechowywanych danych trzeba tworzyć ich kopie na serwerach innych niż ten, na którym znajduje się oryginalna wersja. Liczba tych kopii nosi nazwę `replication factor` (w skrócie RF) i jest w Cassandrze® i Scylli konfigurowalna per keyspace, co oznacza, że każdy może mieć inny RF. Keyspace to jednostka grupująca tabele (przestrzeń nazw). Baza może posiadać wiele keyspace'ów. Aby dane były bezpieczne, trzeba mieć przynajmniej 2-3 repliki danych. W opisywanych bazach repliki danych nie są rozróżnialne i nie ma oryginalnych danych i kopii. W praktyce wszystkie repliki traktowane są równorzędnie i zawsze znajdują się na różnych serwerach.

Zapis danych

Każdy zapis danych jest wysyłany przez klienta do koordynatora żądania. Koordynator przesyła ten zapis do wszystkich serwerów, na których znajdują się kopie zmienianej partycji. Repliki zapisują otrzymane dane i wysyłają koordynatorowi potwierdzenie tej operacji. Każde żądanie zapisu posiada parametr zwany `consistent-`

reklama

Szkolenie dla Ciebie lub Twojego zespołu

Bazy danych NoSQL - Apache Cassandra

Zdobądź ogólną wiedzę dotyczącą baz typu NoSQL, ich funkcjonalnościach, zastosowaniach i ograniczeniach. Szkolenie, poza ogólnym wprowadzeniem do baz nierealacyjnych, skupia się na bazie danych Cassandra, stworzonej pierwotnie przez Facebook'a.

Skorzystaj z 10% zniżki na wszystkie szkolenie otwarte z autorskiej oferty Sages ważnej przy zamówieniach złożonych do końca czerwca 2018 r.
Hasło: PROGRAMISTAMAG

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

cy `level` (w skrócie CL), który decyduje, od ilu replik koordynator musi otrzymać potwierdzenie zapisu danych zanim potwierdzi klientowi pozytywne zakończenie zapisu.

Odczyt danych

Odczyt danych jest analogiczny do zapisu. Klient wysyła do koordynatora żądanie odczytu wraz z parametrem `consistency level`. Koordynator wysyła zapytania o dane do CL replik i czeka na odpowiedzi od nich. Po ich otrzymaniu dokonuje rozwiązania ewentualnych konfliktów ze spójnością danych otrzymanych z różnych kopii i zwraca rozwiązanie do klienta. Takie konflikty mogły powstać, ponieważ jakiś zapis mógł wykonać się już na niektórych replikach, a na innych jeszcze nie. By móc rozwiązać niespójności, każdy fragment danych w bazie jest opatrzony stemplem czasowym. Konflikty rozwiązywane są najprostszą możliwą metodą, czyli ostatni wygrywa. Przesyłanie wszystkich danych z replik do koordynatora w przypadku braku konfliktów byłoby marnotrawstwem. Z tego powodu przesyłane są jedynie kryptograficzne skróty danych, które pozwalają stwierdzić, w którym miejscu dane się różnią. W przypadku wykrycia konfliktu koordynator prosi repliki o tę część danych, która jest niespójna, i na niej dokonuje rozwiązania konfliktu. Bardzo istotne jest, by wszystkie serwery, na których działa baza, miały zsynchronizowany czas systemowy, ponieważ na jego bazie ustalane są stemple czasowe.

Spójność

Taki sposób obsługi zapisów i odczytów nie zawsze gwarantuje spójność danych. Niezbędny jest uważny dobór parametrów CL zapisu i CL odczytu względem parametru RF. Generalna zasada jest taka, że spójność danych jest zachowana, gdy CL zapisu + CL odczytu > RF. Daje to spore pole manewru do optymalizacji wydajności bazy pod charakterystykę otrzymywanej ruchu. Na przykład gdy wiemy, że dane są zapisywane często, ale rzadko odczytywane, możemy wybrać CL zapisu = 1 i CL odczytu = RF. Wtedy zapisy będą wykonywane szybko, co ważne, bo jest ich dużo, kosztem czasu wykonania odczytów, które są jednak rzadkie, więc jesteśmy w stanie się z tym pogodzić. Przy odwrotnej charakterystyce ruchu wystarczy wybrać CL zapisu = RF i CL odczytu = 1. Gdy nie jesteśmy pewni, jak wygląda ruch do naszej bazy, możemy zdecydować się na bezpieczne CL zapisu = RF / 2 + 1 i CL odczytu = RF / 2 + 1. Czasem gdy nie zależy nam na natychmiastowej spójności, możemy wybrać inne wartości CL zapisu i CL odczytu, licząc na to, że mechanizmy opisane w następnej sekcji, z czasem, rozwiązają problemy ze spójnością danych.

Walka z entropią

W systemach rozproszonych może się zdarzyć, że kopie danych na różnych serwerach mogą tracić spójność na stałe. Na przykład na

skutek problemów z siecią, w wyniku których zginął pakiet z żądaniem zapisu danych. Nie jest to problem, ponieważ koordynator żądania odczytu jest w stanie rozwiązać takie konflikty. Wymaga to jednak więcej pracy, której najczęściej chcielibyśmy uniknąć. W tym celu powstał mechanizm napraw podczas odczytów. Działa on w ten sposób, że koordynator, który wykryje konflikt i go rozwiąże, wysyła poprawne dane do wszystkich replik, od których otrzymał nieaktualne dane, by mogły poprawić swoje partycje. Dodatkowo, co jakiś czas, koordynator decyduje się wysłać zapytanie o dane do wszystkich replik, a nie tylko do CL z nich. Dzięki temu naprawa podczas odczytu ma szansę uspójnić większą liczbę replik. W Cassandra® i Scylli istnieje jeszcze mechanizm o nazwie hinted-handoff, który sprawia, że chwilowa niedostępność serwera nie sprawia problemów ze spójnością danych. Mechanizm ten polega na tym, że koordynator zarządzający zapisem, gdy widzi, że jakiś serwer, na którym powinien odbyć się zapis, jest niedostępny, zapisuje sobie informację o tym, że zapis ominął ten serwer, i wysyła go ponownie, gdy serwer staje się znów dostępny.

Plotkowanie

Klastry Cassandra® i Scylli są dynamiczne. Serwery są do nich dołączane i od nich odłączane w trakcie działania bazy. Jest to nieuniknione, ponieważ sprzęt zwyczajnie się psuje, co powoduje awarię serwera. Dodatkowo obserwując zwiększyły ruch do bazy, trzeba reagować dodaniem większej liczby serwerów, by wsparły obsługę żądań. Powoduje to zmiany w strukturze klastra. Każdy serwer trzyma informacje o tym, jak zbudowany jest klaster, by wiedzieć, na których maszynach znajdują się kopie wybranej partycji. Aby posiadać aktualne informacje, każdy serwer „plotkuje” z pozostałymi serwerami. W regularnych interwałach czasowych wybiera kilku „kolegów” i przekazuje im wszystkie informacje o budowie klastra, jakie posiada. W efekcie wystarczy kilka cykli „plotkowania”, by zmiana danych rozpropagowała się do wszystkich maszyn w klastrze.

PRZECHOWYWANIA DANYCH NA SERWERZE

Przyjrzyjmy się teraz, w jaki sposób Cassandra® i Scylla przechowują dane na serwerze. Mechanizmy, na których oparta jest praca serwera, pochodzą w dużej mierze z bazy Google Big Table. W tym celu zbadamy dwie perspektywy: zapis danych i ich odczyt.

Zapis danych

Zapis danych składa się z kilku faz. W pierwszej fazie dane zapisywane są do struktury w pamięci operacyjnej zwanej Memtable oraz do sekwencyjnego logu zapisów o nazwie Commitlog, któ-

ry znajduje się na dysku twardym. W tym momencie dane są już na trwałe zapisane na dysku, więc serwer może wysłać do koor-dynatora potwierdzenie, że operacja się powiodła. Sprawia to, że zapisy są bardzo szybkie, ponieważ wymagają jedynie zapisu do pamięci operacyjnej i sekwencyjnego zapisu na dysku. Pamięć operacyjna to jednak skończony zasób, więc nie można w nieskończoność dodawać do niej nowych zapisów. Gdy zużycie pamięci operacyjnej osiągnie pewien odpowiednio duży poziom, Memtable zostaje zapisane na dysk w postaci plików SSTable. Pliki te są niezmienialnymi plikami przechowującymi posortowane rekordy z danymi. Dla każdego pliku SSTable zapisywane są pomocnicze pliki usprawniające odczyt danych, ale o tym więcej w następnej sekcji. Funkcją Commitlogu jest zabezpieczenie danych, które znajdują się w Memtable, czyli nie zostały jeszcze zrzucone na dysk w postaci plików SSTable. Jeśli serwer ulegnie awarii, dane zostaną odtworzone z Commitlogu.

Odczyt danych

Odczyt danych odbywa się równolegle z Memtable i wszystkich plików SSTable. Na tej podstawie wyliczana jest aktualna wartość danych, która zostaje zwrócona do koordynatora. Brzmi prosto, ale bez dodatkowych usprawnień byłoby to bardzo wolne rozwiązanie. Największy problem stanowi odczyt plików SSTable z dysku. By go uniknąć, stosuje się tak zwane Bloom Filters. Jest to probabilistyczna struktura danych, która pozwala odpowiedzieć na pytanie, czy dany klucz znajduje się w wybranym pliku SSTable. Dzięki temu, że struktura jest probabilistyczna, jej szybkość działania jest bardzo duża, kosztem ewentualnych błędnych odpowiedzi, które mogą się zdarzyć odpowiednio rzadko. Ważne jest to, że błąd może jedynie polegać na stwierdzeniu, że klucz znajduje się w danym pliku SSTable, mimo że go tam nie ma. Nigdy zaś na uznaniu, że klucza w danym pliku nie ma, mimo że tak naprawdę tam się znajduje. Sprawia to, że ewentualne błędne odpowiedzi z Bloom Filters skutkują dodatkową (niepotrzebną) pracą, ale nigdy błędny wynikiem.

Drugim usprawnieniem są indeksy dla pliku SSTable pozwala-jące na odnalezienie danych dla wybranego klucza bez konieczno-sci odczytu całego pliku z danymi. Indeksy mają wiele poziomów, by umożliwić zapamiętanie przynajmniej części z nich w pamięci operacyjnej.

Kompakcja

Jak łatwo zauważać, liczba plików SSTable zapisanych na dysku ma wpływ na szybkość odczytu danych. Im więcej jest plików SSTable, tym dłużej zajmuje pobranie z nich danych. By umożliwić ba-zie działanie przez wiele dni i tygodni bez przerwy, wprowadzono pomocniczy proces działający w tle o nazwie Kompakcja. Celem tego procesu jest łączenie kilku plików SSTable w jeden. Taka ope-racja nie tylko zmniejsza zużycie pamięci dyskowej. Dzieje się tak, ponieważ dwa pliki SSTable mogą zawierać zapis do tej samej komórki w tabeli i wtedy wystarczy zachować najnowszy z tych za-pisów, a pozostałe usunąć, ponieważ są one nadpisane przez ten najnowszy.

Kompakcja w zamyśle ma na celu usprawnienie pracy bazy, ale istnieje ryzyko, że stanie się zupełnie odwrotnie. Działanie Kom-pakcji wymaga użycia procesora i przepustowości dysku. Sprawia to, że zostaje mniej tych zasobów dla procesów przetwarzających aktualne żądania. Jest to częsty problem w Cassandra®, który można częściowo rozwiązać odpowiednim strojeniem prędkości Kompakcji. Tak wystrójona baza działa dobrze dla wybranej charakteryistyki ruchu, ale gdy ruch do bazy się zmieni, musi zostać od nowa wystrójona. Scylla rozwiązuje ten problem za pomocą automatycznego strojenia, który adaptuje się do aktualnej charakteryistyki ruchu i zapewnia dobry balans między Kompakcją a procesami przetwarzającymi aktualne żądania.

NA ZAKOŃCZENIE

Podsumowując: Cassandra® i Scylla są przykładami wydajnych sys-temów rozproszonych opartych na architekturze peer-to-peer bez wybierania liderów. Obie skalują się liniowo i są w stanie poradzić sobie z ewentualnymi problemami ze sprzętem, na którym działa-ją. Scylla dodatkowo zapewnia niskie i stabilne czasy obsługi żądań oraz dobrą utylizację sprzętu, na którym działa. Są to rozwiązania warte rozważenia przez tych, którzy potrzebują wydajnych baz da-nych, będących w stanie obsłużyć setki tysięcy lub miliony operacji na sekundę. Dodatkowo są to rozwiązania o wysokiej dostępności. Wszystkie operacje administracyjne można wykonywać na działa-jącej bazie bez konieczności jej zatrzymywania.



PIOTR JASTRZĘBSKI

Inżynier oprogramowania z dziesięcioletnim stażem. Aktualnie pracuje nad rozproszoną bazą danych NoSQL – Scylla. Wcześniej rozwijał real-time trading system w londyńskim City oraz pracował nad systemem Android w firmie Google. Absolwent informatyki na Uniwersytecie Warszawskim.

Testować bezpieczeństwo – co to znaczy?

Każdy pracujący w świecie IT zetknął się z terminem testowania bezpieczeństwa. Ale tak naprawdę co to znaczy przeprowadzić testy bezpieczeństwa? Czy jest to tym samym co testy oprogramowania albo testy penetracyjne? Jeśli nie, czym więc te czynności się różnią? W końcu kto powinien takie testy przeprowadzać? Spróbujmy odpowiedzieć na te pytania i przybliżyć pułapki, w które możemy wpaść, jeśli nie będziemy dokładnie rozróżniać wspomnianych zagadnień.

TESTOWANIE BEZPIECZEŃSTWA

Zacznijmy od pojęcia testowania bezpieczeństwa stosowanego przez organizację ISTQB¹, zrzeszającą specjalistów testowania oprogramowania. Definicja ta (Rysunek 1) mówi, że jest to testowanie ukierunkowane na ocenę poziomu bezpieczeństwa oprogramowania. Nie jest to zbyt precyzyjne określenie, ale wskazuje kierunek działań i podkreśla potrzebę uzyskania wiedzy o tym, jak bardzo bezpieczny jest produkt – tutaj oprogramowanie.

security testing

Testing to determine the security of the software product.

Synonyms: —

See Also: functionality testing

Rysunek 1. Testowanie bezpieczeństwa wg ISTQB

Osoba zajmująca się bezpieczeństwem rozszerzyłaby tę definicję o dodatkowe obszary, które mogą podlegać testowaniu bezpieczeństwa. A te mogą być dużo bardziej rozległe, niż samo oprogramowanie. Specjalista bezpieczeństwa wskazałby tutaj także między innymi architekturę rozwiązania, infrastrukturę, na której oprogramowanie będzie funkcjonować, środowisko i procesy, które będą związane z rozwojem i utrzymaniem oprogramowania. Nie zapominajmy także o szeregu administracyjnych obszarów, wymagających rygoru i podążania za oficjalną procedurą, które także powinny podlegać testom i wnikliwemu sprawdzeniu. Wszystkie powyższe, oraz także inne niewymienione tutaj, obszary wpływają przecież na bezpieczeństwo informacji przetwarzanych przez oprogramowanie oraz bezpieczeństwo całej firmy z punktu widzenia biznesowego.

Poza różnicą w określeniu, czym jest przedmiot testowania, specjalista bezpieczeństwa zwróciłby także uwagę na różnice w tym, czym jest oczekiwany rezultat i w sposobie jego prezentacji. Testy oprogramowania, w rozumieniu branżowym, mają na celu odpowiedzieć na szereg pytań związanych między innymi ze zgodnością ze specyfikacją, wymaganiami biznesowymi, odpowiednią wydajnością, poprawnością działania, poprawną reakcją na błędne lub nieoczekiwane dane wejściowe lub detekcją niestandard-

dowego zachowania danego komponentu, aplikacji lub systemu. W wielkim uogólnieniu taka odpowiedź najczęściej powinna być binarna i określać, czy dany test dał pozytywny lub negatywny rezultat – w uproszczeniu zwrócić PASS lub FAIL. I nawet w ujęciu testów, które takich binarnych odpowiedzi bezpośrednio nie dają, otrzymaną wartość przyrównujemy później do wymagań lub ograniczeń (np. badając wydajność lub mierząc czas odpowiedzi). Robimy tak po to, by określić, czy uzyskana wartość jest akceptowalna, czy wymagane jest dalsze przeprowadzenie analizy i usprawnienie działania. Wskazuje to też, że testowanie oprogramowania w ujęciu ogólnym ukierunkowane jest na działanie – tj. zidentyfikowanie problemu i umożliwienie jego korekcji. Rezultatem testowania są zgłoszone błędy i defekty, a w szerszej perspektywie celem jest dokonanie oceny liczby błędów oraz ich dotkliwości – wszystko w celu oszacowania nakładu pracy potrzebnego do naprawy.

Jest to zasadnicza różnica względem testowania bezpieczeństwa. Tam gdzie tester oprogramowania będzie poszukiwał defektów, specjalista od bezpieczeństwa będzie starał się określić poziom bezpieczeństwa, przez znalezienie luk lub obszarów wymagających poprawy. Innymi słowy, testowanie bezpieczeństwa będzie weryfikowało, czy na każdym z etapów tworzenia produktu stosowano dobre praktyki, spełniono wymagania bezpieczeństwa oraz zaimplementowano mechanizmy, które to bezpieczeństwo zapewnią. Rodzi się tutaj pierwsze ważne pytanie. Jak powinien wyglądać opis wyników testu bezpieczeństwa, skoro chcemy uzyskać wiedzę o tym, jak bezpieczny jest system bądź aplikacja? Liczba znalezionych defektów lub luk nie powie nam wprost, w jakim stopniu system zapewnia bezpieczeństwo przetwarzanych danych. Nie dostaniemy też informacji o tym, czy możliwe jest ominięcie niektórych mechanizmów bezpieczeństwa, na przykład ominięcie procesu uwierzytelniania. Aby oddać istotę problemu i ich wpływ na bezpieczeństwo, wymagane jest zatem inne podejście do opisywania znalezionych defektów. W celu właściwego przedstawienia stanu systemu specjalista bezpieczeństwa będzie mówił w raporcie o następujących elementach:

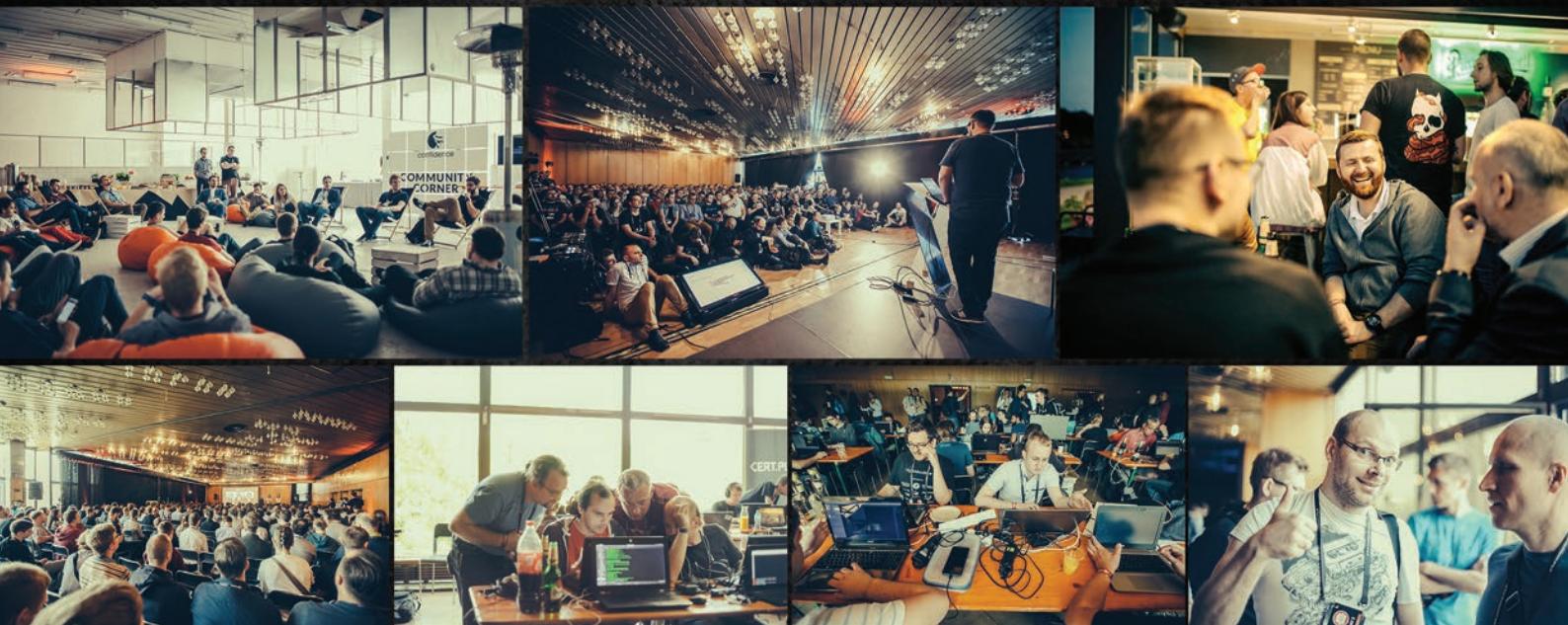
- » Poufność (ang. *confidentiality*) – zapewnienie, że informacje są czytelne lub dostępne tylko dla upoważnionych osób (lub procesów);
- » Spójność (ang. *integrity*) – zapewnienie, że informacje zachowają swoją strukturę i nie nastąpi ich modyfikacja przez osoby (lub procesy) nieupoważnione, bądź taka modyfikacja zostanie łatwo wykryta;
- » Dostępność (ang. *availability*) – zapewnienie, że informacje,

1. International Software Testing Qualifications Board – www.istqb.org



Konferencja IT security w Krakowie!

4-5 czerwca 2018,
Muzeum Lotnictwa Polskiego



Bilety 20% taniej
z kodem: **PROGRAMISTA20**

usługi lub akcje będą dostępne w chwili, gdy będzie to wymagane, a niedostępność będzie na poziomie akceptowalnym.

Specjalista bezpieczeństwa będzie bazował głównie na swoim doświadczeniu oraz wiedzy branżowej, aby odpowiednio ocenić, jak duży wpływ znaleziona podatność może mieć na bezpieczeństwo systemu. Istnieje wiele metryk, które ułatwiają zapisanie wyniku takiej oceny. Najpopularniejszą z nich jest CVSS² (aktualnie w wersji 3). Pozwala ona na ocenę wspomnianych już poufności, spójności oraz dostępności, a także na uwzględnienie w ocenie wektora ataku, poziomu jego skomplikowania, potrzebnych uprawnień, by taki atak przeprowadzić, oraz tego, czy potrzebna jest interakcja ze strony użytkownika. Ponadto podstawowa metryka CVSS (ang. CVSS Base Score) może zostać wzbogacona o dodatkowe metryki:

- » *Temporal* – tj. metrykę zawierającą informacje, które mogą się zmieniać w czasie (np. związanych z dostępnością gotowych bibliotek – exploitów – do przeprowadzania ataku lub dostępnością aktualizacji, która ten atak uniemożliwi);
- » *Environmental* – tj. metrykę niosącą informację o środowisku, w jakim dana luka bezpieczeństwa została znaleziona, i tym, jak to wpływa na poziom zagrożenia niesionego przez tę lukę.

Metryka CVSS, poza ustandaryzowanym sposobem opisu podatności, pozwala także na otrzymanie wartości liczbowej, którą możemy przekładać na krytyczność luki i tym samym używać jej, by porównywać podatności w celu określenia dalszych priorytetów. W raporcie z testów bezpieczeństwa zobaczymy zatem raczej ciąg „CVSS v3.0 Base Score: 5.9”, niż informację PASS lub FAIL. Czasami pełny wektor metryki zostanie przedstawiony, ujawniając odbiorcy raportu poszczególne wartości składające się na wynik CVSS: AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H. Przykładowy pełny opis podatności dla CVE-2018-0002 przedstawiony został na Rysunku 2.

CVSS v3.0 Severity and Metrics:

Base Score: 5.9 MEDIUM

Vector: AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:H (V3 legend)

Impact Score: 3.6

Exploitability Score: 2.2

Attack Vector (AV): Network

Attack Complexity (AC): High

Privileges Required (PR): None

User Interaction (UI): None

Scope (S): Unchanged

Confidentiality (C): None

Integrity (I): None

Availability (A): High

Rysunek 2. CVSS v3 Base Score dla CVE-2018-0002

TESTY PENETRACYJNE

Kolejnym zagadnieniem, które silnie wiąże się jednocześnie z testowaniem oprogramowania oraz bezpieczeństwem, są testy penetracyjne, potocznie nazywane pentestami. Uznaje się, że testy penetracyjne są szczególnym przypadkiem testowania bezpieczeństwa, gdzie punktem zainteresowania jest także bezpieczeństwo produktu lub środowiska, lecz w innym ujęciu. Cechy charak-

terystyczne dla pentestów uwidaczniają się zarówno w określeniu celu, w jakim je stosujemy, jak i w sposobie ich przeprowadzania. Testy penetracyjne mają za zadanie wskazać możliwość włamania lub możliwość przeprowadzenia akcji czy wpłynięcia na działanie systemu w sposób nieuprawniony lub nieprzewidziany przez twórców. To, co jest tutaj kluczowe, to ograniczenie celu do wskazania samej możliwości. Wynika z tego, że testy penetracyjne nie są w swojej naturze kompletne, w sensie takim, jaki rozumiemy przy typowym testowaniu oprogramowania. Testy penetracyjne nie wskazują wszystkich defektów i problemów, a ich kompletność będziemy rozumieć raczej jako przetestowanie każdego komponentu oraz punktu interakcji z użytkownikiem, niż sprawdzenie każdej możliwej podatności czy każdego wymagania. Testy penetracyjne powinny być więc uzupełnieniem do całej gamy testów oprogramowania i testów bezpieczeństwa. W Luxoft dla przykładu idziemy o krok dalej i mówimy całosciowo o bezpiecznym procesie tworzenia kodu SDLC, zamiast tylko o różnych typach testowania.

Jak już wspomniano wyżej, charakterystyczne jest też samo przeprowadzanie testów penetracyjnych. Próźno szukać tutaj szczegółowych planów testowych czy scenariuszy testowych. Zamiast tego zobaczymy jasno określony zakres, opisujący to, co można robić, a czego kategorycznie nie. Na przykład często będziemy mogli przeprowadzać ataki *brute-force* na kontach użytkowników, ale nie dostaniemy zgody na wykonanie ataków DDoS. Ważne też będą ramy czasowe i podstawowe informacje o tym, gdzie testy należy zacząć (np. adres IP lub adresy aplikacji webowych). Testy penetracyjne są zadaniem bardziej otwartym i kreatywnym, wymagającym myślenia niestandardowego, ale równocześnie mocno analitycznego. Celem jest sprawne znalezienie możliwego wektora ataku i użycie go, by głębiej przeszukiwać system lub środowisko. Stąd kluczowe jest ograniczenie czasowe dla testów tego typu, gdyż takie poszukiwania mogłyby się ciągnąć miesiącami, z każdym kolejnym dniem zagłębiając się coraz bardziej w strukturę kodu czy nawet analizować warstwę fizyczną sprzętu.

W zależności od tego, na jaki rodzaj testu zdecydowało się kierownictwo projektu, pentester (bo tak popularnie nazywamy osobę przeprowadzającą testy penetracyjne) będzie miał różny dostęp do informacji o testowanym obszarze. Najogólniejszy i zarazem najpopularniejszy stosowany podział to:

- » *Black-box* – pentester nie będzie miał dostępu do informacji wewnętrznych, a jego testy (ataki) będą najbardziej zbliżone do tego, co mógłby wykonać potencjalny atakujący;
- » *Gray-box* – pentester będzie miał dostęp do części informacji (niedostępnych zwykle dla atakującego) takich jak architektura, nazwy serwerów czy dane logowania do części aplikacji;
- » *White-box* – pentester będzie miał dostęp do wszystkich informacji wewnętrznych, włączając w to dokumentację techniczną oraz kod testowanej aplikacji lub systemu.

Różny dostęp do informacji rzutuje na to, jak wiele oraz jakiego rodzaju wyniki otrzymamy. Udostępniając więcej informacji, możemy spodziewać się wyników bardziej dokładnych, często wprost wskazujących na to, jak dany problem należy skorygować. Równocześnie pentester najprawdopodobniej wskaza tych problemów więcej, niż w przypadku testów *black-box*, gdyż nie będzie spędzał czasu na pozyskiwaniu wstępnie niedostępnych, a potrzebnych do przeprowadzenia ataku informacji. W podejściu tym jednak nie uda się nam uchwycić w pełni warunków takich, jakie test penetracyjny powinien symulować – czyli sytuacji atakującego, który tych informacji zwykle nie posiada. Istnieje też ryzyko, że pentester zbyt

2. Common Vulnerability Scoring System – www.first.org/cvss

mocno skupi się na przekazanych informacjach i raport będzie pełen podatności, które moglibyśmy wykryć w standardowych testach bezpieczeństwa. Musimy pamiętać, że testy penetracyjne powinny być uzupełnieniem do klasycznego testowania bezpieczeństwa i powinny wskazać bardziej złożone problemy oraz nietypowe wektory ataku. Mają one wykryć problemy, na które często składać się będzie kilka różnych podatności w różnych częściach naszego środowiska. Podejście *gray-box* jest tu dobrym kompromisem, pozwalającym na zwiększenie dokładności oraz ograniczenie czasu testów penetracyjnych, a co za tym idzie, kosztu ich przeprowadzenia. Z doświadczenia, jakie zebraliśmy w ramach globalnego centrum kompetencji InfoSec firmy Luxoft, wynika, że to właśnie podejście *gray-box* jest najczęściej wybieranym modelem współpracy, a zakres dostępnych informacji jest każdorazowo proponowany w oparciu o potrzeby i wymagania klienta.

Faza testów penetracyjnych kończy się przedstawieniem raportu. Dokument ten jest zupełnie inny od tego, co zobaczylibyśmy na koniec fazy testowania oprogramowania lub testów bezpieczeństwa. W raporcie pentester starać się będzie udokumentować problemy, jakie udało mu się zidentyfikować w trakcie pracy. Będzie on często stosował te same metryki, co przy testach bezpieczeństwa (np. CVSS). Niemniej jednak, aby wskazać możliwość pomyślnego ataku, pentester będzie musiał jeszcze przedstawić scenariusz ataku oraz *proof-of-concept*. Mają one na celu wskazanie odbiorcy, jak atak mógłby zostać przeprowadzony i które składowe podatności oraz problemy zostały wykorzystane, by tego dokonać.

W ten sposób testy penetracyjne wskazują możliwość uzyskania dostępu do systemu lub wpłynięcia na jego działanie oraz dokumentują, jak duży wpływ dany atak ma na bezpieczeństwo (danych, użytkowników, ciągłości biznesu itd.). Z raportu nie dowiemy się natomiast tego, jak bardzo bezpieczny jest system ani ile defektów się w nim znajduje. Dowiemy się natomiast, jak i czy w ogóle można się do niego włamać, poświęcając wskazany czas i przeznaczając odpowiednie zasoby. Dowiemy się też, jakie to może mieć skutki dla naszego środowiska.

UMIEJĘTNOŚCI I DOŚWIADCZENIE

Niezwłekłe ważne dla każdego profilu stanowiska branży IT jest określenie zakresu wymaganych umiejętności oraz niezbędnego doświadczenia, jakie powinien posiadać pracownik wykonujący dany zestaw zadań.

Częstym błędem jest myślenie, że testy bezpieczeństwa lub testy penetracyjne można powierzyć testerom oprogramowania, np. tym ukierunkowanym na testy funkcjonalne. Odwrotne założenie także jest fałszywe i przynosi fatalne skutki, gdy zastosowane w praktyce. Ponieważ cele obu tych testów są diametralnie różne, tak samo jak sposób ich przeprowadzania, nie możemy tych zagadnień uogólniać i traktować jako po prostu testowania.

O ile wykonywanie typowych testów oprogramowania oparte o scenariusze i procedury testowe nie różni się znacząco od wykonania testów bezpieczeństwa, o tyle różnice w potrzebnym doświadczeniu i wiedzy będą już wyraźnie widoczne przy planowaniu obu tych testów oraz opisie otrzymanych wyników. Tam gdzie tester oprogramowania chciałby powiedzieć, że test dał negatywny rezultat, dla testu bezpieczeństwa potrzebne będzie dodatkowo określić, jaki wpływ na bezpieczeństwo ma znaleziony błąd oraz jakie zagrożenia i ataki mogą wystąpić w związku ze znalezionym defektem. Inaczej też będziemy patrzyć na budowę planu testowania, bo ten będzie wymagał od pracownika dokładnej wiedzy

NIE
BÓJ SIĘ MÓWIĆ!
POSTAW NA ANGIELSKI
BEZ OGRANICZEŃ!



SPECJALISTYCZNE
SŁOWNICTWO
Z BRANŻY IT



SPRAWNE
TŁUMACZENIE
OPROGRAMOWANIA



videopoint

Helion



z zakresu ataków, zagrożeń, typowych błędów oraz zakresu stosowania dobrych praktyk. Wiedzy tej nie będziemy natomiast wymagać od osoby planującej typowe testy oprogramowania. Z drugiej strony natomiast, jeśli chcielibyśmy, aby tester bezpieczeństwa wykonał i opisał testy funkcjonalne, możemy spodziewać się, że jego wiedza techniczna z zakresu działania testowanej aplikacji i zrozumienie oczekiwanych wyników nie będzie często wystarczająca do rzetelnej oceny znalezionego problemu, jak by to miało miejsce, gdyby wszystko wykonywał tester oprogramowania bądź specjalista do spraw jakości oprogramowania.

Inaczej sprawa ma się w przypadku testów penetracyjnych, jak i również innych typów testów bezpieczeństwa, które charakteryzują się eksploracyjnym podejściem (na przykład fuzz-testy lub dynamiczne testy bezpieczeństwa aplikacji DAST). Specyfika pracy pentestera wymaga kreatywnego podejścia do przeprowadzanych działań, często wykraczających poza standardowe lub znane zabiegi. Często polega ona wykorzystaniu intuicji, adaptacji podobnych strategii do nowych obszarów lub implementacji własnych lub zmodyfikowanych *exploitów*, tak by krok po kroku pokonywać i budować kolejne etapy, które złożą się na końcowy atak. Niezwykle ważne jest zatem doświadczenie w przeprowadzaniu testów penetracyjnych połączone z wiedzą ekspercką. Dopiero razem pozwalały one na pełne wykorzystanie eksploracyjnej charakterystyki testów penetracyjnych i znajdowanie skomplikowanych problemów, które nie zostały wykryte podczas wcześniejszych testów (włączając w to testy bezpieczeństwa).

Dla przykładu w Luxoft łączymy najlepszych specjalistów w ramach globalnych centrów kompetencji. Jednym z nich jest globalna praktyka InfoSec ze swoim centrum w Krakowie, w ramach

której skupiamy – między innymi – wiedzę i doświadczenie dotyczące testowania oraz analizy bezpieczeństwa z wielu obszarów i z wielu branż. Świadome korzystanie z różnic, jakimi cechują się różne typy testów i tym samym różnic w wiedzy i umiejętnościach naszych specjalistów, wzmacnia nie tylko wzajemne kompetencje zespołu, ale także jakość wyników, które finalnie trafiają do naszych klientów.

PODSUMOWANIE

Oczywiście należy wspomnieć, że zarówno testy oprogramowania, jak i testowanie bezpieczeństwa to pojęcia bardzo pojemne, zawierające wiele podzbiorów i szczególnych przypadków testowania, które wzajemnie się przepłatają w zakresie definicji i stosowania. Najczęściej podziały starają się usystematyzować różne cele (np. testy funkcjonalne, wydajnościowe, bezpieczeństwa), różnych przedmiot testowania (np. testy jednostkowe, blokowe, systemowe) lub różne metody wykorzystywane w trakcie testowania (np. fuzz-testy, dynamiczne testowanie bezpieczeństwa aplikacji DAST, statyczna analiza kodu). Powyżej starano się odzwierciedlić najistotniejsze różnice, które przekładają się na to, jak należy interpretować wyniki lub jakich cech szukać u specjalisty, który te testy będzie wykonywał.

Najczęściej problemy z tematyką testowania bezpieczeństwa wynikają z nieznajomości tych różnic (Tabela 1). Tylko świadomie określając cel testów oraz oczekiwane przez nas wyniki, jesteśmy w stanie odpowiednio dobrze specjalistów, którzy adekwatnie zaplanują prace, przeprowadzą testy oraz trafnie opiszą rezultaty.

	TESTY		
	Ogólnie rozumiane	Bezpieczeństwa	Penetracyjne
Cel	Określenie zgodności ze specyfikacją i wymaganiami	Określenie poziomu bezpieczeństwa	Wskazanie możliwości pomyślnego ataku (włamania, zmiany działania)
Wyniki	Defekty	Podatności i rekomendacje poprawy	Możliwości ataku
Opis	PASS/FAIL	Krytyczność i/lub ryzyko	Scenariusze ataku i wpływ na bezpieczeństwo
Dane bazowe	Znajomość testowanego obiektu Specyfikacja oraz wymagania Plan testów oraz scenariusze testowe	Doświadczenie i dobre praktyki Wymagania bezpieczeństwa Plan testów oraz scenariusze testowe	Zakres dozwolonych działań Ramy czasowe Dane początkowe (np. adres IP)

Tabela 1. Różnice pomiędzy różnymi typami testów



MARCIN ŚWIĘTY

marcin@swiety.eu.org

Globalny dyrektor ds. cyberbezpieczeństwa w Luxoft. Doświadczony ekspert zajmujący się projektowaniem i zarządzaniem usługami InfoSec dla globalnych firm. Z zamiłowaniem white-hat i entuzjasta CTF. Posiada certyfikaty CISSP, CISM, CISA, CEH, WCSD i ITIL. Pasjonuje się tym, w jaki sposób InfoSec wpływa na strategie biznesowe globalnych korporacji w erze transformacji cyfrowej.

Programiści .NET już po raz piąty spotkają się w Warszawie!

18-19 września to ważne daty dla wszystkich pracujących w technologii .NET! Właśnie wtedy w Hali EXPO XXI odbędzie się piąta edycja stynnej międzynarodowej konferencji .NET DeveloperDays.

Całodzienne prelekcje poświęcone konkretnemu zagadniению w przededniu konferencji (pre-cons), równolegle ścieżki tematyczne prowadzone przez ekspertów z całego świata oraz impreza integracyjna podczas wydarzenia – tak w dużym skrócie zapowiedzieć można kolejną edycję popularnej konferencji dla programistów .NET.

– W związku z tym, że to już piąta edycja, postanowiliśmy wprowadzić małe zmiany w dotychczasowej formule. Nowością będą aż 4 równolegle sesje tematyczne zamiast 3. Co to oznacza dla uczestników? Możliwość wyboru interesujących zagadnień z zakresu technologii .NET z jeszcze większej puli. Dzięki temu zarówno osoby zaawansowane, pracujące od lat z tą technologią, jak i stawiające pierwsze kroki będą mogły uczestniczyć w prelekcjach odpowiadającym ich poziomowi wiedzy – mówią organizatorzy.

W 4. edycji wzięło udział prawie tysiąc uczestników, odbyło się 29 sesji tematycznych, poprowadzonych przez 14 ekspertów

z całego świata. W tym roku organizatorzy stawiają poprzeczkę jeszcze wyżej i do podzielenia się swoją wiedzą zapraszają aż 20 specjalistów!

Wśród nich znajdzie się niekwestionowany „zwycięzca” z zeszłego roku – Sasha Goldstein. Prowadzona przez niego sesja pt. „How I Built An Open-Source Debugger” została oceniona na 4,80 w pięciopunktowej skali, a jego pre-con pt. „Production Performance and Troubleshooting of .NET Applications” zdobył notę 4,96! Oprócz niego na liście prelegentów znajdują się także zeszłoroczní eksperci – Neal Ford i Daniel Marbach, a także nowy prowadzący – Shawn Wildermuth.

Kolejne nazwiska prelegentów będą uzupełniane na bieżąco na stronie konferencji: <http://net.developerdays.pl> oraz na facebookowym profilu: <https://www.facebook.com/DeveloperDays/>, a agenda konferencji zostanie ogłoszona w czerwcu.

Zobaczcie, co o poprzedniej edycji sądzą jej uczestnicy!

Maciej Pawlina z Mazowieckiego Klastra ICT

„To była moja pierwsza konferencja .NET DeveloperDays. Udział w niej to niepowtarzalna okazja do spotkania i dyskusji z ludźmi o zaawansowanej wiedzy praktycznej w tej dziedzinie. Wydarzenie wyjątkowe w branży. Z wielką przyjemnością wezmę udział w kolejnych edycjach!”

Paweł Łukasik – bloger IT

„Konferencja od kilku lat wpisuje się w mój kalendarz obowiązkowych wydarzeń programistycznych w Polsce. Edycja 2017 i tym razem nie zawiodła: dostarczyła zarówno merytorycznych treści, jak i umożliwiła kuluarowe rozmowy pomiędzy programistami i programistkami. Z niecierpliwością czekam na kolejną edycję.”

Ewelina Winska z organizacji GirlsInTech

„Najbardziej ucieśniała mnie możliwość poznania i rozmowy z Michele Leroux Bustamante. To było bardzo inspirujące. Michele zajmowała wysokie stanowiska kierownicze w kilku korporacjach i chętnie dzieli się swoją wiedzą. Spotkanie z nią zdecydowanie zmotywowało mnie do dalszego rozwoju.”



5 EDYCJA
NAJWIĘKSZEJ
KONFERENCJI
DEDYKOWANEJ PLATFORMIE .NET!

(Nie) bezpieczeństwa JWT (JSON Web Token)

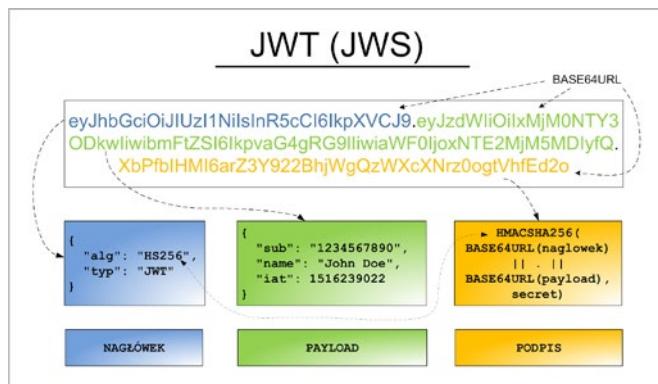
JWT (JSON Web Token) to mechanizm, który jest często wykorzystywany w kontekście API webowych, ale również szerzej – z powodzeniem używany jest w aplikacjach webowych czy mobilnych. JWT możemy znaleźć w popularnych standardach, jak np. OpenID Connect, spotkamy go również czasem, korzystając z OAuth2. Znajduje on zastosowanie zarówno w dużych firmach, jak i mniejszych organizacjach. Dostępnych jest wiele bibliotek obsługujących JWT, a sam standard posiada „bogate wsparcie dla mechanizmów kryptograficznych”. Czy to wszystko oznacza, że JWT jest mechanizmem z natury bezpiecznym? Na tą wątpliwość postaram się odpowiedzieć w dalszej części tekstu.

DEFINICJA

JSON Web Token to w największym skrócie metoda zapisu tzw. deklaracji (ang. *claims*) z wykorzystaniem notacji JSON. Jeśli chodzi o formalną definicję – warto zapoznać się ze stosownym dokumentem RFC (<https://tools.ietf.org/html/rfc7519>). Czytamy tutaj:

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure (...)

Tłumacząc prostym językiem – JWT to pewien ciąg znaków w formacie JSON (<https://www.json.org/>) zakodowany w strukturę JWS (JSON Web Signature) lub JWE (JSON Web Encryption). Dodatkowo każda z tych opcji musi być zserializowana w sposób kompaktowy (ang. *compact serialization* – to jedna z dwóch serializacji wyliczanych w JWS i JWE). Najczęściej w praktyce mamy do czynienia właśnie z **JWS i to ta struktura nazywana jest popularnie JWT**. Z kolei „deklaracja” to najczęściej prosta para typu „klucz”: „wartość” (więcej szczegółów tutaj: <https://goo.gl/pQRKt4>). Przykład konkretnego JWT zaprezentowano na Rysunku 1:



Rysunek 1. Przykład podstawowego JWT

Widac tutaj trzy długie ciągi znaków rozdzielone kropką. Struktura całości wygląda w następujący sposób:

nagłówek . payload . podpis

Powyższe trzy elementy są zakodowane algorytmem BASE64URL (który wygląda bardzo podobnie jak BASE64, przy czym znak plus (+) w wynikowym ciągu zamieniany jest na minus (-), z kolei slash (/) zastępowany jest podkreśleniem (_), nie ma tutaj również standardowego paddingu BASE64, złożonego normalnie ze znaków równości (=)).

Po rozkodowaniu powyższego ciągu (które można wykonać np. za pomocą serwisu <https://jwt.io/>) widzimy w pełni czytelny dla nas nagłówek, payload, a także podpis (tym razem w formie binarnej).

Celem tekstu nie jest całościowe wprowadzenie w świat JWT, jednak osobom, które chcą bardziej kompleksowo zapoznać się z tematyką, polecam następujące zasoby:

1. Wprowadzenie: <https://jwt.io/introduction/>
2. Więcej technicznych szczegółów: <https://goo.gl/reiHTt>
3. Najwięcej detali znajdziemy w dokumentach publikowanych przez projekt JOSE (Javascript Object Signing and Encryption – <https://tools.ietf.org/wg/jose/>). Poza JWT czy JWS mamy tu dodatkowo opis JWK (JSON Web Key), JWA (JSON Web Algorithms), wskazanie różnych zastosowań opisanych mechanizmów (w OAuth czy OpenID Connect); okazuje się, że JWS może mieć więcej niż jeden podpis, JWT mogą być zagnieżdżone w sobie... a to tylko parę przykładów dalszych komplikacji. Kompaktowe podsumowanie tutaj: <https://goo.gl/uv4Q9s>.

KOLEJNY PRZYKŁAD JWT I PIERWSZE PROBLEMY BEZPIECZEŃSTWA

Przejdźmy do sedna tekstu, czyli tematyki bezpieczeństwa JWT. W tym celu przywołam tekst rozwijający zalety JSON Web Tokenów, jako element zastępujący klucz API (<https://auth0.com/blog/using-json-web-tokens-as-api-keys/>). Na marginesie warto zaznaczyć, że JWT nie musi zawsze występować w połączeniu z API, jed-



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

Patroni medialni: sekurak.pl



rozwal.to



nak w praktyce to właśnie tam można go często znaleźć. Wracając do przykładu, tego typu JWT może wyglądać tak:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOitxNDE20TI5MDYxIiwianRpIjoiODAyMDU3ZmY5YjViNGViN2ZiYjg4NTZiNmViMmNjNWIIiLCJzY29wZXMiOnsidXNlcnMiOnsiYWN0aW9ucyI6WyJyZWFKIiwiY3JlYXR1Ii19LCJ1c2Vyc19hcHBfbWV0YWRhGE1OnsiYWN0aW9ucyI6WyJyZWFKIiwiY3JlYXR1Ii19F0Xo.g1l8YBKPLq6ZLkCPLoghaBZG_eyJFLREyLQYx012BG3E
```

Po rozkodowaniu funkcją BASE64URL-decode otrzymujemy:

nagłówek:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

payload:

```
{  
  "iat": "1416929061",  
  "jti": "802057ff9b5b4eb7fbb8856b6eb2cc5b",  
  "scopes": {  
    "users": {  
      "actions": [  
        "read",  
        "create"  
      ]  
    },  
    "users_app_metadata": {  
      "actions": [  
        "read",  
        "create"  
      ]  
    }  
  }  
}
```

podpis:

(binarna zawartość)

Jak widać, dzięki takiemu „kluczowi API” (jego główna zawartość jest w payloadzie) możemy zrealizować uwierzytelnienie (mam prawo do komunikowania się z API) oraz autoryzację (widać tutaj np. akcje możliwe do wykonania przez właściciela klucza).

Od strony bezpieczeństwa można zauważać na początek co najmniej dwa potencjalne problemy.

Pierwszy z nich to brak poufności – byliśmy w stanie łatwo rozkodować payload (i nagłówek). Czasem to nie jest problem (a wręcz zaleta), jednak kiedy wymagamy poufności samych danych przesyłanych w tokenie – jest na to sposób: szyfrowanie tokena (JWE - JSON Web Encryption).

Drugi kłopot to potencjalna możliwość nieautoryzowanego dodania kolejnej akcji przez użytkownika – np. *delete* i tym samym ominięcie autoryzacji. W tym przypadku rozwiązaniem jest standardowa możliwość podpisywania tokenów – we wcześniejszym przykładzie widzieliśmy przecież „podpis”.

Wskazany w nagłówku algorytm HS256 to standardowy HMAC-SHA256 – mechanizm zapewniający integralność całej wiadomości (dzięki niemu użytkownik nie może zmienić payloadu; albo inaczej – może, jednak akceptujący token – API – wykryje to na poziomie weryfikacji podpisu, który nie będzie się zgadzać z payloadem).

Aby skonfigurować HS256, potrzebne jest wygenerowanie klucza (ciągu znaków) i umieszczenie go w konfiguracji naszego API.

Dla jasności przekazu można w dużym uproszczeniu wyobrazić sobie podpis jako:

```
SHA-256(nagłówek || payload || klucz) - gdzie || oznacza konkatenację.
```

Formalnie rzecz ujmując, jest to nieco bardziej skomplikowane: używamy algorytmu HMAC-SHA256, do którego przekazujemy klucz i wiadomość równą wynikowi:

```
BASE64URL(UTF8(JWS Protected Header)) || '.' || BASE64URL(JWS Payload)
```

Wracając do uproszczonej definicji – jeśli ktoś podmieni payload, nie jest w stanie wygenerować nowego podpisu (bo jest do niego potrzebny klucz, który znajduje się tylko w konfiguracji API).

Podsumowując: JWT wygląda na zdecydowanie bardziej elastyczny element niż klucze API – można łatwo przekazywać dowolne dane, można zapewnić ich integralność, a także – w razie potrzeby – poufność. Dodatkowo wszystkie informacje (poza kluczem), które służą do weryfikacji kogoś, kto przedstawia dany token, mogą być w samym tokenie (otrzymujemy bezstanowość i znaczną redukcję obciążenia bazy danych). Nie ma róży bez kolców.

KOLEC PIERWSZY: NADMIERNA KOMPLIKACJA

Jednym z zasadniczych problemów jest fakt, że JWT – biorąc pod uwagę powiązane specyfikacje – jest jednak bardzo skomplikowanym mechanizmem. JWT / JWS / JWE / JWK, mnogość algorytmów kryptograficznych, dwa różne sposoby kodowania (ang. *serialization*), możliwość kompresji, możliwości więcej niż jednego podpisu, szyfrowanie do wielu odbiorców – to tylko kilka przykładów. Komplikacja na pewno nie jest przyjacielem bezpieczeństwa i powoduje możliwość wielu pomyłek w trakcie implementacji. Widać to choćby w kolejnym problemie.

KOLEC DRUGI: NONE

Jak już wspomniałem, często zakłada się, że „właściwy” JWT to właśnie JWT z podpisem (JWS), choć zgodnie z formalną specyfikacją JWT – nie musi być on obecny. Stosowny dokument RFC (<https://tools.ietf.org/html/rfc7519>) wskazuje tzw. „unsecured JWT” – czyli bez sygnatury. Jak taki niepodpisany token JWT wygląda? W nagłówku mamy:

```
{  
  "alg": "none",  
  "typ": "JWT"  
}
```

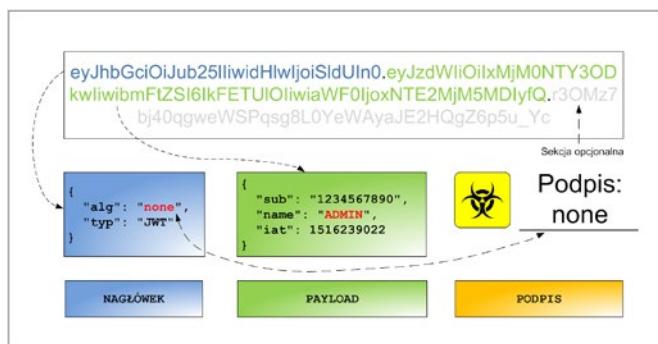
w payloadzie – to, co zazwyczaj. Sekcja podpisu jest pusta (a przy najmniej ignorowana przez przetwarzającego taki token).

Co ciekawe, powyżej wskazany algorytm none to jeden z dwóch, które wg stosownego RFC muszą być zaimplementowane:

Of the signature and MAC algorithms specified in JSON Web Algorithms [JWA], only HMAC SHA-256 ("HS256") and "none" MUST be implemented by conforming JWT implementations.

Co to daje atakującemu? Otóż dostajemy JWT (z podpisem), chcemy go zmienić (np. dodać sobie nowe uprawnienie) – ustawiamy więc w nagłówku `{"alg": "none"}` i dowolnie zmieniamy payload. Wysyłamy całość do API z podpisem lub bez. Czy serwer powinien przyjąć taki token? Teoretycznie tak, ale byłoby to przecież za przeczenie całej idei podpisów. Takie sytuacje rzeczywiście jednak miały (jeszcze mają?) miejsce w wielu bibliotekach obsługujących JWT: <https://goo.gl/nAmBW9>.

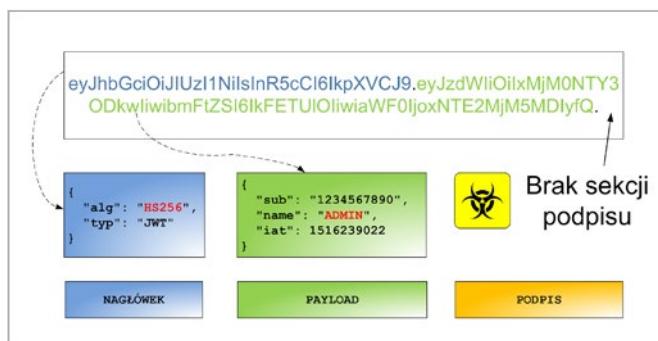
Przykład na Rysunku 2.



Rysunek 2. Algorytm none

Przy okazji warto wspomnieć o innym problemie: co się stanie, jeśli w nagłówku będziemy mieli dowolny algorytm podpisu (np. HS256 czy HS512), ale z tokena usuniemy cały podpis?

Jak widać tutaj: <https://goo.gl/dXWUxR> (CVE-2018-1000125) czasem taki token... będzie zweryfikowany poprawnie! Można powiedzieć nieco z przekąsem – to nowoczesna odmiana problemu „none” (patrz Rysunek 3):



Rysunek 3. Brak sekcji podpisu, mimo wskazania algorytmu w nagłówku

Czasem mamy do czynienia jeszcze z innym problemem – jeśli atakujący nie umie wytworzyć prawidłowego podpisu, to może... będzie on umieszczony w komunikacie błędu, zwracanym do samego atakującego? Nieprawdopodobne? Warto w takim razie zobaczyć tę podatność: <https://github.com/jwt-dotnet/jwt/issues/61>.

Critical Security Fix Required: You disclose the correct signature with each SignatureVerificationException.

Czyli jeśli ktoś zmienił payload i wysłał taki token do serwera, serwer nas o tym grzecznie informował, podając... poprawny token, który pasuje do naszego payloadu:

Invalid signature. Expected S2LYALD0A20rNSqpJDWljqFxmxEuwArW8iE9HQRT5KJM= got 6A7DHMy6EV7eensz4xyVq+i0QJmn7DgMqm406XGI7Tk=

Żeby całość zadziałała, serwer musiał być skonfigurowany w trybie wyświetlania wyjątków do użytkownika, ale wcale nie jest to rzadka (choć niepoprawna) konfiguracja.

KOLEJ TRZECI: ŁAMANIE HASŁA DO HMAC

Wróćmy do podpisu, a dokładniej algorytmu HS256 (HMAC-SHA256). W jaki sposób następuje podpis? W pewnym uproszczeniu: każde miejsce, na którym działa API (np. osobne serwery), które chce mieć możliwość podpisu/weryfikacji podpisu, musi mieć ustawiony klucz¹ – np. słowo „sekret123”.

Cały podpis to HMAC-SHA256 – który w pewnym uproszczeniu sprowadza się do podwójnego wykonania SHA256 na nagłówku sklejonym z payloadem i wyżej wspomnianym kluczem (pełen opis HMAC można zobaczyć tutaj: <https://tools.ietf.org/html/rfc2104>).

Aby wytworzyć podpis dla zmienionego payloadu, należałoby znać klucz (ale jest on dostępny tylko w konfiguracji API). A może jednak byłaby możliwość jego odzyskania?

Standardowym atakiem jest tutaj użycie dowolnego tokena wygenerowanego przez API i próba jego łamania – czyli klasyczny atak typu bruteforce/słownikowy/mieszany itp.

Jedna iteracja łamania wymaga wyliczenia dwóch hashy SHA256 (tak działa HMAC), a istnieją również gotowe narzędzia automatyzujące całą operację – jak choćby hashcat (<https://hashcat.net/hashcat/>) – implementujący łamanie klucza do JWT na kartach graficznych. Posiadając kilka bardzo dobrych kart graficznych, można uzyskać prędkość powyżej miliarda sprawdeń na sekundę. Co więcej – całą operację można wykonywać zupełnie bez interakcji z API (wystarczy uzyskać jeden dowolny token z podpisem).

Przykładowa sesja łamania JWT wygląda tak:

```
Session.....: hashcat
Status.....: Running
Hash.Type.....: JWT (JSON Web Token)
Hash.Target....: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWliOilxMjM0NTY3ODkwliwibmFtZSI6IkFETUIOliwiaWF0IjoxNTE2MjM5MDlyfQ.
Time.Started....: Fri Mar 30 16:01:35 2018 (1 min, 30 secs)
Time.Estimated...: Fri Mar 30 16:12:55 2018 (9 mins, 50 secs)
Guess.Mask.....: ?1?2?2?2?2?2 [7]
Guess.Charset....: -1 ?1?d?u, -2 ?1?d, -3 ?1?d*!$@_, -4
Undefined
Guess.Queue.....: 7/15 (46.67%)
Speed.Dev.#1....: 198.0 MH/s (9.68ms) @ Accel:32 Loops:8
Thr:512 Vec:1
Recovered.....: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.....: 17964072960/134960504832 (13.31%)
Rejected.....: 0/17964072960 (0.00%)
Restore.Point....: 0/1679616 (0.00%)
Candidates.#1....: U7veran -> a2vbj14
HWMon.Dev.#1....: Temp: 75c Fan: 48% Util: 98% Core:1873MHz
Mem:3802MHz Bus:16
```

Na jednej karcie Nvidia GTX 1070 uzyskujemy prędkość łamania około 200 milionów hashy na sekundę. Jeśli hashcatowi uda się złamać klucz, otrzymamy taki wynik, gdzie secretey to właśnie poszukiwany klucz:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWliOilxMjM0NTY3ODkwliwibmFtZSI6IkFETUIOliwiaWF0IjoxNTE2MjM5MDlyfQ.W7TG0x5Ed1GLN6eOPhTNHeL8TfwBFRSQnqleolhXG0:secretey

Jak widać, zbyt słabe hasło ustawione w konfiguracji może prowadzić do jego odzyskania i w konsekwencji możliwości generowania dowolnych (przechodzących poprawnie weryfikację

1. Terminy: klucz i hasło będą w kontekście HMAC używane zamiennie – choć w ogólności (np. w algorytmie AES) nie muszą być tożsame.

podpisu) tokenów przez atakującego (odzyskany klucz może służyć do podpisu).

Jak bardzo złożonego klucza mamy więc użyć? Odpowiedź jest zakopana w stosownym RFC (JSON Web Algorithms): <https://tools.ietf.org/html/rfc7518>:

A key of the same size as the hash output (for instance, 256 bits for "HS256") or larger MUST be used with this algorithm. (This requirement is based on Section 5.3.4 (Security Effect of the HMAC Key) of NIST SP 800-117 [NIST.800-107], which states that the effective security strength is the minimum of the security strength of the key and two times the size of the internal hash value.).

Przy okazji warto jeszcze raz podkreślić, że łamanie hasła do JWT w ten sposób jest zupełnie niezauważalne dla części serwerowej (weryfikującej podpis) – bardzo trudne więc może być wykrycie tego faktu (token „prawdziwy” od tokena „sfałszowanego” w zasadzie nie będzie się różnił – podpisy są poprawne na obu!).

KOLEC CZWARTY: GDZIE ALGORYTMÓW SZEŚĆ, TAM NIE MA BEZPIECZEŃSTWA

Większość problemów bezpieczeństwa z JWT to problemy implementacji (skomplikowanego) standardu. Zobaczmy przykład.

Jaki możemy wybrać algorytm podpisu w JWS? Do tej pory wspominałem o HMAC (i to tylko z funkcją SHA256), ale to nie jedyna opcja. Przegląd po różnych wariantach podpisu można zobaczyć np. tutaj: <https://goo.gl/A3LKcZ>.

Częstą opcją jest użycie algorytmu asymetrycznego – RSA. W tym przypadku zobaczymy w nagłówku „alg”: „RS512” lub „alg”: „RS256”.

Przypomnijmy – do podpisu w RSA służy klucz prywatny, a skojarzony z nim klucz publiczny może weryfikować podpis. Zamiast więc symetrycznego klucza np. z algorytmu HS256 generujemy parę kluczy RSA.

Swoją drogą, niektórym podejrzane może się wydać oznaczenie RS512 czy RS256 – czyżby mamy tutaj wymóg stosowania 512 czy 256 bitowych kluczy RSA? Przecież tego typu klucze są obecnie łatmalne minimalnym kosztem i niewielkim nakładem czasu (porównaj np.: <https://goo.gl/vjLf4s> czy <https://goo.gl/HjGWZw>). Obecnie nawet klucz 1024-bitowy RSA nie jest uznawany za bezpieczny.

Na szczęście w powyższym oznaczeniu chodzi o funkcję SHA wykorzystywaną w kooperacji z RSA w celu realizacji do podpisu. RS512 oznacza więc RSA plus funkcja SHA512. Ale co z kluczem RSA? Tutaj o jego długości decyduje osoba go generująca, co stanowi kolejny potencjalny problem (co więcej, na różnych forach można znaleźć konkretne polecenia wykorzystujące openssl i generujące klucze 1024-bitowe).

Przy okazji warto też pamiętać, że wykorzystanie RSA będzie miało wpływ na wydajność całego systemu (weryfikacja podpisu), czyli w tym przypadku wybieramy wariant wolniejszy niż HS256.

Wracając do sedna, przy okazji algorytmu RSA mamy jeszcze co najmniej jeden ciekawy problem bezpieczeństwa. Jak pisalem wcześniej, klucz publiczny jest używany do weryfikacji podpisu, zazwyczaj będzie on więc ustawiony w konfiguracji API jako **klucz_weryfikujący**.

Tutaj warto zwrócić uwagę, że dla HMAC mamy tylko jeden klucz symetryczny realizujący podpis i weryfikację.

Co może teraz zrobić atakujący, aby spróbować **wytworzyć dowolny, prawidłowo podpisany token**?

1. Zdobyć klucz publiczny (sama jego nazwa wskazuje, że... może być publiczny). Czasem jest on przesyłany w samym JWT.
2. Wysłać przygotowany przez siebie token – uwaga – z ustawionym w nagłówku algorymem HS256 (czyli HMAC, nie RSA) i podpisać token za pomocą klucza publicznego RSA. Tak, to nie pomyłka – stosujemy klucz publiczny RSA (który podajemy w formie ciągu znaków) jako klucz symetryczny do HMAC.
3. Serwer otrzymuje token, sprawdza, jakiego algorytmu użyto do podpisu (HS256). **klucz_weryfikujący** został ustawiony w konfiguracji jako klucz publiczny RSA, więc...
4. Podpis się zgadza (bo użyto dokładnie tego samego klucza do weryfikacji, co do wytworzenia podpisu, a algorytm podpisu został ustawiony przez atakującego na HS256).

W czym tkwił tutaj problem? W tym, że realnie daliśmy możliwość podania algorytmu podpisu przez użytkownika, choć w zamierzeniach mieliśmy weryfikować podpis tokena tylko za pomocą RSA. Zatem albo wymusząmy tylko jeden wybrany algorytm podpisu (nie dajemy możliwości jego zmiany przez zmianę tokenu), albo zapewnijmy osobne sposoby weryfikacji (i klucze!) dla każdego algorytmu podpisu, który obsługujemy.

KOLEC PIĄTY: CHOĆ MOŻE NALEŻAŁOBY MU SIĘ PIERWSZENSTWO

Jak się okazuje, z powodu błędu w bibliotece obsługującej JWT można czasem... po prostu podać w payloadzie tokena własny klucz, który następnie zostanie użyty przez API do weryfikacji tego tokena! Brzmi nieprawdopodobnie? W takim razie warto zobaczyć szczegóły podatności CVE-2018-0114 w bibliotece node-jose: <https://goo.gl/k3F9Vm>.

Zacytuje oryginalny opis niemal w całości:

The vulnerability is due to node-jose following the JSON Web Signature (JWS) standard for JSON Web Tokens (JWTs). This standard specifies that a JSON Web Key (JWK) representing a public key can be embedded within the header of a JWS. This public key is then trusted for verification. An attacker could exploit this by forging valid JWS objects by removing the original signature, adding a new public key to the header, and then signing the object using the (attacker-owned) private key associated with the public key embedded in that JWS header.

Jak widać, jest tu wskazanie na problem bezpieczeństwa w samym RFC, który dotyczy podpisu JWT – klucz opcjonalnie można przekazać w formie struktury JWK (JSON Web Key) w nagłówku JWT. Generujemy więc swoją parę kluczy RSA. Publiczny dołączamy do tokena w formie JWK, a prywatnym podpisujemy token. API weryfikuje podpis... dostarczonym kluczem publicznym. Problem nie jest nowy, wcześniej (2016 rok) wykryto podobną podatność w bibliotece Go-jose (<https://goo.gl/VdtoQD>):

RFC 7515, <https://goo.gl/qWj4wV> "jwk" (JSON Web Key) Header Parameter allows the signature to include the public key that corresponds to the key used to digitally sign the JWS. This is a really dangerous option.

KOLEC SZÓSTY: CZY SZYFROWANIE JWT MOŻE W OGÓLE DZIAŁAĆ?

Co z szyfrowaniem (JSON Web Encryption) – tutaj do wyboru mamy kilka opcji algorytmów (szyfrowanie samej wiadomości, szyfrowanie klucza symetrycznego użytego do szyfrowania wiadomości). I ponownie – tutaj istnieją niezbyt pozytywnie nastrajające badania, a mianowicie kilka implementacji JWE umożliwiało odzyskanie klucza prywatnego przez atakującego: <https://blogs.adobe.com/security/2017/03/critical-vulnerability-uncovered-in-json-encryption.html>.

Dokładniej problem istniał w implementacji algorytmu ECDH-ES (który swoją drogą ma status „Recommended+” w stosownym dokumencie RFC: <https://tools.ietf.org/html/rfc7518>).

Może jest to wyjątek? Zobaczmy na algorytm AES z trybem GCM, ale tutaj bywa średnio: <https://goo.gl/tD5r2A>.

GCM is fragile but its implementations were rarely checked.

Do wyboru mamy też choćby algorytm RSA with PKCS1v1.5 padding. Co jest z nim nie tak? Problemy są znane od 1998 roku: <ftp://ftp.rsa.com/pub/pdfs/bulletn7.pdf>. A niektórzy (por.: <https://goo.gl/1Wff7d>) podsumowują to tak:

PKCS#1v1.5 is awesome — if you're teaching a class on how to attack cryptographic protocols.

Więcej szczegółów odnośnie możliwych algorytmów i ewentualnych problemów można zobaczyć np. tutaj:

» <https://paragonie.com/blog/2017/03/jwt-json-web-tokens-is-bad-standard-that-everyone-should-avoid>

- » <https://paragonie.com/blog/2016/12/everything-you-know-about-public-key-encryption-in-php-is-wrong>
- » <https://paragonie.com/blog/2018/04/protecting-rsa-based-protocols-against-adaptive-chosen-ciphertext-attacks>

Czy zawsze będziemy skazani na porażkę, stosując JWE? Oczywiście nie – warto jednak zweryfikować, czy używamy odpowiednio bezpiecznego algorytmu szyfrującego (i jego bezpiecznej implementacji). To ostatnie może być trudne, sprawdźmy więc przynajmniej, czy w używanej przez nas bibliotece do obsługi JWE nie wykryto istotnych podatności w tym obszarze.

KOLEC SIÓDMY: DEKODOWANIE/WERYFIKACJA – CO ZA RÓŻNICA?

Zaczynało się prosto, ale możemy czuć się przytłoczeni mnogością opcji. Chcemy w końcu tylko po stronie API „odekodować” token i użyć zawartych w nim informacji. Pamiętajmy jednak, że „odekodować” nie oznacza zawsze tego samego co „zweryfikować” – niby oczywiste, ale różne biblioteki mogą dostarczać różnych funkcji do dekodowania i/lub weryfikacji tokenów. Przykład tego typu pytania czy wątpliwości można znaleźć tutaj: <https://github.com/auth0/jwt-decode/issues/4>.

W skrócie – jeśli użyję tylko funkcji typu decode() w API, to czasem wykonam tylko dekodowanie payloadu (ew. też nagłówka z BASE64URL, bez weryfikacji).

Weryfikacja może być osobną funkcją udostępniania przez bibliotekę, choć może być wbudowana w funkcję typu decode().

Czasem to wręcz użytkownicy proszą o taką opcję – w zacytowanym poniżej przypadku o przeciążenie metody decode(), tak żeby mogła również akceptować sam token (bez klucza):

reklama



devstyle.pl
ŚWIAT OKIEM PROGRAMISTY

I have seen a issue in the framework to get the payload of an JWT. To get a payload of an JWT without validation we don't need a key/secret. So in the file [jwt/src/JWT/JwtDecoder.cs](#) we missed a overload, for the method Decode that only need a token.

Jeśli teraz programista korzystający z biblioteki użyje najprostszego wywołania: `decode(token)`, to podpis nie będzie weryfikowany.

Dodatkowo w cytowanej wcześniej dyskusji jest jeszcze pytanie o możliwość weryfikowania podpisu po stronie klienckiej, co jak już wiemy w przypadku HS256 wiąże się z użyciem klucza, który zazwyczaj powinien być... sekretem. Niestety często użycie JWT sprowadza się do poniższych czynności: wybierzmy pierwsze z brzegu algorytmy, przekopujmy kawałki kodu z Internetu. Działa? Owszem – więc w czym problem?

KOLEC OSMY: PRZECHWYCENIE DOWOLNEGO TOKENA = PRZEJĘCIE DOSTĘPU DO API?

Jedną z często wskazywanych zalet JWT jest realizacja uwierzytelnienia (lub autoryzacji – zależy w jakim kontekście zostanie całość użyta) bez konieczności realizowania zapytania do bazy danych. Co więcej, możemy to zrobić równolegle na kilku niezależnych serwerach. W końcu sama zawartość tokena wystarcza do podjęcia tej decyzji. Ma to też pewną wadę – co w przypadku, gdy dostępny na wielu serwerach klucz podpisujący w jakiś sposób wycieknie? Będzie można oczywiście generować wtedy prawidłowo podpisane tokeny i zaakceptują je wszystkie maszyny, które do weryfikacji stosują odpowiedni klucz. Co może dzięki temu uzyskać atakujący? Na przykład nieautoryzowany dostęp do funkcji API czy kont innych użytkowników.

Mamy tu również potencjalnie inny problem – co w przypadku kiedy jeden wygenerowany token może być użyty w wielu różnych kontekstach? Na przykład generujemy prawidłowy token z zawartością `{ "login" = "manager" }` – i w jednej funkcji API daje on możliwość edycji pewnych danych; ale jednocześnie zapomnieliśmy że zupełnie inną funkcją otrzymującą ten sam token daje możliwość pełnego dostępu!

W tym przypadku istnieje możliwość użycia pewnych parametrów definiowanych przez samą specyfikację, np.: `iss` (issuer) oraz `aud` (audience). Dzięki nim można generować tokeny, które mogą być następnie przyjęte tylko przez określonych naszych odbiorców.

Na przykład:

```
{  
  "iss" = "my_api ",  
  "login" = "manager ",  
  "aud" = "store_api "  
}
```

Można tu wskazać jeszcze jeden problem – pewne zarezerwowane słowa kluczowe w JWT (*Registered Claim Names*) – np. `iss`/`aud`/`iat`/`jti` – mogą być umieszczone przy zupełnie dowolnych elementach definiowanych przez użytkownika JWT – np. `login` z naszego przykładu. To rodzi kolejne zamieszanie, które wskaże w kolejnym problemie.

KOLEC DZIEWIĄTY: REPLAY JWT

Co w przypadku kiedy konkretny token powinien być użyty tylko raz? Tutaj wyobraźmy sobie scenariusz, gdy użytkownik zapisuje

wygenerowany token umożliwiający wykonanie metody `DELETE` w naszym API. Następnie, np. po roku – kiedy teoretycznie już nie ma stosownych uprawnień – próbuje go użyć ponownie (tzw. atak typu *replay*).

Sposobem na to może być użycie pól: `jti` oraz `exp`. `Jti` (JWT ID) to identyfikator tokena, który musi być unikalny, a `exp` – to określenie daty ważności tokena. Połączenie obu pól da nam odpowiednio krótką ważność tokena oraz jego unikalność.

Warto jednak zwrócić uwagę na to, czy mamy poprawną implementację obsługi tych pól – tutaj polecam spojrzeć na błąd, gdzie wartość `exp` w ogóle nie była brana pod uwagę (<https://github.com/jwt-dotnet/jwt/issues/134>).

[JWT not throwing ExpiredTokenException in .NetCore](#)

Tworzący bibliotekę użyli do sprawdzania wartości pola `Expire` (niezgodnie ze specyfikacją JWT), a błąd po zgłoszeniu został skorygowany.

KOLEC DZIESIĄTY: ATAKI CZASOWE NA PODPIS

Jeśli podpis z JWS sprawdzany jest **bajt po bajcie** z podpisem, który jest prawidłowy (wygenerowanym po stronie akceptującej JWS), oraz sprawdzanie to **kończy działanie na pierwszym niezgodnym bajcie**, możemy być podatni na atak czasowy.

Zauważmy, że w takim przypadku im więcej bajtów zgodnych, tym więcej potrzebnych jest porównań i stąd dłuższy czas potrzebny na odpowiedź.

Można więc obserwować czasy odpowiedzi, generując kolejne podpisy – rozpoczynając od pierwszego bajtu podpisu, później przejść do drugiego itp. Dokładny opis takiego ataku można zobaczyć tutaj: <https://goo.gl/wemmbn>.

Według cytowanego opracowania, przy dużej ilości generowanego ruchu (aż 55 tysięcy requestów na sekundę) podpis dowolnej wiadomości można by uzyskać w 22 godziny (warunki laboratoryjne). Przy mniejszym ruchu oczywiście będziemy potrzebowali więcej czasu (kilka/kilkanaście dni) – ale efekt może być porażający (możemy wygenerować dowolny JWT i przygotować podpis, który będzie zweryfikowany jako poprawny).

Czy atak jest rzeczywiście możliwy do zrealizowania w praktyce? Jak można sobie wyobrazić, zmiany w czasie odpowiedzi są minimalne, ale można je jednak próbować mierzyć. W tym miejscu warto też przypomnieć nieco starszy tekst o atakach czasowych: <https://www.cs.rice.edu/~dwallach/pub/crosby-timing2009.pdf>.

Udało się tutaj osiągnąć następujące pomiary:

Our work analyzes the limits of attacks based on accurately measuring network response times and jitter over a local network and across the Internet. We present the design of filters to significantly reduce the effects of jitter, allowing an attacker to measure events with 15-100μs accuracy across the Internet, and as good as 100ns over a local network.

Z kolei przykład zgłoszenia konkretnej tego typu podatności można znaleźć np. w tym miejscu:

» <https://github.com/jasongoodwin/authentikat-jwt/issues/12>

lub tutaj:

» <https://github.com/emarref/jwt/pull/20>

KOLEJ JEDENASTY: MNOGOŚĆ BIBLIOTEK

Jako jeden z pierwszych problemów JWT wymieniłem mnogość dostępnych opcji i różnorakich algorytmów. Na całość nakłada się również duża liczba często niekompletnych implementacji JWT, pisanych, delikatnie mówiąc, na kolanie. Niełatwki i czasem generujący problemy bezpieczeństwa standard, wymagane użycie skomplikowanych algorytmów kryptograficznych przez osoby często nie posiadającej głębszej wiedzy o samej kryptografii... zgodnie z zasadą *garbage in, garbage out*: trudno spodziewać się tutaj super bezpiecznych bibliotek.

Część błędów w bibliotekach wymieniłem już wcześniej. Inne przykłady tutaj:

- » <https://www.cvedetails.com/cve/CVE-2017-12973/>
- » <https://www.cvedetails.com/cve/CVE-2017-10862/>
- » <https://pivotal.io/security/cve-2017-2773>
- » <https://github.com/auth0/node-jsonwebtoken/issues/212>
- » <https://goo.gl/tkhztt>

Przy okazji warto wspomnieć jeszcze o ogólnych problemach bezpieczeństwa dotyczących wykorzystywanych bibliotek. Po pierwsze, warto zastanowić się, czy używam biblioteki bez znanych publicznie podatności? A może używa ona podatnych zależności? Czy mam opracowane monitorowanie wykorzystanej biblioteki – na wypadek, że ktoś zlokalizuje w przyszłości istotną podatność?

ALTERNatywa DO JWT?

Patrząc na wiele problemów bezpieczeństwa, które wskazałem w JWT, można się zastanawiać, czy mamy jakąś sprawdzoną alternatywę? Na obecną chwilę odpowiedź jest raczej negatywna. Jednym z nowych pomysłów jest PASETO: <https://paseto.io/>.

PASETO is everything you love about JOSE (JWT, JWE, JWS) without any of the many design deficits that plague the JOSE standards.

Czyli w wolnym tłumaczeniu PASETO ma być bezpieczną wersją JWT. Czy rzeczywiście spełni obietnice? Ciężko obecnie powiedzieć – jest to bardzo młody projekt, cały czas znajdujący się w fazie rozwoju (stan na początek 2018 roku). Więcej informacji można znaleźć tutaj: <https://goo.gl/2S7wBF>.

Na temat motywacji do powstania projektu przeczytamy natomiast tutaj: <https://goo.gl/KZQU42>.

CZY JWT MOŻE BYĆ BEZPIECZNE?

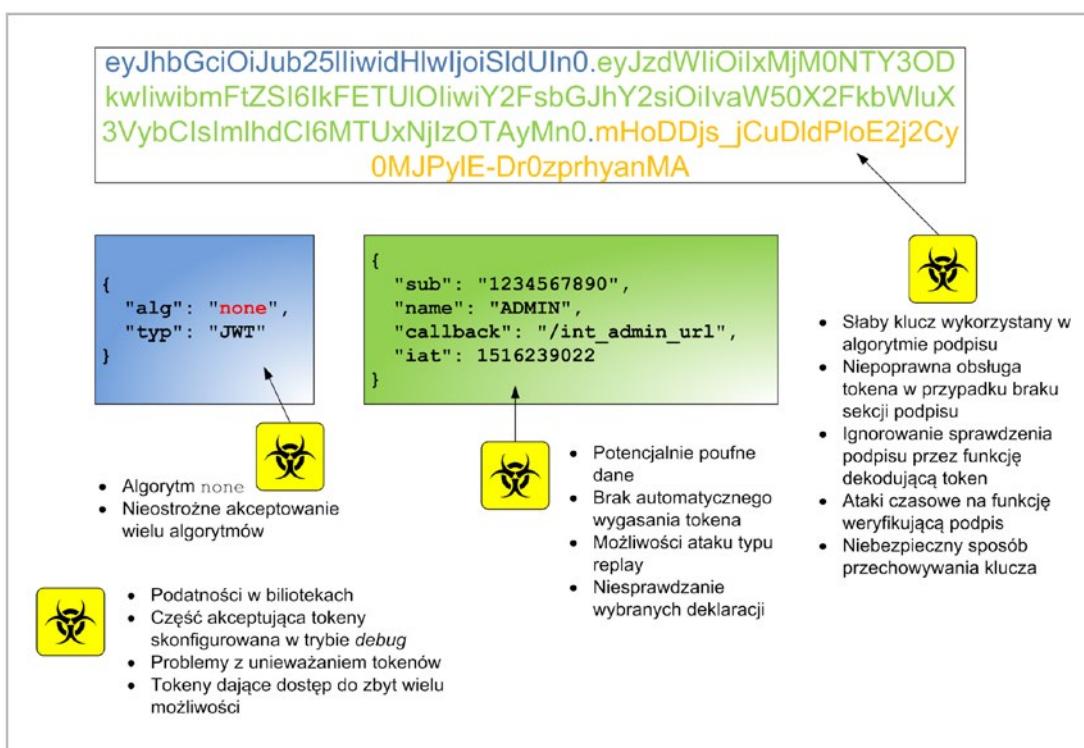
Zdania w środowisku osób zajmujących się bezpieczeństwem są pozielone. Niektórzy kategorycznie odradzają użycia JWT, inni wskazują jedynie na słabo przygotowane implementacje, pozostali z kolei opisują dokładnie same mechanizmy JWT, pozostawiając decyzję użytkownikowi. Na całość nakłada się fakt, że JWT jest jednak bardzo popularnym mechanizmem, do którego nie ma ustandaryzowanej alternatywy, która dodatkowo udowodniła swoje bezpieczeństwo.

Wskazane przeze mnie wcześniej problemy bezpieczeństwa można umieścić w kilku kategoriach:

1. Problemy samej specyfikacji JWT (np. algorytm none).
2. Błędy implementacji bibliotek, w tym błędy implementacji algorytmów kryptograficznych (prawdopodobnie jest to najliczniejsza grupa problemów).
3. Niepoprawne użycie biblioteki.

Większość częstych problemów zaprezentowano na Rysunku 4.

Podsumujmy jeszcze całość praktycznymi radami mogącymi zwiększyć bezpieczeństwo JWT. Część zaleceń może zawierać skróty myślowe, które są jednak wyjaśnione wcześniej w tekście, dodatkowo rozszerzone linkami do materiałów źródłowych.



Rysunek 4. Zestawienie problemów bezpieczeństwa w JWT (JWS)

Na początek

1. Zrozum to, z czego chcesz skorzystać: zastanów się, czy potrzebujesz JWS czy JWE, wybierz stosowne algorytmy, zrozum zasadę ich działania (przynajmniej na ogólnym poziomie – np. HMAC, klucz publiczny, prywatny). Sprawdź dokładnie, jakie możliwości oferuje biblioteka, którą wybrałeś do obsługi JWT. Może istnieje już gotowy, prostszy mechanizm, który możesz wykorzystać?

Klucze

2. Używaj odpowiednio złożonych kluczy symetrycznych/asymetrycznych.
3. Miej przygotowany scenariusz na wypadek kompromitacji (ujawnienia) jednego z kluczy.
4. Przechowuj w odpowiednio bezpieczny sposób klucze (np. nie zapisuj ich na trwałe w kodzie źródłowym).
5. Nie pozwalaj na ustawienie algorytmu podpisu wysyłającemu token (najlepiej wymuś konkretny algorytm podpisu po stronie serwerowej).

Podpis

6. Sprawdź, czy Twоя implementacja nie akceptuje algorytmu podpisu none.
7. Sprawdź, czy Twоя implementacja nie akceptuje pustego podpisu (czyli go nie sprawdza).
8. W przypadku użycia JWE sprawdź, czy używasz bezpiecznych algorytmów oraz bezpiecznej implementacji tych algorytmów.
9. Rozróżnij funkcje `verify()` od `decode()`. Innymi słowy – sprawdź, czy na pewno weryfikujesz podpis.

Ogólne zasady

10. Sprawdź, czy wygenerowany w jednym miejscu token nie może być użyty w innym w celu uzyskania nieuprawnionego dostępu.

11. Sprawdź, czy został wyłączony tryb debug i czy nie można go włączyć prostym trikiem (np. `?debug=true`).
12. Unikaj przesyłania tokenów w URLu (nie jest to bezpieczne – np. tokeny są wtedy zapisywane bezpośrednio do logów webserwerów).

Payload

13. Sprawdź, czy w payloadzie JWS nie umieszczasz poufnych informacji.
14. Sprawdź, czy chronisz się przed ponownym wysłaniem (atak typu *replay*) tokena.
15. Sprawdź, czy tokeny mają odpowiednio krótką ważność (np. poprzez wykorzystanie deklaracji „exp”). Sprawdź, czy „exp” rzeczywiście jest poprawnie sprawdzane.
16. Zastanów się, czy potrzebujesz funkcji unieważnienia pojedynczych tokenów (sam standard na to nie pozwala, jednak istnieje kilka sposobów implementacji tego typu mechanizmu).

Biblioteki

17. Przeczytaj dokładnie dokumentację biblioteki.
18. Sprawdź podatności w wykorzystywanej przez siebie bibliotece (np. w serwisie: cvedetails.com czy na stronie projektu).
19. Sprawdź, czy Twoje poprzednie projekty nie korzystają z podanej biblioteki; sprawdź, czy monitorujesz nowe błędy w bibliotece (mogą one się pokazać np. po miesiącu od wdrożenia).
20. Śledź nowe podatności w bibliotekach obsługujących JWT. Był może ktoś w przeszłości znajdzie podatność w innym projekcie, która w dokładnie takiej samej formie występuje w wykorzystywanym przez Ciebie rozwiązaniu.

W ramce „W sieci” znajduje się podsumowanie interesujących zasobów pokazujących zarówno problemy z JWT, jak i dobre praktyki postępowania z tym mechanizmem.

W sieci

1. JSON Web Token Best Current Practices: <https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp-01>
2. JWT Handbook: <https://auth0.com/resources/ebooks/jwt-handbook> (załącznik: „Best Current Practices”)
3. Dyskusja o słabościach JWT: https://lobste.rs/s/r4lv76/jwt_is_bad_standard_everyone_should_avoid
4. Wybrane słabości w JWT wg OWASP: [https://www.owasp.org/index.php/JSON_Web_Token_\(JWT\)_Cheat_Sheet_for_Java](https://www.owasp.org/index.php/JSON_Web_Token_(JWT)_Cheat_Sheet_for_Java)
5. Kilka pomysłów na bezpieczne użycie JWT: <https://dev.to/neilmadden/7-best-practices-for-json-web-tokens>
6. Zbiór argumentów przeciwko używaniu JWT do tworzenia sesji: <http://cryto.net/~joepie91/blog/2016/06/13/stop-using-jwt-for-sessions/>
7. Porównanie JWT z identyfikatorami sesji oraz rady dotyczące odpowiednich zabezpieczeń: <http://by.jtl.xyz/2016/06/the-unspoken-vulnerability-of-jwts.html>



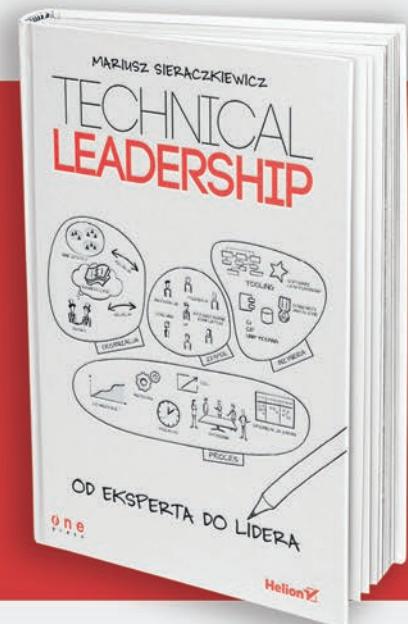
MICHAŁ SAJDAK

Konsultant d/s bezpieczeństwa IT w firmie Securitum, redaktor serwisu sekurak.pl; prowadzi szkolenia i warsztaty – w tym z bezpieczeństwa API REST.

BNS IT - SZKOLENIA OTWARTE

WARSZAWA / 04-06.07.2018
TECHNICAL LEADERSHIP™
ROLA LIDERA TECHNICZNEGO

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



WARSZAWA / 16-18.05.2018
WZORCE PROJEKTOWE
I REFAKTORYZACJA DO WZORCÓW

1. Wprowadzenie do wzorców projektowych
2. Jakość kodu źródłowego
3. Refaktoryzacja
4. Wzorce GoF

WARSZAWA / 11-13.07.2018
NOWOCZESNE
ARCHITEKTURY APLIKACJI

1. Wprowadzenie do pojęcia architektura oprogramowania
2. Domain-Driven Design
3. Micro Services
4. Ports & Adapters
5. Clean and Onion Architecture
6. Reactive Architecture
7. Serverless Architecture

P O Z O S T A Ł E T E R M I N Y :

Zbieranie wymagań i współpraca z klientem	Łódź	16-18.05.2018	2100,00 PLN
Tworzenie Microservices z użyciem Spring Boot	Warszawa	28-30.05.2018	2100,00 PLN
Wzorce projektowe i refaktoryzacja do wzorców	Łódź	11-13.06.2018	2100,00 PLN
Zbieranie wymagań i współpraca z klientem	Łódź	14-16.11.2018	2100,00 PLN
Wzorce projektowe i refaktoryzacja do wzorców	Łódź	17-19.12.2018	2100,00 PLN

CENY NETTO

Współczesne architektury aplikacji biznesowych. Warstwy i Domain-Driven Design

Ten cykl artykułów ma na celu dokonać przeglądu różnych trendów architektonicznych, które pojawiły się w ciągu ostatnich kilku lat, po to aby je uporządkować, zestawić ze sobą, wskazać główne powody zastosowania, jednocześnie układaając je w ewolucyjną ścieżkę, którą może podążać system na tle zmian architektonicznych. Przyjrzymy się klasycznej architekturze warstwowej, Domain-Driven Design, Ports and Adapters, microservices, architekturze reaktywnej i serverless.

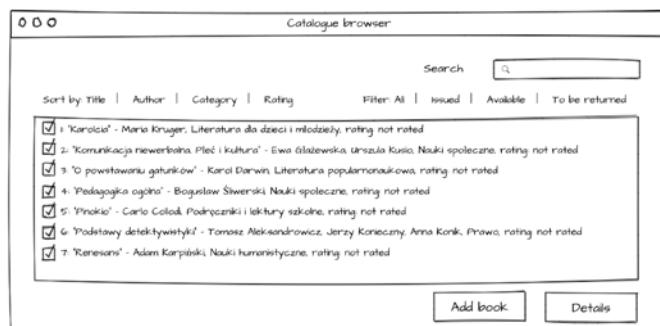
MODOM MÓWIMY: NIE

Branża informatyczna jest bardziej zmienna niż branża modowa, powinna się nowe podejścia, ale najczęściej są one kolejnym wcześnieiem znanych od dawna w świecie inżynierii oprogramowania paradymatów, ponazywanymi inaczej, bardziej świeże, z nowymi guru i pomyślnymi wdrożeniami dokonanych przez rozpoznawalne firmy. W ciągu ostatnich kilku lat można by wymienić co najmniej kilka: microservices, event-driven architecture czy programowanie reaktywne, które są niczym innym jak implementacją znanych już wcześniej koncepcji systemów rozproszonych, przetwarzania zdzieleniowego czy programowania asynchronicznego. Zjawisko to ma swoje plusy i minusy. Po pierwsze, dzięki niemu świat IT cały czas kipi energią, pojawia się coś nowego i społeczność zaczyna się tym interesować, co pobudza motywację do działania. Po drugie, najczęściej kolejne inkarnacje cechują się pogłębianiem tematu i technik, a przede wszystkim dostępnych narzędzi, dzięki którym na przykład tworzenie systemów rozproszonych przy użyciu rozwiązań z rodziny microservices jest łatwiejsze niż jeszcze kilkanaście lat temu z użyciem technologii CORBA. Po trzecie, wokół nowych trendów zazwyczaj powstaje bardzo dużo szumu, który powoduje wrażenie, że każdy powinien wdrożyć je w siebie natychmiast, bo dotychczas stosowane rozwiązania są zafafane, powodują tylko i wyłącznie problemy, które szybko doprowadzą do katastrofy. W efekcie zespoły zaczynają stosować nowe koncepcje, nie mając realnego uzasadnienia dla ich użycia, komplikując niewspółmiernie do problemu kod i architekturę.

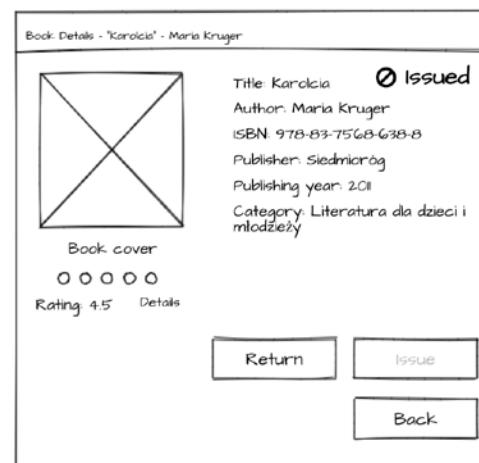
Architektura to twór ewolucyjny, co oznacza, że nie musisz rozpoczynać od podejścia, które akurat teraz jest najbardziej popularne i prawdopodobnie dość skomplikowane, a zacząć od najprostszego, a w razie potrzeby wprowadzać do niego złożoność. W tej części serii przyjrzymy się pierwszym dwóm podejściom, wcale nie najnowszym, natomiast mocno rozposzczonym w obecnych aplikacjach: klasycznej architekturze warstwowej oraz Domain-Driven Design.

ARCHITEKTURA

Każdy bardziej złożony system wymaga wysokopoziomowej struktury, aby można było go rozwijać w sposób przewidywalny i spójny, szczególnie gdy pracuje nad nim wiele osób. Architektura to właśnie ta wysokopoziomowa struktura, w ramach której następuje



Rysunek 1. Ekran przedstawiający katalog książek



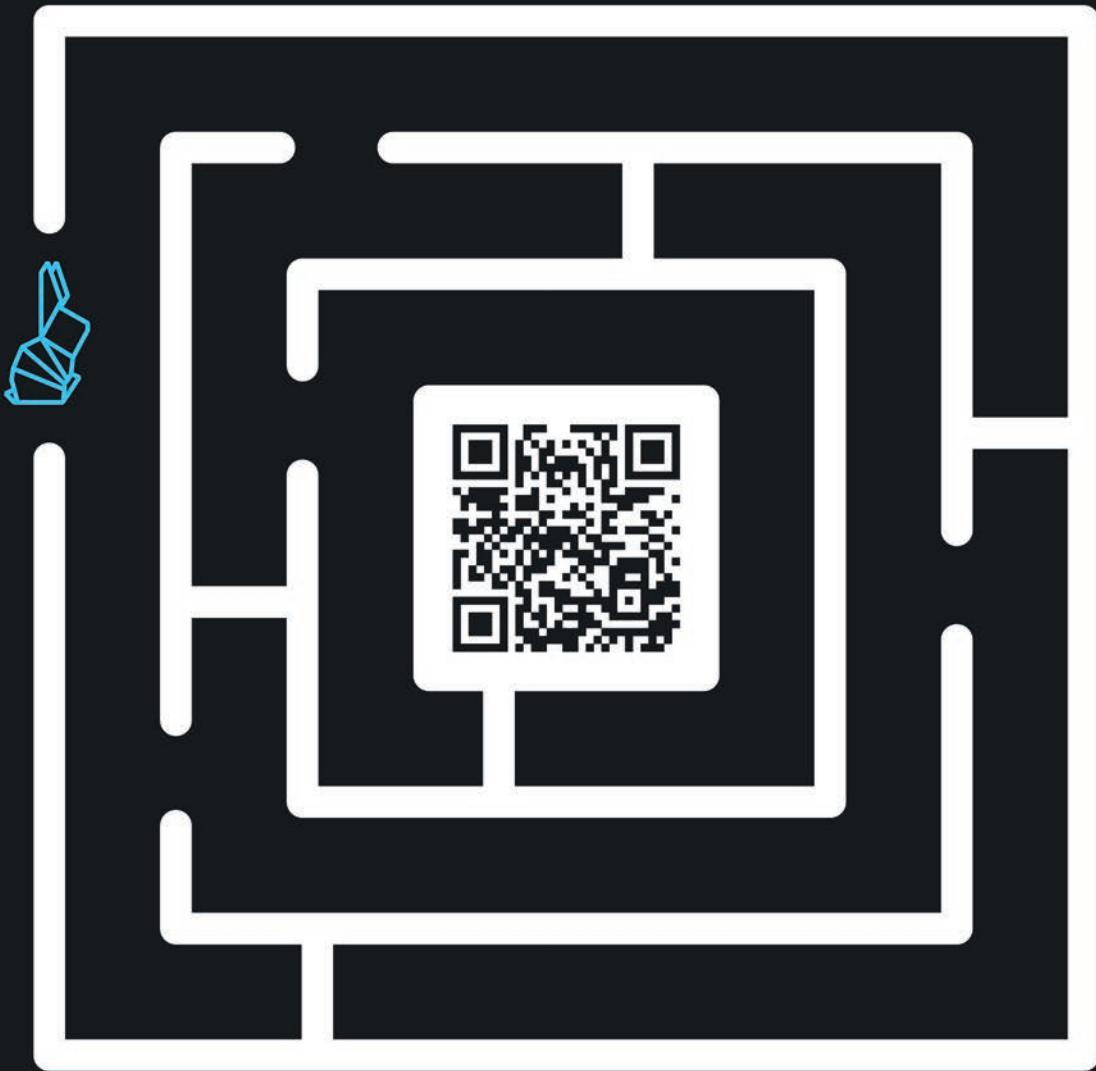
Rysunek 2. Ekran przedstawiający szczegóły związane z książką

dobór technologii i narzędzi. Często jest wypełniana blokami budującymi, z których składa się aplikacja, takimi jak kontrolery, usługi, encje, widoki, DAO, repozytoria.

Konstruując system, możemy potraktować go jako złożoną, jednolitą całość, budując jeden model danych (tzw. monolit) lub tworząc go jako połączenie wielu mniejszych podsystemów nazywanych w zależności od przyjętego podejścia: podsystemami, poddomenami lub Bounded Contextami.

Przyglądając się poszczególnym architekturom, pokażemy kluczowe wyróżniki i różnice na bazie przykładowego systemu wypożyczalni książek. Przykładowe funkcjonalności zostały wymienione poniżej.

follow the blue rabbit...



Simple?
More riddles is waiting for you!

coming soon

Jako dowolny użytkownik:

- » Mogę zobaczyć listę książek.
- » Mogę zobaczyć szczegóły odnośnie wybranej książki.
- » Mogę przefiltrować książki: wszystkie, wypożyczone, dostępne, do zwrotu.
- » Mogę posortować listę książek po: ocenie, nazwie książki.

Jako pracownik biblioteki:

- » Mogę dodać/zmodyfikować książkę.
- » Mogę wypożyczyć komuś książkę.
- » Mogę zobaczyć wszystkie wypożyczone książki posortowane po dacie oddania.

Jako wypożyczający:

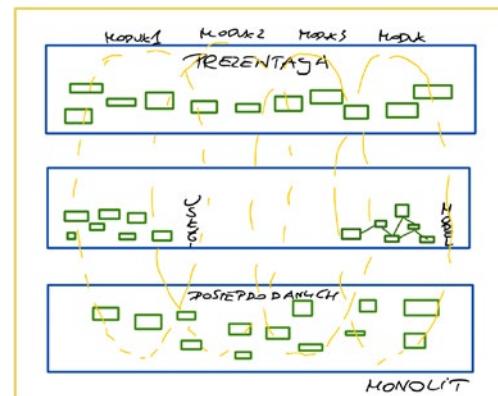
- » Mogę ocenić książkę.
- » Mogę zobaczyć historię ocen.

Na Rysunku 1 i 2 znajdziesz dwa przykładowe ekrany obejmujące powyższe funkcjonalności.

Dla uzyskania klarowności przytaczanych przykładów przyjęto, że dana książka ma tylko jeden egzemplarz, który jest wypożyczany i oceniany.

KLASYCZNA ARCHITEKTURA WARSTWOWA (MONOLIT+WARSTWY)

Jedną z najbardziej rozpowszechnionych zasad w inżynierii oprogramowania jest wydzielanie odpowiedzialności, co w przypadku architektury może przekładać się m.in. na podział warstwowy, w ramach którego wydzielamy odpowiedzialności związane z technicznymi aspektami funkcjonowania systemu. Typowym podziałem na warstwy jest wyróżnienie części prezentacyjnej (związanej często z interfejsem użytkownika), części związanej z logiką biznesową oraz części związanej z dostępem do danych. A to wszystko najczęściej w formie monolitu. Dlatego nazywam ją klasyczną, gdyż duża część systemów istniejących co najmniej kilka lat jest zbudowana w taki sposób.

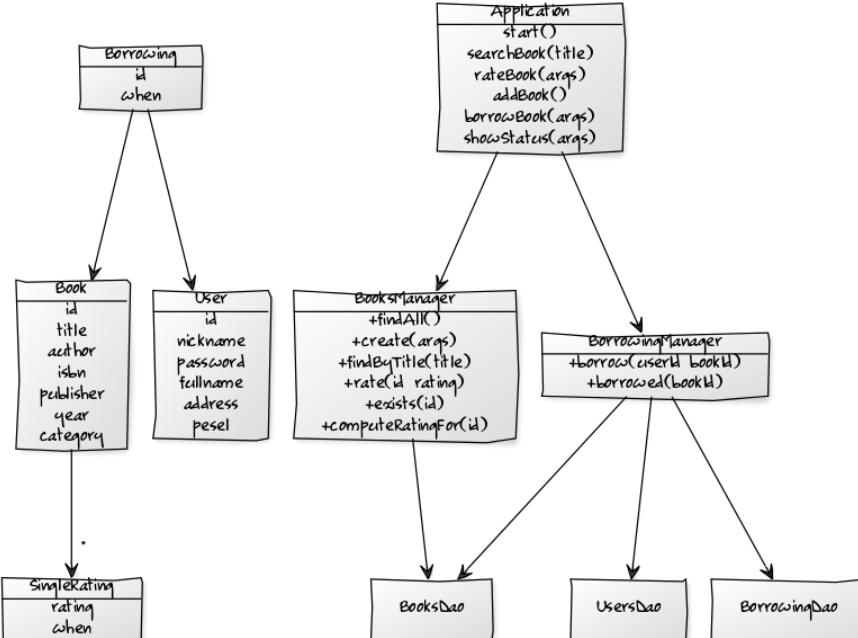


Rysunek 3. Ogólny schemat architektury warstwowej

Dodatkowo owej klasycznej architekturze najczęściej towarzyszy jeszcze tzw. anemiczny model, tzn. klasy reprezentujące dziedzinowe pojęcia w systemie przechowują głównie dane, zaś logika związana z regułami biznesowymi znajduje się w innych klasach, najczęściej w tej konwencji w klasach typu Manager lub Service. Jest to mało obiektywe podejście, gdyż dane i zachowanie są przechowywane osobno, logika biznesowa związana z tymi samymi pojęciami biznesowymi bywa rozrzucona pomiędzy wiele klas, tworząc duże ryzyko duplikacji. Poza tym implementacja reguł biznesowych staje się skomplikowana, gdyż wszystkie dane do algorytmów przychodzą z zewnątrz.

Przykładowy system do wypożyczania książek w tej konwencji mógłby zostać zaprojektowany w sposób przedstawiony na Rysunku 4.

Klasa Application reprezentuje warstwę prezentacji i udostępnia funkcjonalności końcowemu użytkownikowi. Warstwę logiki biznesowej reprezentują obiekty typu Manager, które przyjmują jako parametry typy proste oraz wewnętrznie operują na obiektach modelu. Same obiekty modelu przechowują tylko dane, nie posiadają zachowania (znaczących metod). Ostatecznie, aby w sposób trwał zapisać dane, używane są obiekty typu DAO, główny reprezentant warstwy dostępu do danych, które w tym przypadku dokonują operacji na bazie danych (głównie operacje typu CRUD: create, retrieve, update, delete).



Rysunek 4. Diagram klas fragmentu systemu do wypożyczania książek

Cechą charakterystyczną tego monolitycznego podejścia jest próba ujęcia wszystkich danych w postaci jednego spójnego modelu powiązanych ze sobą obiektów (tutaj: Borrowing, Book, User, Simple Rating). Najczęściej w praktyce tworzenie takich systemów odbywa się wg następującego schematu: zaprojektuj bazę danych, najczęściej z kilkudziesięcioma tabelami, a następnie wokół niej zbuduj system.

Zalety i wady

Zaletą tego typu podejścia jest jego prostota. Wyraźnie wydzielone główne odpowiedzialności techniczne pozwalają nawet mało doświadczonym osobom zrozumieć architekturę. Dlatego jeśli dysponujemy zespołem o względnie niskich kompetencjach, tego typu konstrukcja może być dużym plusem.

Monolityczność rozwiązania najczęściej jest powiązana z tym, że dla systemu jest jedna baza kodu, która najczęściej kompluje się do jednego artefaktu (typu jar lub assembly), co powoduje, że wnioskowanie na temat struktury projektu i sposobu wdrażania jest dość proste. Ta własność ma też swoją drugą stronę – struktura projektu może stać się bardzo rozbudowana, a sam artefakt duży.

Wspólny, monolityczny model danych dość szybko powoduje, że system staje się implementacyjnie skomplikowany, a pytania o konkretny zestaw danych są trudne do napisania i często niewydajne. Ponieważ zachowania są implementowane w innych klasach niż dane (najczęściej Service'ach lub Managerach), to istnieje duże ryzyko, że te same operacje będą powielone w kilku miejscach, a same klasy będą się nadmiernie rozrastały.

Ponieważ system stanowi jedną całość, obciążenie jednej części systemu może wpływać na spowolnienie innej części systemu. Kiedy zdecydujemy się na skalowanie aplikacji, skalujemy całość, nawet jeśli tylko wybrana część systemu tego wymaga.

Opisana klasyczna architektura warstwowa przedstawia najczęściej stosowany sposób implementacji, który może się różnić w zależności od projektu. Jednocześnie warto dodać, że architektura warstwowa sama z siebie nie implikuje monolityczności modelu ani samego systemu.

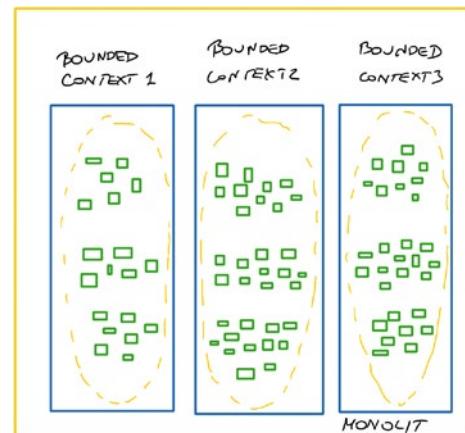
DOMAIN-DRIVEN DESIGN

Kolejnym podejściem, które przedstawimy, jest Domain-Driven Design, przynoszące koncepcje, które jeszcze kilkanaście lat temu wydawały się kontrowersyjne, a obecnie stanowią bazę dla projektowania nowoczesnych aplikacji i są świetnym źródłem wskazówek dla tworzenia aplikacji w architekturze microservices. Uściślimy, iż Domain-Driven Design (DDD) samo w sobie nie jest architekturą, jest raczej zbiorem technik i modeli mentalnych, które możemy zastosować, tworząc złożony system. Ponieważ jednak metoda DDD zawiera sporo wytycznych odnośnie osadzenia jej w kontekście architektury, potraktujemy ją na równi z pozostałymi opisami.

Domain-Driven Design wynosi na piedestał myślenie biznesowe – to model mentalny eksperta dziedzinowego oraz słownictwo, którego używa, powinno być głównym punktem odniesienia do tego, jak powinniśmy strukturyzować aplikacje. Każdy niebanalny system to najczęściej kilka, kilkanaście, a nawet kilkadziesiąt obszarów funkcjonalnych, które tworzą niezależne części zwane Bounded Contextami wraz z charakterystycznym dla niego słownictwem zwanym Ubiquitous Language. Pozornie to samo pojęcie może mieć różne znaczenie, dla różnych osób korzystających z systemu. Na przykład w systemie CRM możemy mieć do czynienia z klientami na różnym etapie współpracy:

- » lead – klient, z którym dopiero chcemy nawiązać kontakt, zazwyczaj mamy do niego telefon lub email i chcemy ocenić jego potencjał zakupowy,
- » client – klient, który już coś od nas zamówił, komu wystawiamy faktury,
- » customer – klient obsługiwany przez help desk, zgłaszający problemy, które powinniśmy w ramach wsparcia rozwiązać.

Mimo że laikowi może się wydawać, że w każdym przypadku ma do czynienia z klientem, to każdy z tych przypadków ma inną charakterystykę, inne dane, które są z nim powiązane, oraz inne operacje związane z procesami, które go dotyczą, dlatego traktujemy je jako osobne podsystemy.



Rysunek 5. System podzielony na kilka Bounded Contexts

Zgodnie z DDD można zbudować większy system, który logicznie jest podzielony na wiele Bounded Contexts, a fizycznie cały czas jest to jeden artefakt wdrożeniowy. Albo idąc dalej, można potraktować każdy Bounded Context jako autonomiczny podsystem i wtedy zbliżamy się do modelu microservices, o czym więcej w kolejnym artykule.

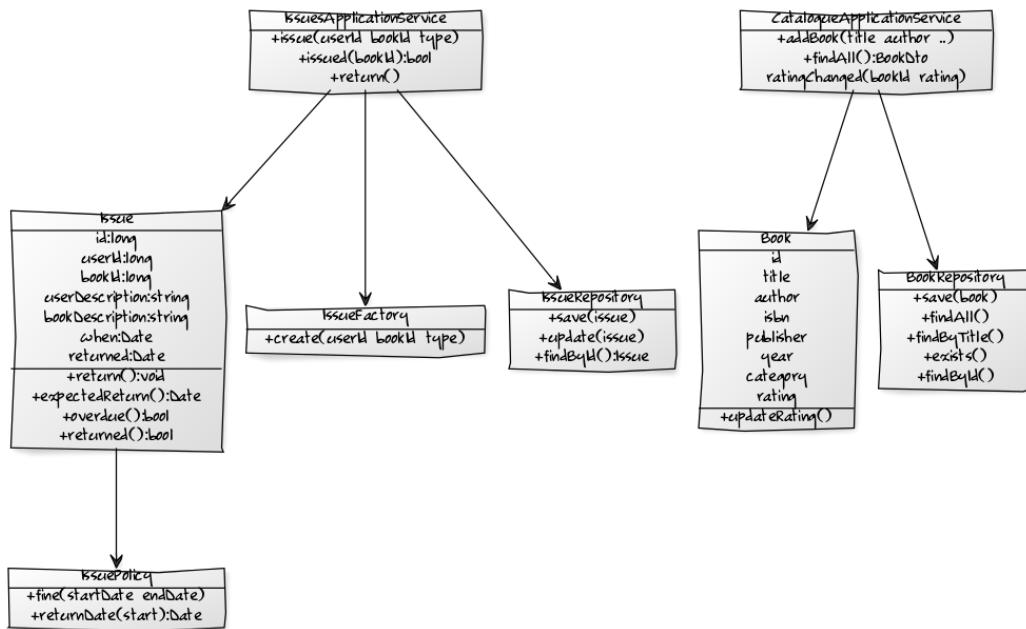
Domain-Driven Design proponuje również bloki budujące, z których może składać się aplikacja, z których przykładowe to:

- » Entity – klasy reprezentujące znaczące pojęcia dziedzinowe,
- » Value Object – klasy reprezentujące proste, szeroko wykorzystywane obiekty, np. kolor, pesel, kod pocztowy, kwota waluty,
- » Aggregate – zbiór kilku Entity i Value Objects, które tworzą bardziej złożoną całość, np. zamówienie, produkt,
- » Repository – abstrakcje umożliwiające pracę z danymi (typu twórz, zaktualizuj, wyszukaj) o interfejsie kolekcji,
- » Services – klasa reprezentująca operacje najczęściej na wielu agregatach, które trudno powiązać z konkretnym obiektem,
- » Policy – reprezentacja algorytmów implementująca wzorzec Strategii,
- » Application Services – reprezentują zewnętrzną warstwę dla klientów, związane z konkretnymi przypadkami użycia.

Przykład systemu do wypożyczania książek jest dość prosty, więc wyodrębnianie Bounded Contexts z praktycznego punktu widzenia wydaje się być zdecydowanie na wyrost. Niemniej chcemy zobaczyć, na czym polega ten koncept, a w szczególności wydzielania modelu.

W przypadku systemu wypożyczania książek moglibyśmy wyodrębnić następujące Bounded Contexts:

- » Books Catalogue – przypadki użycia związane z przeglądaniem informacji o książkach,



Rysunek 6. Diagram klas dla przykładowych Bounded Contexts

- » Issues – przypadki użycia związane z wypożyczaniem książek,
- » Ratings – przypadki użycia związane z ocenianiem książek,
- » Users – przypadki użycia związane z zarządzaniem użytkownikami.

Przykładowa uproszczona struktura klas dla Books Catalogue oraz Issues mogły wyglądać tak, jak na Rysunku 6.

Zwróćmy uwagę na to, iż modele w poszczególnych Bounded Contextach są niezależne. Jeśli odwołują się do siebie, np. w wypożyczeniu jest odwołanie do książki, to następuje to poprzez identyfikator agregatu (bookId). Co więcej, jeśli wypożyczenie potrzebuje charakterystycznych informacji na temat książki, przechowuje go we własnym modelu (Issue.BookDescription). Zauważmy również, że klasy modelu zawierają zachowanie (Issue.Overdue(), Issue.ExpectedReturn()), w tym również mogą mieć wstrzyknięte zależności (Issue.IssuePolicy).

Wraz z takim podejściem pojawia się wiele pytań:

1. W jaki sposób synchronizować dane?
2. W jaki sposób Bounded Contexts powinny się komunikować?
3. Jak przeprowadzać transakcje, które mogą dotyczyć wielu agregatów (wielu Bounded Contexts)?

Tymi tematami zajmiemy się bliżej przy omawianiu podejścia microservices oraz architektury reaktywnej.

Zalety i wady

Zdecydowaną zaletą stosowania DDD jest to, że przestajemy myśleć o wielkim systemie z wielkim modelem jako całością, a składającym go z mniejszych części, często posiadających różnych użytkowników biznesowych. Nie próbujemy usilnie uwspółniać danych potrzebnych w różnych częściach systemu, możemy potraktować je niezależnie. Dzięki temu również implementacja procesów i algorytmów jest prostsza. Łatwiej podzielić pracę pomiędzy zespołami, gdyż dany zespół zajmuje się jednym lub kilkoma Bounded Contextami i jest ich właścicielem.

Z drugiej strony podejście DDD wymaga dobrych kompetencji programistycznych, bardzo sprawnego posługiwania się programowaniem obiektowym, więc może być trudne do zastosowania

w mniej kompetentnych zespołach. Praca nad rozwojem modelu powinna odbywać się inkrementacyjnie, tzn. model powinien być ulepszany w kolejnych iteracjach, tak aby odzwierciedlał rzeczywiste wymagania biznesowe, raczej trudno będzie go zaprojektować w całości na początku (up-front). Niezależność Bounded Contexts powoduje, że należy włożyć spory wysiłek w ich integrację, synchronizację danych oraz zapewnienie ich spójności. Większa elastyczność ma swój koszt.

Z punktu widzenia rozwoju architektury w początkowo niewielkim systemie można stopniowo wyodrębniać kolejne Bounded Contexty wraz z jego rozrostem. Jeśli zrobimy to na odpowiednim etapie, operacja ta nie będzie kosztowna, a pozwoli ewolucyjnie wyodrębniać kolejne części systemu w razie potrzeb.

PODSUMOWANIE

Niniejszym przedstawiliśmy pierwsze z dwóch podejść do strukturyzowania aplikacji: architektury warstwowej oraz Domain-Driven Design, relatywnie najprostszych ze wszystkich podejść, które zostaną omówione w ramach cyklu, m.in. dlatego, że nie pojawia się tu jeszcze koncept rozproszenia aplikacji. Są one często dobrym punktem startowym, który może nas ewolucyjnie przygotować na wprowadzanie większej złożoności do systemu (jak to będzie miało miejsce w architekturze microservices, reaktywnej czy serverless). Zostały przez kilkanaście ostatnich lat zweryfikowane w boju, więc możemy być świadomi wszystkim pozytywnych i negatywnych konsekwencji, które ze sobą niosą.

MARIUSZ SIERACZKIEWICZ

m.sieraczkiewicz@bnsit.pl



Od ponad dwunastu lat profesjonalnie zajmuje się tworzeniem oprogramowania. Zdobyte w tym czasie doświadczenie przenosi na pole zawodowe w BNS IT, gdzie jako trener i konsultant współpracuje z czołowymi polskimi zespołami programistycznymi. Jego obszary specjalizacji to: refaktoryzacja, czysty kod oraz technical leadership. Autor metody Naturalnego Porządku Refaktoryzacji ®. Autor książki „Technical Leadership. Od eksperta do lidera”.



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

Sprzedaż



Dystrybucja



Import na zamówienie

TTS Company Sp. z o.o.

ul. Domaniewska 44A

02-672 Warszawa

www.tts.com.pl

UX dark patterns ciąg dalszy, czyli jak z głową zarezerwować hotel

Twórcy serwisów nieustannie poprawiają customer experience, czyli ścieżkę swojego potencjalnego klienta. Dzięki usprawnieniom sprzedaż wzrasta, gdyż coraz więcej osób nie tylko przegląda ofertę, ale ją finalizuje udaną płatnością. Organizacja wymarzonych wakacji to nie jest prosty temat, w związku z tym presja – strona działała jak najlepiej, a cały proces był prosty – jest olbrzymia. Nie tylko zresztą presja, potencjalne zyski są też spore i warto o nie zawałczyć. Dlatego tak wiele różnego rodzaju ciekawostek UX-owych można znaleźć na stronach związanych z podróżowaniem, a konkretnie z rezerwacją hoteli i pensjonatów.

Mowa o znanej (zapewne większości) stronie firmy z Amsterdamu. Znanej nie tylko z tego, że całkiem łatwo i wygodnie można dzięki niej zarezerwować nocleg na wyjazd, ale też z tego, że wywiera dosyć dużą presję na użytkownikach.

Serwis aggrejuje informacje o różnego rodzaju hotelach większych i mniejszych, pensjonatach, pokojach gościnnych. Każdy może dołączyć, dodać swoją ofertę. Społeczność weryfikuje takie miejsce i gdy okaże się niskich standardów – wypadnie w końcu z serwisu. Bariera wejścia z pozycji osoby zgłaszającej swój obiekt nie jest duża. Opłaty za umieszczenie oferty też są na poziomie akceptowalnym – zależnie od regionu jest to kilkanaście procent, odliczanych od ceny, jaką właściciel obiektu prezentuje gościom (co oznacza, że gość płaci dokładnie tyle, ile widzi na stronie, zaś serwis rozlicza swoją prowizję z właścicielem obiektu poza transakcją, w której bierze udział potencjalny gość).

Dlaczego serwis stał się tak popularny? To dosyć proste – aggrejuje bardzo dużo obiektów, utrzymując zarazem spójny (a więc porównywalny) sposób opisu miejsc, ustalanie cen czy formularz rezerwacji i płatności (jeśli jest on-line). Oferuje darmowe przewodniki i mapy. Prezentuje opinie innych gości, by wybór danego lokalu na pewno był uwiarygodniony (w końcu rekommendacja innych to najlepsza reklama, a ostrzeżenie od rzeczywistych osób jest największą antyreklamą). Z punktu widzenia klienta jest to wyasmine rozwiązanie.

Z kolei z perspektywy wystawiającego swój obiekt zalet też jest wiele: serwis pomaga ulepszać wyświetlane oferty, pozyjonuje, ułatwia współpracę z firmami oferującymi inne usługi, np. transportowe czy gastronomiczne. Co ważne – dodany obiekt można łatwo obejrzeć, sprawdzić dostępność w kalendarzu. Zdejmuję to wielu drobniejszym graczom na rynku problem z głowy w postaci konieczności stworzenia strony internetowej z opcjami rezerwacji i płatności. Do tego firma dosyć agresywnie promuje swoją platformę oraz wystawione w niej hotele czy pokoje. A to nie wszystko, bo tak naprawdę serwis pomaga... „nagonić” klientów, którzy wybiorą pokój za pośrednictwem platformy od razu, gdy tylko na stronę wejdą.

Jak firma z Amsterdamu to robi, że odwiedzający, którzy chcieli się rozejrzeć po cenach, kończą swoją wizytę opłaconą rezerwacją? Tu wkraczają wszystkie UX-owe sztuczki i chwyty, dobrze przemyślane i często przetestowane, znane pod zbiorczą nazwą *dark patterns*.

ZACZNIJMY OD POCZĄTKU!

Jak wymóc działanie na użytkowniku, który przed sekundą otworzył stronę główną serwisu? Wystarczy wyświetlić informację o dużym rabacie (Rysunek 1)! Co powiecie na 50%? Wystarczy się zalogować bądź zarejestrować!



Rysunek 1. Zachęcające do zalogowania rabaty*

Trudno w spokoju przeglądać oferty, gdy atakuje nas tak kuszący baner. Co daje zalogowanie do serwisu? Nam, jako klientom, rabaty (może, bo przecież to nie jest żadna konkretna oferta, tylko zapowiedź, że jakieś zniżki gdzieś tam są...). A co firma zyskuje, jak użytkownicy się logują? Bardzo wiele! Na przykład możliwość śledzenia, co interesuje daną osobę: jakie lokalizacje, które terminy, ceny, konkretne obiekty. Wystarczy raz rozejrzeć się po ofertach w serwisie, by do przysłowiowego „końca życia” ścigały nas mailinki z „kuszącymi jak pączki” (kto widział, to wie :)) ofertami w tych właśnie lokalizacjach, cenach, terminach itd. itp, które oglądaliśmy. To sprawia, że użytkownicy wracają. A gdy wracają, rezerwują. A gdy rezerwują – firma zarabia. I ten cykl się powtarza.

CO DALEJ? CZYLI PRZYKŁADY

Czas uruchomić wyobraźnię. Weźmy pod uwagę następujący przykład: szukamy hotelu w Krakowie, w maju. Daty i miejsca przypadkowe. Wklepujemy dane w wyszukiwarkę i widzimy... takie przyjemne komunikaty (Rysunek 2 i Rysunek 3).

Czy zaczynacie czuć presję? Widzicie już ten urlop pod namotem, bo nie chciało się Wam szukać noclegu wcześniej, a teraz – jak widać – „dostępność spada” i aż „538 osób szuka” w tym terminie, w tym mieście, tego samego, co Wy! Co teraz?



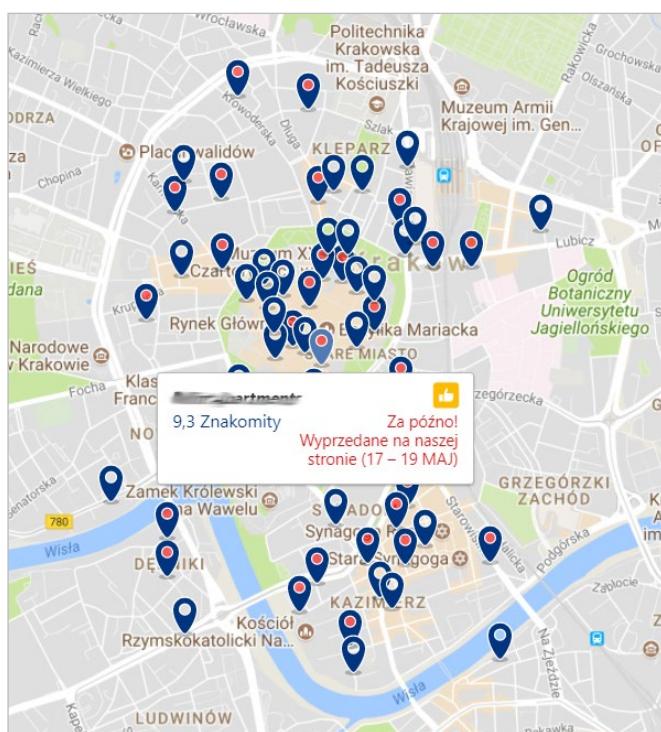
Rysunek 2. Dużo osób szuka w tym samym momencie



Rysunek 3. Dostępność spada!

Wygląda na to, że trzeba się zdecydować już i teraz. I na to firma liczy – że jako użytkownik serwisu zobaczycie te komunikaty, przestraszycie się, że zaraz wszystkie ciekawe (albo w ogóle wszystkie!) noclegi znikną z mapy i urlop będzie w ruinie.

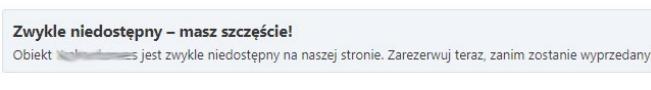
Takie wrażenie pogłębia mapa. Spójrzcie na Rysunek 4:



Rysunek 4. Mapa hoteli i pensjonatów

Widać wyraźnie: jeszcze przed chwilą było mnóstwo dostępnych miejsc. Ale „za późno!”. Piny z czerwonymi kropkami pokazują na mapie te hotele czy pensjonaty, gdzie zupełnie nieoczekiwanie, dosłownie przed chwilą, miejsca się skończyły. Pinów z białą kropką jest podobna ilość, ale nagromadzenie znaczników z alarmującym czerwonym kolorem potęguje niepokój i skłania do pośpiechu.

A spieszyć się trzeba, o czym przekonuje podgląd w jakimś dośćnym hotelu (klik w biały pin). Na Rysunkach 5 i 6 pokazano kolejne komunikaty, jakie atakuują oglądającego ofertę:

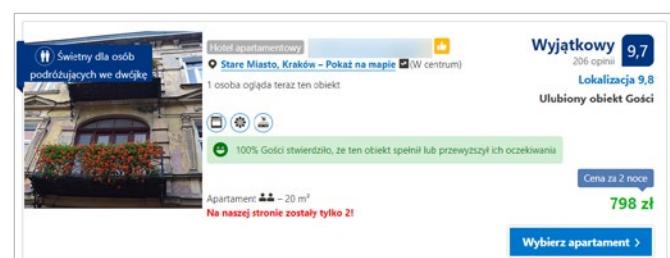


Rysunek 5. Zwykle niedostępne, ale Ty masz szczęście!



Rysunek 6. Uwaga, za moment ceny wzrosną!

Presja jest coraz większa! Co więcej: słusznie. Znajdujecie bowiem ciekawy hotel, jest dostępny, ale: widzicie, że zostały tylko dwa apartamenty (Rysunek 7). Bezpośrednio pod taką ofertą mamy inną, z komunikatem „Niewiele Ci zabrakło” (Rysunek 8). Czy to znaczy, że ta oferta była aktualna sekundę temu?



Rysunek 7. Jeszcze tylko dwa!



Rysunek 8. A to pech! Sprzedalo się... kilka dni temu!

Jak się okazuje, nie. Ta oferta jest nieaktualna od kilku dni. Ale trzeba to doczytać – a w pędzie i stresie jest ogromna szansa, że użytkownik tylko rzuci okiem, po czym przestraszy się czerwonych alertów. Co to da? Otóż zwiększy prawdopodobieństwo, że jeśli oglądamy jakąś ofertę i nie mamy pewności, to tę pewność „zyskamy”, widząc, jak szybko inne miejsca stają się niedostępne. W końcu urlop to ważna sprawa, lepiej zdecydować szybko i „coś” mieć, niż nie mieć gdzie spać, prawda?

A GDYBY TAK MIEĆ CZAS I PRZYJRZEĆ SIĘ OFERTOM?

Załóżmy, że nie dajemy się ponieść emocjom i dajemy sobie czas, by rozejrzeć się w ofertach. Chcemy sprawdzić opisy danych lokalizacji, porównać ceny. Nawet jeśli obiekty znikają (albo mają opisy, że dosłownie przed sekundą jeszcze były na liście...), nie stresujemy się, szukamy wytrwale. Czytamy opinie, po których spodziewamy się uzyskać więcej rzetelnych informacji. Opinia społeczności jest ważna, w końcu komentują osoby, które w tych miejscowościach były, widziały na własne oczy, zrobiły zdjęcia. Ufamy.

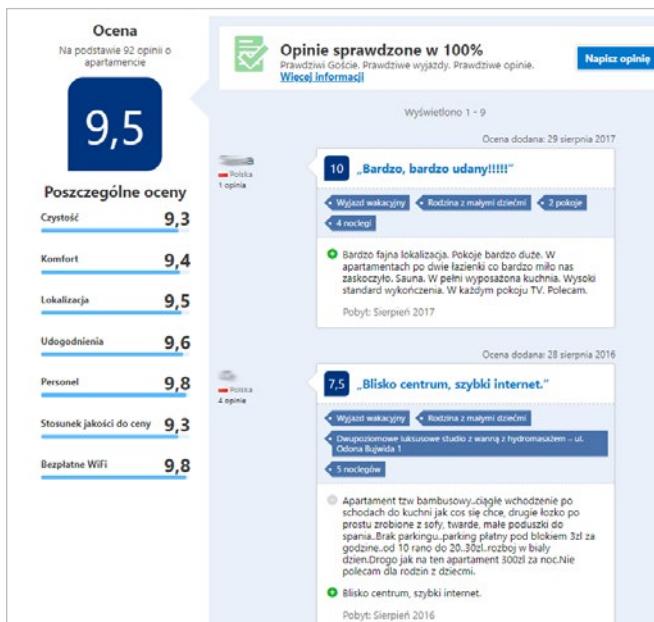
Rysunek 9 to przykład opinii, jaką ma jeden z hoteli. Jest to opinia zbiorcza, jak widać – na bazie 91 opinii (czy to mało, czy to dużo, niech każdy oceni sam). Obiekt ma opis „Wyjątkowy”, co sugeruje, że przytaczająca większość z tych opinii była bardzo dobra. Eksponowane, ostatnio dodane opisy bardzo zachęcają! To pocieszające, wzbudzające chęć zarezerwowania noclegu. Czy jednak na pewno?



Rysunek 9. Wyróżniona opinia

Spójrzmy na Rysunek 10. Wszystkie screeny są z kwietnia 2018 roku. Są to realne ujęcia – takie, jak każdy z nas by wtedy mógł zobaczyć, szukając na maj jakiegoś noclegu w Krakowie. Oglądamy więc oferty, zerkamy na opinie – są dobre! Tylko chwila, z jakiego okresu tak naprawdę pochodzą? Uwaga, dwie najwyżej wyświetlone (i prezentowane w miniaturce na ekranie głównym oferty tego miejsca) są z... sierpnia 2017 roku i sierpnia 2016 roku. „Aktualne”, biorąc pod uwagę, że są sprzed roku i sprzed dwóch lat. Rodzi się pytanie – czy to manipulacja, czy jeszcze nie?

Godna polecenia jest też druga opinia z wyświetlanych – ma wysoką notę (7,5 to nie aż tak źle). Sam komentarz jednak nie jest pozytywny: *niewygodnie, drogo, nie polecam*. Suma ocen jest jednak tak liczona, że i tak wychodzi całkiem nieźle. Jak więc zaufać opiniom? Czyżby znowu potwierdzało się, że ufać można, ale tylko negatywnym? :)



Rysunek 10. Widok opinii

I KWINTESENCJA!

Na koniec Rysunek 11, czyli zbiorczy widok oferty jakiegoś pokoju, tak charakterystyczny dla serwisu. Aż cztery komunikaty są podświetlone kolorem czerwonym, każdy ma na celu wzbudzenie pośpiechu w oglądającym – nawet nie zachęcenie, ale wymuszenie kliknięcia „rezerwuj”. Czytajmy to tak: jest pokój z dwoma łóżkami pojedynczymi. „Zostały tylko 3 pokoje” – nie jest źle. Ale jak się przyjrzymy, to „ktoś właśnie dokonał tu rezerwacji”, więc zaczyna się robić nerwowo. Zwłaszcza że „6 osób ogląda teraz ten obiekt”. Czy się wahamy? Jeśli tak, wyskakuje okienko z „alertem cenowym”. Taka okazja się nie powtórzy, co więcej: ta też zaraz zniknie!... Klikamy?



Rysunek 11. Kwintesencja

PODSUMOWUJĄC

Zaprezentowane działania są bardzo dobrym przykładem, jak wykorzystać dobre UX-owe pomysły, by wymusić na użytkownikach określone działanie. Jak oglądających stronę zestresować, ponaglić, przekonać. Efektem jest zarobek serwisu, zadowolenie wystawiających oferty (ruch w interesie i też zarobek), a nawet zadowolenie przeglądających. Czasami faktycznie lepiej szybciej podjąć decyzję, mieć zarezerwowany nocleg i „całe to szukanie z głowy”. Tylko czy na pewno chcemy planować urlop pod taką presją, z wyskakującymi alertami, straszyciemy, że zaraz coś nam umknie? Może idea samego urlopu nam tu też umyka?

Na koniec warto zaznaczyć, że dbanie o dobry, przejrzysty UX jest działaniem na korzyść użytkownika. Takie jest podstawowe założenie. Szkoda, że czasami się gdzieś „gubi” w walce o rosnące słupki sprzedaży.

*Rysunki pochodzą z serwisu świadczącego usługi rezerwacji noclegów online i na potrzeby artykułu zostały zmodyfikowane



KATARZYNA MAŁECKA

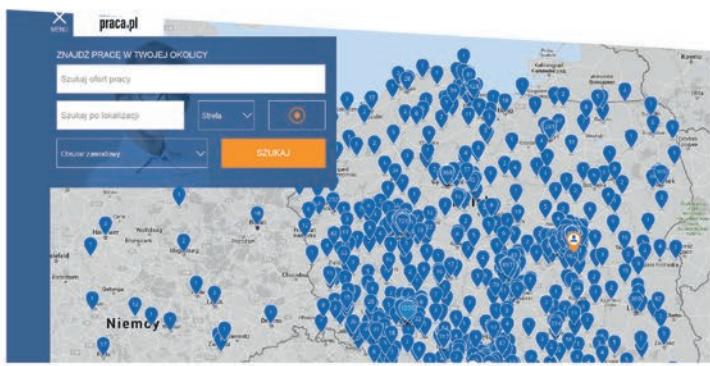
katarzyna@teina.co

Autorka od 9 lat odpowiada za prowadzenie projektów IT, posiada także praktyczne doświadczenie z zakresu UX: tworzenia logiki user flows, prototypowania (makietki lo-fi) oraz prowadzenia badań jakościowych w formie indywidualnych wywiadów, przy użyciu eye-trackera oraz EEG. Zawodowo: właścicielka agencji badawczej. Prywatnie: autorka bajek dla dzieci.



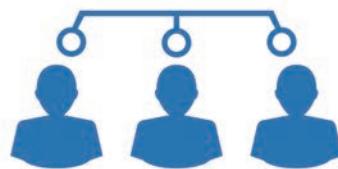
Chcesz dobrze zarobić?

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

Ciemne strony Javy 8

Java 8 była jedną z najcieplej przyjętych wersji tego języka wydanych w ciągu ostatnich kilkunastu lat, jednakże nie była pozbawiona wad, miejsc, w których z łatwością dało się poprawić funkcjonalność, czy najzwyklejszych w świecie wpadek.

PARALLEL STREAMS

Wprowadzenie prostego zrównoleglania operacji na kolekcjach poprzez wykorzystanie Stream API było jednym z najjaśniejszych (i najczarniejszych) punktów Javy 8.

W teorii, aby zrównoleglić dowolną operację na kolekcji, wystarczy skorzystać z jednego słowa kluczowego i cieszyć się natychmiastowym skokiem wydajności.

Na przykład:

```
List<String> strings = Arrays.asList("1", "2", "3", "4");
strings.parallelStream()
    .forEach(x -> System.out.println(x + "-" +
        Thread.currentThread().getName()));
```

I faktycznie po uruchomieniu powyższego kodu możemy zauważyc, że elementy wypisywane są w niedeterministycznej kolejności oraz że każdy z nich przetwarzany jest w innym wątku.

Zbyt piękne, żeby było prawdziwe? Owszem. Problem łatwo jest przeoczyć w czasie szybkiego testowania nowej funkcjonalności (tak jak powyżej), ale staje się on widoczny, kiedy zaczynamy myśleć o środowisku produkcyjnym i musimy odpowiednio dobrą parametry puli wątków. Wtedy wychodzi na jaw, że po prostu nie możemy tego zrobić, gdyż twórcy API nie udostępnili operacji pozwalającej na przekazanie własnej, skonfigurowanej puli wątków, w której wykonywane byłyby zrównoległe operacje.

Po chwili przeglądania kodu okazuje się, że Parallel Streams wykorzystują współdzieloną statyczną instancję puli ForkJoinPool, którą możemy znaleźć w klasie o tej samej nazwie:

```
public class ForkJoinPool extends AbstractExecutorService {
    static final ForkJoinPool common;
    public static ForkJoinPool commonPool() {
        return common;
    }
    //...
```

Współdzielenie jednej puli wątków przez wszystkie zrównoległe strumienie w naszej aplikacji może błyskawicznie doprowadzić do jej wysycenia i znacznej degradacji czasu przetwarzania (zwłaszcza jeśli wykonywane są operacje blokujące).

Na szczęście, skoro wiemy, że Parallel Streams korzystają z ForkJoinPool, możemy wykorzystać „trick”, który pozwoli nam podzielić naszą własną pulę wątków. Wystarczy bowiem stworzyć własną instancję puli i uruchomić w niej nasze zadanie składające się z wykorzystania zrównoległonego strumienia:

```
ForkJoinPool customPool = new ForkJoinPool(42);
customPool.submit(() -> list.parallelStream() /*...*/);
```

Niestety, jest to możliwe tylko na podstawie obserwacji detali implementacyjnych – zrównoleglanie w oparciu o instancję Fork-

JoinPool nie jest częścią żadnego kontraktu i teoretycznie może ulec zmianie bez łamania kompatybilności wstępnej.

STREAM API I LENIWE PRZETWARZANIE

Stream API nawiązuje do koncepcji Leniwych Sekwencji (ang. *Lazy Sequences*) i nie jest pełnoprawną strukturą danych, lecz rodzajem iteratora.

Wykorzystanie leniwych operacji pozwala na operowanie na potencjalnie nieskończonych kolekcjach oraz na unikanie wykonywania niepotrzebnych operacji przy funkcyjnym przetwarzaniu kolekcji.

Na przykład:

```
List<Integer> list = Arrays.asList(1, 2, 3);
list.stream()
    .map(s -> s * 2)
    .map(s -> s + 3)
    .findAny();
```

Na pierwszy rzut oka może się wydawać, że wykonywane jest sześć operacji tylko po to, żeby zwrócić pierwszą wartość z brzegu. Nic bardziej mylnego – wykorzystanie leniwych operacji powoduje, że możliwe jest wzięcie pierwszej wartości, wykonanie na niej dwóch przekształceń map() i zwrócenie wyniku.

Niestety, operator flatMap() zawiera poważny błąd powodujący, że operacje wewnętrznie niego nie są wykonywane leniwie. Skutkuje on niepotrzebnym narzutem na wydajność oraz – w skrajnych przypadkach – wpadnięciem w nieskończoną pętlę przetwarzania.

Spójrzmy na prosty przykład:

```
Stream.of(42)
    .flatMap(i -> Stream.generate(() -> i))
    .findAny();
```

W przypadku gdybyśmy mieli do czynienia z poprawną implementacją, to powyższy przykład powinien zakończyć się po przetworzeniu pierwszego elementu, ale niestety tak się nie dzieje i skończy się dopiero po wyczerpaniu pamięci.

Problem został rozwiązany w Javie 10.

WYRAŻENIA LAMBDA I CHECKED EXCEPTIONS

Wprowadzenie wyrażeń lambda pozwoliło znaczco poprawić przejrzystość kodu pisaneego w Javie. Niestety, nie we wszystkich przypadkach – szczególnie niewygodna stała się obsługa Checked Exceptions w obrębie ciął wyrażeń lambda.

Spójrzmy na prosty przykład:

```
list.stream()
    .map(i -> i.toString())
    .map(s -> s.toUpperCase())
    .forEach(s -> System.out.println(s));
```

Wyobraźmy sobie teraz, że `toString()` i `toUpperCase()` rzucają wyjątek. Nagle cztery linijki kodu rozrastają się do:

```
list.stream()
    .map(i -> {
        try {
            return i.toString();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    })
    .map(s -> {
        try {
            return s.toUpperCase();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    })
    .forEach(s -> {
        try {
            System.out.println(s);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    });
});
```

Nawet jeśli nie interesuje nas, na którym etapie przetwarzania wyjątek się pojawił, i tak musimy obsłużyć każdy przypadek z osobna.

Powyższy problem można ograniczyć – na przykład – poprzez skorzystanie z biblioteki throwing-function (<https://github.com/pivovarit/throwing-function>).

WARUNKOWE KOŃCZENIE STRUMIENI

Czasem może się zdarzyć, że chcemy przerwać przetwarzanie strumienia po wystąpieniu jakiegoś warunku. Do osiągnięcia tego celu można użyć metody `filter()`, ale charakterystyka operatora powoduje, że kolejne elementy są ciągle pobierane ze strumienia do momentu jego zakończenia, co powoduje, że ta metoda nie może być wykorzystana przy pracy z dużymi oraz nieskończonymi sekwencjami.

Wyobraźmy sobie, że chcemy pobrać wszystkie elementy mniejsze od dziesięciu z listy składającej się z numerów od 1 do 1000:

```
Arrays.asList(1, 2, 3, ... 1000).stream()
    .filter(i -> i < 10)
    .forEach(e -> ...);
```

Powyższa operacja zadziała poprawnie, jednakże sprawdzone zostaną wszystkie elementy strumienia zamiast pierwszych dziesięciu, co skutkuje niepotrzebnym narzutem wydajnościowym.

Na szczęście w Javie 9 możemy już znaleźć nowy operator, który rozwiązuje powyższy problem: `takeWhile()`/`dropWhile()`.

Korzystając z niego, możemy zapisać powyższą operację jako:

```
Arrays.asList(1, 2, 3, ... 1000).stream()
    .takeWhile(i -> i < 10)
    .forEach(e -> ...);
```

I cieszyć się brakiem narzutu wydajnościowego, gdyż operator przewie przetwarzanie natychmiast po napotkaniu pierwszego elementu niespełniającego warunku.

PODSUMOWANIE

Java 8 przyniosła wiele nowych, interesujących rozwiązań, jak Optional, Stream API czy CompletableFuture, które zostały umiejętnie wplecone w istniejące rozwiązania, ale warto zdawać sobie sprawę z jej czarnych stron, których znajomość może uratować nam życie (albo przynajmniej produkcję).

Doprowadziło to do sytuacji, w której w ramach API jednej klasy zaczynają się ścierać ze sobą różne konwencje, lub po prostu istniejące rozwiązania blokują możliwość refaktoryzacji rozwiązań, które nie sprawdziły się w praktyce.

Kompatybilność wsteczna Javy jest jej wielką zaletą, ale i przekleństwem. Jak to mówią Amerykanie: *There ain't no such thing as a free lunch.*



GRZEGORZ PIOWWAREK

gpiowwarek@gmail.com

Jest niezależnym konsultantem, trenerem w firmie Bottega IT Minds oraz prelegentem na międzynarodowych konferencjach (Devoxx, Geecon, OracleCode Los Angeles). Na co dzień tworzy i rozwija wysoce wydajne systemy rozproszone, udziela się w projektach open-source, bloguje na 4comprehension.com oraz gra na gitarze elektrycznej w progresywnym zespole metalowym.

reklama



**Obserwuj
nas
na Twitterze**



@PROGRAMISTAMAG

To nie jest zawód dla starych ludzi

Wyobraź sobie, że mając 64 lata, ciągle pracujesz w korporacji, a do upragnionej emerytury został Ci już tylko rok. Managerowie ciągle wymyślają nowe metody produkcji oprogramowania, spotkania wloką się tak długo, że twój pęcherz już tego nie wytrzymuje. Dodatkowo musisz każdego dnia uczyć się nowych narzędzi, procedur i procesów. Odliczasz dni, by wreszcie udać się na zasłużony odpoczynek. Czy tak wyobrażasz sobie swoją przyszłość? Zapewne nie. I nie jesteś w tym osamotniony.

UWAGA!

Masz około 40 lat lub więcej i jeszcze nie pracowałeś w IT? To nie jest artykuł o Tobie. Zawarte w nim tezy dotyczą ludzi, którzy w IT przepracowali już kilkanaście, a przepracują i kilkadziesiąt.

Nie da się pracować w korporacyjnym IT całe życie. Przynajmniej nie na stanowiskach operacyjnych – jak programista czy tester. Choć problemy życia w dużej firmie dotykają również managerów. Każdego dnia jesteś narażony na walkę, monotonię i powtarzalność zadań, wypalenie, a po piętach depczą Ci tańsi pracownicy i wydajniejsze metody. Do tego na rynku już powoli pojawiają się ludzie, którzy nie znają życia bez technologii i Internetu. Przychodzą z zupełnie innymi umiejętnościami i podejściem do pracy.

Polskie IT zaczęło się tworzyć we wczesnych latach 90. Co prawda w PRLu mieliśmy jeszcze dość dobrze rozwiniętą inżynierię oprogramowania i nawet sukcesy na tym polu, ale komunę odcinamy grubą kreską. Duży korporacyjny świat zawitał do nas dopiero po czasach transformacji systemowej. Pojawiły się pieniądze na informatyzację państwa. Pojawiły się i zawód programisty, taki jakim znamy go dzisiaj. Na przełomie wieków skuszone tanią siłą roboczą i dobrymi umiejętnościami pracowników pojawiły się masowo w Polsce zagraniczne firmy, które budują oprogramowanie głównie na rynki zagraniczne. To był całkowity bum i znaczący wzrost zatrudnienia. Ludzie, którzy wtedy zaczynali, a zazwyczaj byli to magistrzy i inżynierowie po pięciu latach studiów, dziś albo nieuchronnie zbliżają się do czterdziestki, albo już ją przekroczyli.

CZAS NIE STOI W MIEJSCU

Będąc młodym programistą, na swoją pierwszą delegację zagraniczną wyjechałem do niemieckiego Mannheim. Były to początki XXI wieku i z radością ruszyłem na podbój nowego świata, który wydawał mi się malować jedynie w kolorowych barwach. To, co spotkałem na miejscu, było szokiem na wielu płaszczyznach, ale to do poznanych ludzi chciałbym odwołać się najbardziej. Pierwszy raz w życiu spotkałem programistów 50+. My, młodzi i rzutcy z Polski, nie mieliśmy doświadczeń z ludźmi, którzy tej pracy oddali już kilkadziesiąt lat swojego życia. Zapamiętałem ich jako smutnych panów, w szarych swetrach, z dziesiątkami małych codziennych rutyn. Przyciągali za swoimi starymi maszynami, jawnie okazywali nam niechęć, unikali rozmów i hurtowo odrzucali nasze pomysły racjonalizatorskie. W sumie to odrzucali wszystkie pomysły, bo był to czas, kiedy korporacja zaczęła głośno negować ich ulubiony mo-

del V i promować zwinność. Potem programistów z roczników powojennych spotykałem jeszcze w wielu miejscowościach – od Francji po Skandynawię. Wszyscy oni wiedli życie pełne codziennych małych dramatów z obawy o swoje zatrudnienie. Okopywali się w swoich rolach i obszarach kompetencyjnych i rozwalały, kiedy przyjdą i po nich. Mieli przekonanie, że kwestią czasu będzie to, kiedy ich miejsca pracy zostaną zabrane przez automatyzację czy tańszą siłę roboczą. Wtedy nie rozumiałem tego zamknięcia. Jak mówią, czas jest najlepszym nauczycielem. Dziś rozumiem ich bardzo, a lista potencjalnych zagrożeń rozszerzyła się o kolejnych wrogów, takich jak algorytmy uczące się i sztuczna inteligencja. Słusznie przewidywali, że ich czas w IT się kończy. Pamiętam jednak, że każdy z nich miał jakiś swój plan awaryjny.

Korporacyjne IT to nie jest łatwy kawałek chleba. Ciągle deadline'y, zmiany, wdrożenia nowych metod i narzędzi powodują niewyobrażalną presję, której wielu nie wytrzymuje. Zmigrował się już z SVNa na GITa, zastąpił Skype Asaną, a Eclipsa IntelliJem? A słyszałeś już, że są to podejścia passe i zaraz trzeba będzie wdrożyć nowy zestaw narzędzi? Ta biegunka narzędziowa w IT jest nie do zatrzymania. Oczywiście każde kolejne narzędzie jest efektywniejsze od poprzedniego. Robi to samo szybciej i lepiej, ale jest to następna rzecz do nauczenia się. Z upływem lat twoja wola do uczenia się staje się coraz mniejsza, a zdolność przyswajania nowych rzeczy się pogarsza.

Twoje kolana i plecy nie dają już rady siedzieć cały dzień. Prosiłeś o podnóżek? Biurko z regulowaną wysokością? Piłkę zastępującą krzesło? Czujesz, że ręce drębwieją i doczytałeś, że może chodzić o cieśń nadgarstka? Czasu nie oszukasz. Dziś to są jeszcze problemy do pominięcia, ale za chwilę może się okazać, że sypiesz się człowieku, i to sypiesz się na całej linii.

Nie chodzisz już na spotkania projektowe, ale na stand-upa i retro? Ktoś jednak uznał, że wyświetlanie Ci burdowa w toalecie na ścianie może nie jest najlepszym rozwiązaniem? Firmy płynnie modyfikują swoje strategie i na przestrzeni lat przechodzą od modeli kaskadowych, przez inkrementacyjne do zwinnych. Dziś robisz (pseudo)Scruma, ale już wiesz, że dostawy w iteracjach 2-tygodniowych są rzadkie i trzeba dostarczać od TERAZ. Idziesz już w kierunku w CI/CD i DevOps. A Ty marzysz tylko o tym, by mieć kubek pełen kawy, najwygodniejsze kapcie w biurze, najlepsze miejsce w open space-ie i móc spokojnie pokodować.

Do tego twoje wynagrodzenie maleje. Każdego roku, kiedy społgasz na średnie zarobki w branży, to dochodzisz do wniosku, że z twojej pensji chyba wypłacają gażę młodszym programistom. Realnie twoja płaca nie spada, ale po przekroczeniu magicznej bariery wieku zaczyna być mniejsza różnica w odniesieniu do pensji mniej

doświadczonych od Ciebie. Przez to spada Ci motywacja. W końcu to Ty uratowałaś w tej firmie niejeden projekt, a oni odstawiają Cię na śmiertnik historii!

A przecież niedługo będziesz miał 40, 50 i 60 lat. Wiek emerytalny jest przesuwany, bo i czas życia się wydłuża i może nawet mając siedemdziesiątkę na karku, ktoś każe Ci pisać kod. Korporacje nie lubią ludzi słabych i nie adaptujących się. Czy schorowany starszak będzie miał dla nich jakąkolwiek wartością czy też zostanie bezceremonialnie zredukowany? Ale czekaj, czekaj... Nie musisz wcale czekać do starości, aby poczuć, że IT Cię już nie potrzebuje.

WYPALENIE ZAWODOWE

Czy już od początku dnia czujesz zmęczenie, a może wynika ono z tego, że słabo spałeś? Myśl o pracy momentalnie wywołuje u Ciebie stany depresyjne? Masz kłopoty z koncentracją, a każde zadanie powoduje u Ciebie prawie fizyczny ból? Kłócisz się z kolegami w pracy i ciągle masz w sobie podenerwowanie? Na drodze dyszysz nienawiścią do innych kierowców? A może odczuwasz już ból głowy albo masz nadciśnienie? Może to oznaczać, że właśnie przeżywasz wypalenie zawodowe. Dla niektórych ludzi w IT przyjdzie ono w okolicach czterdziestki. Część zacznie odczuwać je jeszcze szybciej. Jak pokazują badania, wypalenie wynika z dwóch podstawowych czynników: stresu oraz kultury pracy. Choroba ta dotyczy częściej osoby, których głównym motywatorem do pracy są pieniądze niż te, które motywują się trudnością zadań do wykonania czy informacją zwrotną od kolegów z pracy.

Chcesz się „wyleczyć” z wypalenia, ale nie wiesz jak? Rozwiązań jest kilka. Większość pomoże Ci sobie poradzić i pozostać w miejscu, w którym jesteś.

W sieci znajdziesz wiele porad. Część osób próbuje poradzić sobie z wypaleniem przez zmianę miejsca pracy. Trzeba podkreślić, że może to być rozwiązanie krótkoterminowe, a w wielu przypadkach nieskuteczne. Przecież zmieniając pracę z programowania w jednym miejscu na drugie, często spotykamy się z tym samym otoczeniem projektowym czy firmowym. Wielu psychologów prady podkreśla, że kluczowe jest zrozumienie, dlaczego to, co sprawiało Ci do tej pory przyjemność i co chciałeś robić, nagle stało się przekleństwem? Co doprowadziło do tego, że masz już dosyć? Tu często nie obędzie się bez pomocy z zewnątrz (nawet psychologa), który pomoże znaleźć Ci odpowiedzi na pytania. Jest to jednak kolejna próba zatrzymania Cię tam, gdzie jesteś.

Często rozwiązaniem jest nauczenie się mówić NIE. Jeśli ludzie ciągle przychodzą i czegoś od Ciebie chcą, to na twoją głowę spada coraz więcej zadań. Odmów im. Naucz się asertywności, która pozwoli Ci realizować własne zadania i jednocześnie nie zrazić do siebie innych.

Czasami zmiany muszą być poważniejsze. Do działania napędza nas często chęć zmiany albo wola rozwijania umiejętności. Może czas na naprawdę dużą zmianę? Decyzja o zmianie framework'a programistycznego, a może nawet całego języka może postawić Cię przed zadaniem, jakiego nie doświadczyłeś od czasu pierwszej pracy. To naprawdę silny motywator.

Czasami obroną przed wypaleniem są naprawdę proste rzeczy. Może to kwestia efektywnego wypoczynku? Przecież po 8h kodyowania w firmie nie odpoczniesz w domu, spędzając kolejne 8 godzin przed komputerem. Wyświechtane work-life balance jest bardzo ważne dla twojej higieny psychicznej. Warto przestrzegać rad typu: nie odbieram telefonów od przełożonego po godzinach pracy i nie zaglądam do korporacyjnej poczty w weekendy.

Wszystkie te rozwiązania pozwalają Ci pozostać w miejscu, w którym ciągle jesteś. Firmy, które Cię (potencjalnie) zatrudniają – albo żyją z rekrutacji – nie chcą, abyś dokonywał drastycznych kroków. A takim posunięciem na pewno byłoby porzucenie źródła wypalenia i rozpoczęcia zupełnie innego zajęcia. Rozwiązaniem mogłoby być również znalezienie sobie nowego wyzwania, takiego jak na przykład dłuża podróź.

Swego czasu wzorcem do naśladowania stał się dla mnie szef moich szefów w Volvo IT. Powiedział kiedyś, a po niespełna roku zrealizował swój plan, że przed pięćdziesiątką przejdzie na „emeryturę”. Postanowił kupić jacht i wraz z żoną popłynąć w rejs dookoła świata. Tak ambitny plan oznacza jednak, że przestajesz pracować, więc przestajesz mieć wartość dla korporacji.

Prawdziwe historie

Darek jest rozwojodnikiem po czterdziestce. W korporacji doszedł już do poważnego stanowiska managerskiego, ale jedyne, o czym myśli w pracy, to kiedy się ona już skończy. Cały swój wolny czas poświęca na góry. Od wiosny do jesieni się po nich wspina, a w zimę jeździ na nartach. Czasami zabiera swoje dzieciaki. Wykorzystuje już nie tylko urlop płatny i weekendy, ale zaczyna również urlop bezpłatny. Kiedy dochodzi do ściany znużenia i zmęczenia, porzuca pracę i wyjeżdża na dłużej. Po powrocie bez problemu znajduje kolejną pracę.

UCIEC, ALE DOKĄD

Jeśli masz wystarczająco środków i jesteś osobą, która może funkcjonować bez poczucia stabilności, to dobrze dla Ciebie. Niestety nie ma takich wielu pośród nas.

Prawdziwe historie

Krzesiek jest programistą i pracuje w naprawdę dużej korporacji. Ma nieliczącą żonę Monikę i dwójkę dzieciaków. Żyją na wysokim poziomie w mieszkaniu na 30-letni kredyt (we frankach). Krzesiek chętnie porzuca pracę korporację i założył własny biznes, ponieważ ma kilka gwarantowanych zamówień na kodowanie. Zmęczył się już pracą w dużej firmie i ma poczucie jej bezsensu. Niestety Monika nie potrafi zrozumieć i nie chce zrezygnować ze stałego dochodu męża, wysokiego ubezpieczenia zdrowotnego i emerytalnego, prywatnej opieki medycznej i darmowej karty do siłowni. „Dorośnij! Przecież mamy dzieci” – potrafi powiedzieć w awanturze. Postanowił zostać w korporacji.

Większość ciągle musi pracować i zarabiać. Bezpiecznym rozwiązaniem jest inwestowanie środków zarobionych w IT. Pozwala to mieć stabilne zatrudnienie i jednocześnie, kiedy już korporacja nas porzuci lub my porzucimy korporację, mieć oszczędności na życie. Prawie każda osoba, z którą rozmawiam, a która myśli o zabezpieczeniu swojej przyszłości, inwestuje w nieruchomości. Czasami wystarczy posiadać jakiś kawałek ziemi albo wynajmować mieszkanie krótko- lub długoterminowo. Aktualnie nieruchomości są na krzywej wznoszącej i zdaniem analityków ten trend się utrzyma. Wydaje się to być dobra lokata środków. Na pewno nie warto trzymać funduszy na lokacie w banku. Przez inflację i niskie oprocentowanie realnie generuje to stratę. Pamiętaj również, że twoje środki zagwarantowane są tylko do kwoty pół miliona złotych, a unijne przepisy umożliwiają bankom w trudnej sytuacji finansowej przejmowanie pieniędzy klientów.

Prawdziwe historie

Michał bliżej ma już do pięćdziesiątki i już nie pracuje w korporacji. Zainwestował dużo środków własnych i kredytów w małe mieszkania i utrzymuje się z czynszu od wynajmujących. Jako rodzina żyją na co dzień na dość przeciętnym poziomie, ale są w stanie raz do roku pojechać na 8-tygodniowe wakacje wraz z dziećmi. Czasami zdarza się Michałowi zrobić szkolenie programistyczne dla zaprzyjaźnionej firmy, ale traktuje to jako odszkoczną od codziennego życia, a nie jako formę zarobkowania. Te dodatkowe środki przeznacza na fanaberie swoje lub swojej rodziny.

Ludzi, którzy grają na giełdzie czy inwestują w bitcoiny, pomijam. Szanuję ich odwagę w inwestowaniu i widzę w tym duży potencjał, ale dla mnie to zbyt duże ryzyko i losowość. Jeden z moich zaprzyjaźnionych programistów, który namiętnie inwestował na giełdzie, stał się wiceprezesem w Goldman Sachs, ale zawsze podkreślam, jeden. Tylko JEDEN! Oprócz tego widziałem dużo finansowych i rodzinnych tragedii związanych z giełdą. Do tej działy naprawdę trzeba mieć wyczucie i dużo szczęścia.

Pewnym rozwiązaniem jest ucieczka do startupów. Nie ma tam tego całego korporacyjnego naddatku procesowego. Nie jesteś anonimowym headcountem ani menhourem. Mały zespół daje większe poczucie wspólnoty i zbliżania się do zdefiniowanego celu działania. Startupy jednak muszą działać dynamicznie, a to powoduje, że dziś funkcjonują, a jutro może ich nie być. To miejsce pracy po raz kolejny nie daje nam stabilności.

Wiele osób, które znam i które zbliżają się do czterdziestki, zaczyna mówić głośno o chęci poważniejszej zmiany w swoim życiu. Mając już na koncie środki odłożone po kilku(nastu) latach pracy w IT, myślą o swojej przyszłości poza murami korpo. Możliwością jest ucieczka z korporacji we własne, małe biznesy ciągle związane z IT. Zamiast pracować na umowie o pracę możemy funkcjonować jako freelancerzy na umowie B2B. Część osób ucieknie w coaching albo w bycie trenerem. Część sama zacznie dostarczać zasoby do firm. To aktualnie bardzo bezpieczna opcja. Z jednej strony nie pracuje się bezpośrednio dla IT, z drugiej – ciągle dostarcza się jej usługi. Wypracowane znajomości i dobrze zbudowana sieć kontaktów bardzo to ułatwia. Korporacje potrafią dać naprawdę dobrze zarobić również swoim kooperantom.

Prawdziwe historie

Magda (39 lat) porzuciła korporację pomimo bardzo dobrych dochodów i niewielkich obowiązków. Ulokowała się w organizacji tak dobrze, że miała samodzielne stanowisko i w sumie nikt do końca nie wiedział, czym się zajmowała. W pracy miała jednak absolutne poczucie bezcelowości. Dziś oferuje usługi i szkolenia związane z zarządzaniem czasem i zadaniami.

Są i tacy, którzy完全 zmieniają obszar działania, a dotychczas zarobione pieniądze inwestują w swoje pasje. Jednym z bardziej świetlistych przykładów jest Michał Szafrański, który zajął się blogowaniem i doradztwem finansowym. Zarobił miliony na swojej książce „Finansowy ninja”. Ludzie inwestujący w swoje hobby mają przewagę ponad wszystkimi innymi. Często swoje biznesy rozwijają po godzinach, by, kiedy nastąpi już poczucie stabilności, porzucić korpo i zarabiać samodzielnie.

Prawdziwe historie

Pan Stanowski ma hobby. Lubi delektować się wysokoprocentowym alkoholem. Swoją pasję zamienił w bloga, a może docelowo w biznes. Degustuje wysokoprocentowe alkohole i to opisuje (<http://stanowski.it>)

W Polsce jest jeszcze wiele januszobiznesów, które można poconać, mając prawdziwą pasję i odrobinę marketingowego zacięcia. Jeśli znacie dany rynek, wiecie, czego potrzebuje, to zostało już tylko przełamanie się i działanie.

Prawdziwe historie

Marek jest uznany specjalistą i autorytetem w swojej dziedzinie IT. Mieszka w Warszawie wraz z żoną i dwójką dzieci. Pracuje zdalnie, ale w tym samym czasie projektuje koncept boksów garażowych do maksymalnego wykorzystania przestrzeni nad parkującym samochodem.

Wiele osób, które znam, myśli o otworzeniu własnej gastronomii. To takie marzenie z młodości, które niewielu udało się z powodzeniem spełnić. Założyciel CD Projekt zrobił własną wegeteriańską knajpę, a Ty nie dasz rady? Gastronomia to bardzo wdzięczny biznes, ale jednocześnie bardzo pracochłonny i trudny w realizacji. Kto nie spróbował, ten się nie przekona. Na tym rynku oczywiście funkcjonuje już wiele profesjonalnych firm, ale jest też wiele niskiej jakości restauracji czy kawiarni, które łatwo można pokonać odrębnią kreatywnością. Co ważne, rynek ten bardzo rośnie i zmienia się nastawienie Polaków, którzy coraz tłumniej odwiedzają lokale.

Wielu moich kolegów, jak również autor tego tekstu, próbuje inwestować również w projekty związane z IT albo w mniejsze startupy. Ma to sens, jeśli macie głębokie przekonanie poparte do głębnią analizą, że to zadziała. Pamiętajcie jednak, aby dopuszczać możliwość, że może jednak się nie udać. Moim zdaniem próbować zawsze warto. Jednorazowy strzał związany ze sprzedażą może Wam dać duży zastrzyk finansowy lub nawet ustawić do końca życia. Tutaj często zamiast środków można zainwestować również swoje umiejętności (np. kodowania).

Pamiętaj, że w każdej inwestycji najważniejszy jest z niej zwrot. Włóżysz złotówkę, chcesz wyciągnąć z niej złoty pięćdziesiąt albo nawet i dwa złote. Ważne jest jednak również to, by nie wyeksploatować się. Nie po to rzucasz korpo, by obierać ziemniaki i obsługiwać klienta. Twoim celem jest zarobić, a nie zamienić jedną trudną pracę na inną.

Podsumowując, nie dasz rady i pewnie nie wyobrażasz sobie, że spędziisz całe swoje życie w korporacji. Już dziś widzisz, że dają Ci zarobić tyle, że możesz zacząć inwestować i planować swoje życie poza firmą. Mam dla Ciebie jedną radę, z której kiedyś sam skorzystałem. Inwestuj! Działaj i zabezpiecz swoją przyszłość, a przy odrębnie szczęścia twoja emerytura nastąpi dużo szybciej niż obiecuja Ci przełożeni i politycy.

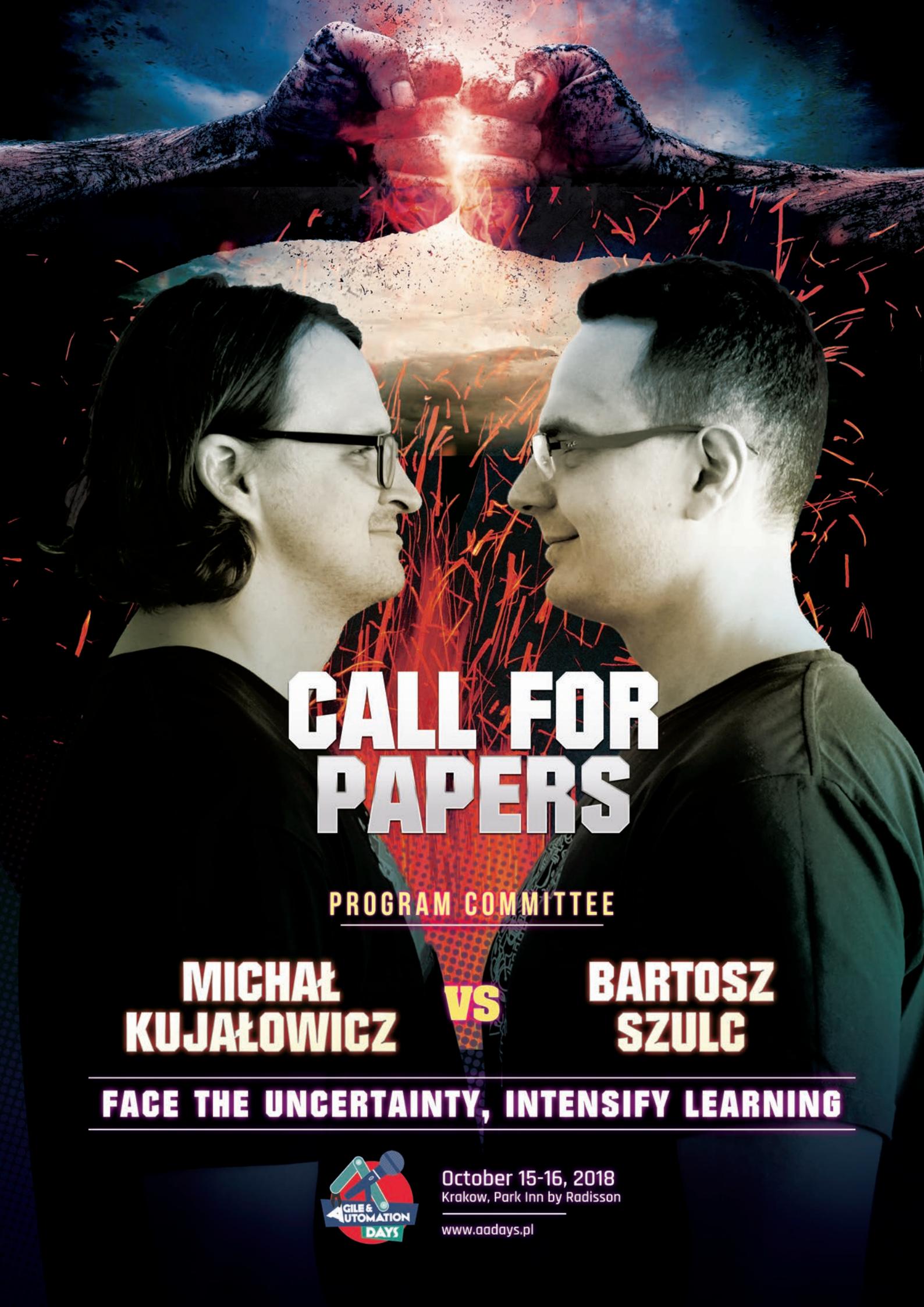
*) PRAWDZIWE HISTORIE są zbudowane o autentyczne doświadczenia osób, które znam, ale ze względu na prywatność tych osób zmieniłem ich dane osobowe i szczegóły, które mogłyby doprowadzić do ich rozpoznania.



RADEK SMILGIN

radek@testerzy.pl

Rocznik 77. Pieniądze zarobione w IT inwestują w gastronomię, nieruchomości oraz własne startupy. Część inwestycji to realizacja pasji, a część to inwestycje stricte komercyjne. Jeśli chcesz postułać o tym, jak inwestuję, jakie błędy popełniłem i jaką lekcję z tego wyciągnąłem, zaproś mnie na swoją konferencję lub zamkniętą prelekcję. Chętnie opowiem więcej.



CALL FOR PAPERS

PROGRAM COMMITTEE

MICHał
KUJAŁOWICZ

VS

BARTOSZ
SZULC

FACE THE UNCERTAINTY, INTENSIFY LEARNING



October 15-16, 2018
Krakow, Park Inn by Radisson

www.aadays.pl

Tata programistą

Jak znaleźć czas na kodowanie, będąc młodym rodzicem

Czy podjęlibyście się wyzwania w postaci projektu, który trwa co najmniej kilkanaście lat? Praca nad nim będzie trwać 24 godziny na dobę, 7 dni w tygodniu. Nie ma możliwości zmiany tego projektu na inny. Stawka godzinowa wynosi 0 zł na godzinę. Jedynym wynagrodzeniem jest radość życia. Jeżeli warunki Was zającają, to tak właśnie rozpoczęliście projekt pod nazwą „rodzic”. Gratulacje! Co dalej? Jak pogodzić ten najważniejszy projekt w naszym życiu z całą resztą naszych zainteresowań, w tym z programowaniem?

JAK TO JEST NAPRAWDĘ

Rzecz działa się kilka lat temu na 3-dniowej konferencji dla programistów. Nie miałem wtedy jeszcze założonej rodziny, ale wkrótce miało się to zmienić. Drugiego dnia po wykładach zaczęło się after-party, na którym dominowały bieżące tematy technologiczne. Im pora była późniejsza, tym tematy stawały się coraz luźniejsze. Pracowałem już kilka lat w zawodzie, więc znałem już kilka osób, z którymi mogliśmy otwarcie porozmawiać na każdy temat. Wtedy padło pytanie: „W jaki sposób ułożyć sobie życie tak, żeby mieć czas na programowanie oraz na dzieci i rodzinę?”. Osoba, która miała jedno dziecko, odpowiedziała pewnie: „Kiedy ja zajmuję się dzieckiem, żona ma czas dla siebie. Ale kiedy ona zajmuje się dzieckiem, ja mogę powrócić kodować”. Osoba z dwójką dzieci odpowiedziała również przekonywując: „Ja zajmuję się jednym dzieckiem, moja partnerka drugim. Kiedy dzieci pójdą spać, mamy czas, żeby usiąść do komputera”. Do odpowiedzi z wielkim zadowoleniem i radością podeszła trzecia osoba, z trójką dzieci. Z trójką dzieci tak się nie da.

„Zatrudniliśmy opiekunkę i sprzątaczkę – dzięki temu możemy wygospodarować czas dla siebie”. Ten wieczór i ta noc utwierdziły mnie w przekonaniu, że możliwe jest połączenie dwóch marzeń z dobrym skutkiem, to znaczy założenie rodziny i robienie kariery jako programista lub programistka.

Pełną parą

Przed założeniem rodziny żyłem pełną parą. Jak większość moich kolegów i koleżanek ze studiów pracowaliśmy na pół etatu lub więcej, jednocześnie studując na studiach dziennych. Całymi dniami byłem pochłonięty pracą i nauką. Czas na odpoczynek przychodził raz na jakiś czas i występował najczęściej w postaci wyjazdu ze znajomymi. Po studiach praktycznie nic się nie zmieniło. Jeżeli do zrobienia był projekt, to pracowaliśmy nad nim, ile nam tylko się stało. Świat wtedy wyglądał trochę chaotycznie. Nie było żadnego planu. Każdy z nas chciał zdobyć doświadczenie, nauczyć się dobrych praktyk oraz pracować przy najlepszych i najciekawszych projektach. Często nie musiałam planować następnego dnia, tygodnia czy miesiąca, ponieważ na wszystko zawsze był czas. Ży-

cie toczyło się z dnia na dzień. Można było poświęcić wiele godzin na komplikację jądra linuxowego, tylko po to, żeby się pobawić i eksperymentować z niestandardowymi ustawieniami. Ot tak, żeby zdobyć doświadczenie, a przy tym nieźle się bawiąc. Jednak wszystko to musiało się zmienić.

Pismo obrazkowe

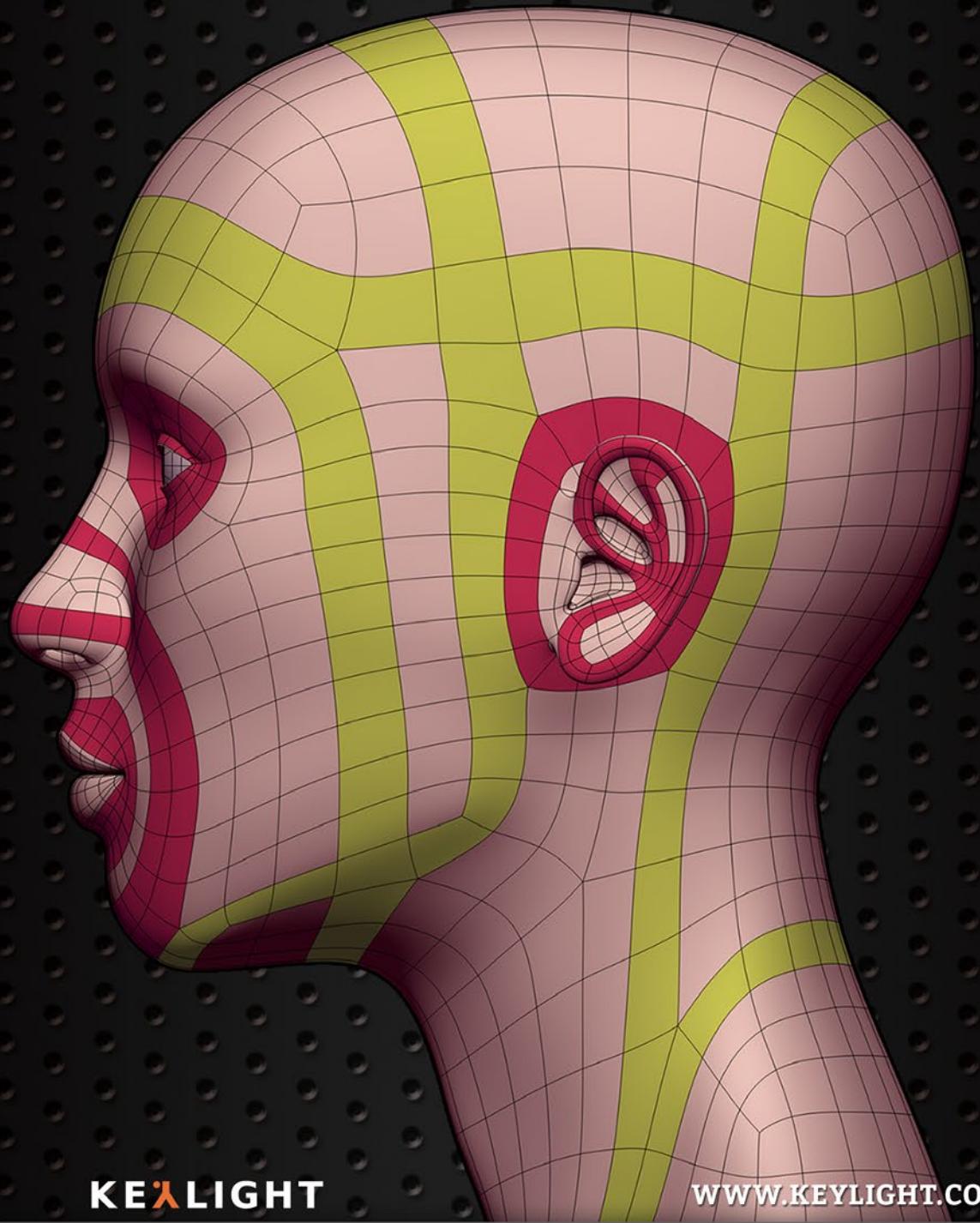
Media społecznościowe są częścią naszego życia – czy tego chcemy czy nie. W ten sposób komunikujemy się ze światem, w ten sposób mamy większe grono znajomych. Większość osób jest wyewentu w Internecie. W ten sposób kreujemy w naszych głowach jakiś plan dnia.

Rano wystawne śniadanie, następnie trzeba pobiegać, potem jedziemy do pracy, lunch, po południu można trochę popracować, ale po powrocie do domu jest gra online, dobra książka, przygotowanie wystąpienia na konferencję, napisanie posta na bloga, zrobienie pull requesta do open source, w międzyczasie możemy obejrzeć film lub serial – dla relaksu, tylko po to, żeby dzień zakończyć odpisywaniem na wszystkie listy dyskusyjne, na których jesteśmy. Nie wspomniałem o tym, że dobrze też raz na jakiś czas przeczytać najnowsze wpisy na obserwowanych blogach.

Czy taki dzień nie byłby czymś pięknym? Dla mnie tak. Czy się prowadzić taki tryb życia tak, żeby na wszystko mieć czas? Zdecydowanie nie. Musimy zdawać sobie sprawę, że nie starczy nam czasu na wszystko. Choćby nie wiem, jakbyśmy byli doskonali, pracowici i idealni. Codziennie musimy podejmować decyzje i rezygnować z wielu rzeczy, żeby te nieliczne udało nam się zrealizować. To od Ciebie zależy, co uda Ci się zrealizować, a co będziesz musiał odpuścić.

Brak czasu

Przestańmy się oszukiwać i spójrzmy prawdzie w oczy. Czasu na hobby lub pracę po nocach jest mniej, gdy jesteśmy rodzicami. Nikt mnie nie przekona, że tego czasu będziemy mieli tyle samo, będąc rodzicami. Z każdym kolejnym dzieckiem czasu będzie ubywać, nawet jeżeli będziemy korzystali z pomocy sprzątaczek, go-



KEYLIGHT

WWW.KEYLIGHT.COM.PL

sposi domowej i opiekunek. Zawsze instynkt rodzicielski domagał się tego, żeby spędzać ten czas z naszymi dziećmi.

Co w takim razie możemy zrobić?

Zacznijmy od ustalenia, dlaczego chcemy robić w życiu to, co robimy? Każdy z nas musi sobie szczerze odpowiedzieć na to pytanie. Może doprowadzi nas to do zupełnie innych wniosków? Dobrze jest mieć spisane swoje cele i marzenia. Jeżeli już je mamy, jesteśmy bardzo blisko sukcesu. Następnie zastanowmy się, jak chcemy je osiągnąć. Tutaj metod jest już wiele. Po pierwsze, możemy wygospodarować więcej czasu poprzez delegację lub nie wykonywanie pewnych czynności. Po drugie, możemy nauczyć się lepiej wykorzystywać dostępny czas. Każdy z nas musi znaleźć i odkryć swoją metodę. W kolejnych częściach artykułu będę opisywał, jak możemy to robić.

POZIOM OGÓLNY

Na problem nie powinniśmy patrzeć z poziomu ojca programisty czy mamy programistki. Jesteście rodzicami – jesteście jedną drużyną. Najlepsze, co możecie zrobić dla siebie nawzajem, to rozmawiać i współpracować ze sobą, tak żeby każde z Was mogło w jak największym stopniu realizować się zawodowo, tak by ta druga strona też była zadowolona, a rodzina się rozwijała. Jeżeli życie potoczyło się tak, że jesteśmy samotnymi rodzicami, w niektórych sytuacjach będziemy mieli trudniej.

Jeżeli mamy to opanowane, pozostałe elementy to detale implementacyjne. Niestety mamy mają troszeczkę trudniej, ponieważ tatusiowie nie zrobią tych czynności, które wykonuje mama. Na przykład nie nakarmią naturalnym pokarmem małeństwa. Niemniej aby wszystko się udało, musicie tworzyć jeden zgrany zespół.

Cel

Mam w życiu dużo celów do zrealizowania oraz marzeń do spełnienia. Kto ich nie ma! Cele mają nas do czegoś doprowadzić, pomóc osiągnąć jakiś krok milowy. Marzenia powinny być czymś uroczystym, czymś odświętnym oraz wspaniałym, czymś, co będzie nam nadawało sens życia. Ja mam spisany każdy swój cel, który chcę realizować, oraz każde marzenie, które przyjdzie mi do głowy. Czy spisuję je celem zrealizowania każdego z nich? Absolutnie nie. Spisuję je po to, żeby ich nie realizować. Kiedy skończę jakiś krok w swoim życiu – może to być z okazji nowego miesiąca, skońzonego modułu w projekcie lub zrealizowania jakiegoś celu lub marzenia – otwieram swoją notatkę, do której przyrostowo spisywałem rzeczy, które chciałbym osiągnąć. Przechodzę po liście punkt po punkcie. Z każdym punktem mogę zrobić jedną z trzech rzeczy:

- » **Zaplanować daną rzecz do realizacji.** Na przykład kilka dni temu przeczytałem o nowym języku programowania, który mnie zaciekał. Stwierdzam, że to może być dobra inwestycja. Stawiam sobie jakiś cel do zrealizowania – na przykład napisanie określonego programu w tym języku programowania. Istotne jest zaplanowanie pierwszego zadania, pierwszego kroku, który doprowadzi nas do osiągnięcia zamierzonego celu.
- » **Zapomnieć, że chciałem zrealizować taką rzecz.** Na przykład usłyszałem na przyjęciu u znajomych, że pewna osoba była na wycieczce, na której skoczyła ze spadochronem oraz brała udział w spływie kajakowym. W momencie kiedy o tym słuchałem, obydwie te rzeczy wydały mi się ciekawym pomysłem do spróbowania, więc je zapisałem. Jednak teraz kiedy

o tym myślę, przypomniałem sobie, że mam ogromny lęk wysokości i nie chcę skakać ze spadochronem – więc wykreślam ten pomysł. Spływ kajakowy mnie ciekawi, więc zaplanuję tę rzecz do zrobienia.

- » **Oznaczyć tę rzecz do zrealizowania na kiedyś w przyszłość.** Co do zasady, odkładanie rzeczy na później nie jest dobrym pomysłem. Jednak w tym przypadku warto odłożyć sobie listę rzeczy, które chcielibyśmy zrobić w przyszłości. Może je zrobię, a może nie – tego nie wiem. Wiem, że potrzebuję więcej czasu do zastanowienia, czy chcę zrealizować to marzenie czy nie.

Często też konsultuję swoje pomysły z ludźmi, którymi się otaczam. Spojrzenie na pewne sprawy z innej perspektywy często postrafi zmienić nastawienie na rozmaite życiowe cele. Przez to można podjąć inną decyzję. Konsultowanie decyzji, co do których ma się wątpliwości, z innymi ma też drugie dno. Często stajemy się podobni do ludzi, którymi się otaczamy.

System, w którym prowadzę notatkę ze swoimi celami i marzeniami, stosuję już kilku lat. Obserwuję, że jestem w stanie zrealizować mniej więcej co piąte swoje marzenie. Nie przejmuję się tym, że 80% naszych marzeń idzie do kosza. Ważne jest to, żeby realizować 20% z nich, tych, z których będziemy naprawdę dumni.

Kiedy myślę o tym, co było kiedyś, dochodzę do wniosku, że nie realizuję swoich marzeń z dwóch powodów:

- » Na horyzoncie pojawiają się ważniejsze rzeczy do zrealizowania.
- » Realizacja tego celu wymaga więcej czasu niż przewidziałem.

Dzięki temu, że planuję, co chcę w życiu osiągnąć, mogę świadomie podjąć decyzję o tym, co zrealizuję, a z czym muszę się z bólem serca rozstać. Dzięki temu nie pozostawiam mojego życia przypadkowi, zwłaszcza kiedy czasu mam niewiele.

Kariera

Czy zastanawiamy się nad tym, jak potoczy się nasza kariera zawodowa za pięć czy za dziesięć lat? Wiele osób o tym nie myśli lub myśli od czasu do czasu, jednak nie planuje swojej kariery w branży IT. Dlaczego by mieli to robić? Programiści, inżynierowie oprogramowania lub inne osoby z branży IT zazwyczaj dobrze zarabiają, co trochę może uśpić naszą czujność. Jednak wkładając niewiele wysiłku w planowanie naszej kariery, możemy osiągnąć dużo więcej. Nie możemy zakładać, że skoro dzisiaj jest dobrze, to i jutro też będzie podobnie. Już samo odpowiedzenie sobie na pytanie, czy chcę zostać ekspertem w jakiejś dziedzinie i zajmować się tylko nią, czy chcę mieć uniwersalny wachlarz doświadczeń, czy może przejść na ścieżkę lidera technicznego, może dużo nam powiedzieć o tym, co powinniśmy robić. Bardzo ważne jest to, żeby wieść, co się chce robić, jeżeli mamy bardzo mało czasu.

Przed założeniem rodziny miałem dużo czasu na rozwój. Czas znajdował się na wszystko. Na pisanie bloga, organizowanie konferencji, występowanie publiczne, co tydzień chodzenie na meetup, wieczorami robienie kursów specjalistycznych na coursera. Teraz po założeniu rodziny, mając mniejszą ilość czasu do dyspozycji, musimy podejmować trudne decyzje. Musimy mieć przemyślane to, z czego chcemy zrezygnować. Nie można mieć wszystkiego i w pewnym sensie trzeba myśleć i planować swoją karierę w ścisłe określonym kierunku. Dlaczego to jest takie ważne? Wszystko rozbija się o czas.

ZAGIĄĆ CZAS

Skoro wiemy już, że ważne jest określenie tego, co chcemy osiągnąć, trzeba przystąpić do realizacji postanowionych rzeczy w tak ograniczonym czasie.

Organizacja czasu: kalendarz

Jeżeli chodzi o organizację czasu, to większym przełomem niż kalendarz personalny jest tylko współdzielony z partnerką/partnerem kalendarz cyfrowy. Jest to narzędzie, które odpowiednio użyte pozwala lepiej wykorzystać czas. Wszystkie istotne wydarzenia z życia rodziny muszą być zapisane w kalendarzu. Wspólne planowanie tygodnia pozwala pracować trochę jak maszyna. Dzięki temu nie tracimy czasu na ciągłe zastanawianie się – kto teraz zajmuje się dzieckiem – ale też lepiej możemy zorganizować swoje zadania, tak aby życie rodzinne przebiegało bez zakłóceń. Na przykład zaplanowaliśmy, że w najbliższy czwartek zajmuje się dzieckiem od godziny 16:30 do końca dnia. Wiem też, że do piątku muszę wysłać list do urzędu. Planuję tydzień w taki sposób, by mieć zaadresowany i przygotowany list dzień wcześniej. Żeby przy okazji wspólnego spaceru z dzieckiem tylko wskoczyć na chwilę na pocztę i go wysłać. Gdybym nie planował kalendarza wcześniej i ustalił to z żoną w środę rano, nie miałbym czasu odpowiednio przygotować listu. Wieczorem nie miałbym sposobności, żeby przygotować odpowiednie pismo, jednocześnie zajmując się dzieckiem, przez co musiałbym to zrobić w piątek, a tym samym stracić trochę czasu.

Organizacja czasu: lista zadań

Organizację dnia prawie zawsze zaczynam od listy zadań na dany dzień, sporządzając ją dnia poprzedniego. Moja lista zadań jest podzielona na dwie sekcje: zadania dotyczące pracy oraz zadania dnia codziennego. Wszystko to mam zapisane na papierze – w moim organizerze. Dlaczego na papierze? Papier przyjmie wszystko, jest najszybszy do edycji, a ponadto jest miłą odmianą i chwilą oddechu od komputera. Oczywiście pod koniec dnia aktualizuję swoje główne zadania i przypomnienia w postaci cyfrowej.

Zadania dotyczące pracy są zazwyczaj krótkie do wykonania. Nie trwają dłużej niż 30-45 minut. Natomiast bardzo często się zdarza, że wykonanie jednego tworzy kolejne. Na przykład implementacja jakiejś zmiany rodzi potrzebę refaktoryzacji jakiegoś fragmentu kodu.

Zadania dotyczące życia dnia codziennego zajmują porównywalnie tyle samo czasu i również generują nowe podzadania. Na przykład po przeczytaniu artykułu lub zrobieniu rozdziału zadań w kursie online powoduje automatycznie dużą ilość nowych pomysłów, które wydają mi się interesujące. Stosuję tę samą taktykę co do celów i marzeń. Zapisuję je na bok – na kartce, ale pod koniec dnia po chwili refleksji mogę uznać, że były to rzeczy niewarte zainteresowania i wybieram tylko te najlepsze i najciekawsze zajęcia.

Takie podejście do spraw powoduje, że wchodzę i pracuję na wysokich obrotach. Kiedy kończę dzień i pomyślę o tym, co robiłem rano, uświadamiam sobie, jak dużo rzeczy zrobiłem. Początkowo miałem problemy z zadaniami, które rozwijały moją kreatywność, robieniem eksperymentów z różnymi nowymi narzędziami i tym podobnymi zadaniami. Z czasem jednak zdałem sobie spra-

wę, że postawienie sobie konkretnych zadań przybliża mnie szybciej do osiągnięcia tego, czego chciałem się nauczyć lub sprawdzić, ponieważ nie traciłem czasu na zbędne rozpraszczenia, a moja praca nad tematem była dogłębna i wnikiwała.

Organizacja czasu: zasada przystankowa

Moim ulubionym środkiem transportu w mieście w godzinach szczytu jest komunikacja miejska. Do pracy jeżdżę autobusem, który na swojej drodze ma 7 przystanków. Podróż autobusem zajmuje mi 14 minut, mniej więcej po 2 minuty pomiędzy przystankami.

Na większość maili i wiadomości, które spływają do mnie, odpowiadam właśnie w autobusie. Dlaczego? 2 minuty to idealny czas na odpisanie na wiadomość e-mail. Ponadto dwie minuty wystarczą na załatwienie większości spraw dnia codziennego takich jak umówienie się z kimś na spotkanie, znalezienie wolnego terminu u dentysty lub zamówienie jakiegoś produktu przez Internet. Na początku słabo mi to szło, ale z każdym kolejnym dniem montowałem na telefonie dedykowane aplikacje do obsługi szczególnych przypadków – do obsługi banku lub umawiania wizyty u lekarza i tym podobne. Motywatem było dla mnie to, by załatwić sprawę w ciągu tych dwóch minut, czyli przed następnym przystankiem autobusu. Oczywiście ważne jest tutaj poznanie każdej funkcji i wielu aplikacji w swoim telefonie, jednak po pewnym czasie inwestycja jest bezcenna, jeżeli chodzi o oszczędzony czas.

Kiedy opanowałem do perfekcji sztukę robienia rzeczy pożytecznych w autobusie – zamiast gapić się na portal społecznościowy, zauważałem, że zmieniło się moje podejście do spraw i obowiązków dnia codziennego. Jeżeli jestem w stanie zrobić coś w czasie dwóch minut, nie zastanawiam się, tylko to robię, żeby mieć temat z głowy. Dzięki temu mam więcej czasu na to, co naprawdę jest ważne, i na to, co powiniensem robić.

Zasada przystankowa nauczyła mnie innej ważnej rzeczy. Dziel swoje zadania na mniejsze i bądź zawsze gotowy na ich realizację. Na przykład chcę zrobić kurs online. Wiem, że potrzebuję na niego 5 godziny tygodniowo. Tak naprawdę wiem, że jak usiadłbym w sobotę rano, to zrobiłbym to zadanie w 4 godziny. Czy tak podchodzi do tego problemu? Absolutnie nie. Wiem, że ciężko będzie mi znaleźć cztery godziny w sobotę, żeby skupić się na jednym zadaniu. Dlatego wykorzystuję każdą chwilę, którą mam. Rano, wieczorem, w autobusie do pracy – wszędzie gdzie tylko zobaczę, że mam kilkanaście minut wolnego czasu na zrobienie małego kroku do przodu – robię go. Małymi kroczkami jestem w stanie osiągnąć wielkie rzeczy.

Organizacja czasu: dbaj o przerwy

„Odpocznij. Pole, które odpoczęło, daje obfite plony.” – tak mawiał Owidiusz oraz mój tata. I to jest prawda. Jeżeli przez kilka tygodni działały na najwyższych obrotach, przetwarzając nasze zadania z największą precyzją i wydajnością, po kilku tygodniach musi przyjść czas na odpoczynek. Nawet jeżeli podczas naszego planu będziemy spędzać ten czas z rodziną i codziennie poświęcali chwilę czasu na odpoczynek, dobrym pomysłem jest zrobić kilkudniową przerwę.

Warto wtedy wyjechać gdzieś z rodziną. Nie musi to być daleko, nie musi być na dłucho. Po zrobieniu sobie krótkiej, a nawet weekendowej przerwy od tempa życia dnia codziennego nabieramy świeżości i energii do kolejnego maratonu.

Organizacja czasu: czas jest wszędzie

Bycie rodzicem zmusiło mnie do odkrycia niestandardowych pór dnia i nocy do pracy. Kiedy dziecko było jeszcze małe, często budziło się pomiędzy trzecią a czwartą nad ranem. Sytuacja zazwyczaj szybko była opanowana, ale co wtedy robić? Wrócić i iść spać dalej, żeby wstać za dwie godziny normalnie do pracy? Absolutnie nie! Wykorzystać ten czas i to, że jesteśmy na nogach. O godzinie 4 nad ranem możemy się skupić tylko i wyłącznie nad tym, co chcemy robić. Nikt nam nie przeszkadza, nie jesteśmy niczym rozproszeni. Możemy po prostu działać. Co więcej, jeżeli nie mamy ustalonych godzin pracy, możemy zacząć pracować, żeby tę pracę wcześniej skończyć. Zdziwienie kolegów z firmy jest ogromne, kiedy przychodzą na 9:00 do pracy i widzą same pull requesty do przejrzenia.

Nawet dzisiaj, kiedy nasze dziecko już nie budzi nas rano, nastawienie budzika godzinę wcześniej przed normalną porą wstawania daje ogromny wachlarz możliwości. To jest siedem godzin w tygodniu. Tak jakbyśmy nagle odkryli jeden dodatkowy dzień w tygodniu. Oczywiście nie należy przesadzać ze zbyt wcześnieym wstawaniem rano, ponieważ nie damy rady dotrwać do końca dnia. Każdy z nas musi znaleźć swój własny balans.

Więcej czasu: blokuj rozpraszacze

Przez kilka miesięcy stosowałam aplikację o nazwie RescueTime. Jest to aplikacja natywna instalowana na naszym komputerze. Program ten obserwuje, na jakie strony internetowe wchodzę, jakie aplikacje mam otwarte oraz jak długo. Na koniec dnia aplikacja zawsze pokazywała, ile czasu poświęciłem na pracę, na czytanie wiadomości, a ile na rozrywkę. Robiła to całkowicie automatycznie, bez mojej żadnej ingerencji.

Pierwsze dni jej używania były tragiczne dla mojego samopoczucia. Ogromne ilości czasu liczone w godzinach były przeze mnie marnowane na byle co i różnego rodzaju rozpraszacze. Co robisz, kiedy odpalasz testy, które trwają 3 minuty? Wchodzisz i czytasz wiadomości, czy robisz coś pozytecznego z tym czasem? Czy 3 minuty wystarczą na to, by załatwić inną sprawę? Jeżeli tak – zrób to.

Niestety ciężko było mi zmienić swoje nawyki. Musiałem sięgnąć po drastyczniejsze metody, takie jak blokowanie stron internetowych, na które tracimy najwięcej czasu. Było to lekarstwo, którego potrzebowałem. Dzięki temu odzyskałem dużą część czasu.

Więcej czasu: delegacja prostych zadań

Czy zyskamy więcej czasu, jeżeli będziemy robili zakupy przez Internet? Czy zyskamy więcej czasu, jeżeli będziemy korzystać

z pomocy osoby do sprzątania w naszym domu? Odpowiedź brzmi: tak. Jednak zanim zdecydujemy się na zakupy online lub na sprzątaczkę, adoptujmy metodę Scrum do naszego rytmu dnia codziennego.

Co zrobić, żeby mniej czasu tracić na sprzątanie? Do problemu możemy podejść iteracyjnie. Chodzi nam o minimalizację czasu, więc z każdą kolejną iteracją powinniśmy mierzyć czas. Po każdej iteracji sprzątania – powiedzmy raz na tydzień – przychodzi czas na retrospekcję – czyli zastanowienie się, co moglibyśmy zmienić, żeby sprzątanie zajęło nam mniej czasu. Po kilku zmianach organizacyjnych w metodzie i sposobie sprzątania można dojść do niesamowitych rezultatów.

Początkowo sprzątanie łazienki zajmowało mi godzinę. Płyн do mycia zawsze miałem w innym schowku, nie mogłem znaleźć odpowiednich szmatek, często też w samej łazience było dużo rzeczy, które przewalałem z kąta w kąt. Po kilku iteracjach kompleksowe mycie łazienki zajmuje mi mniej niż 15 minut. Wszystko dzięki lepszej organizacji, dobremu przygotowaniu i zaplanowanemu działaniu.

DZISIAJ

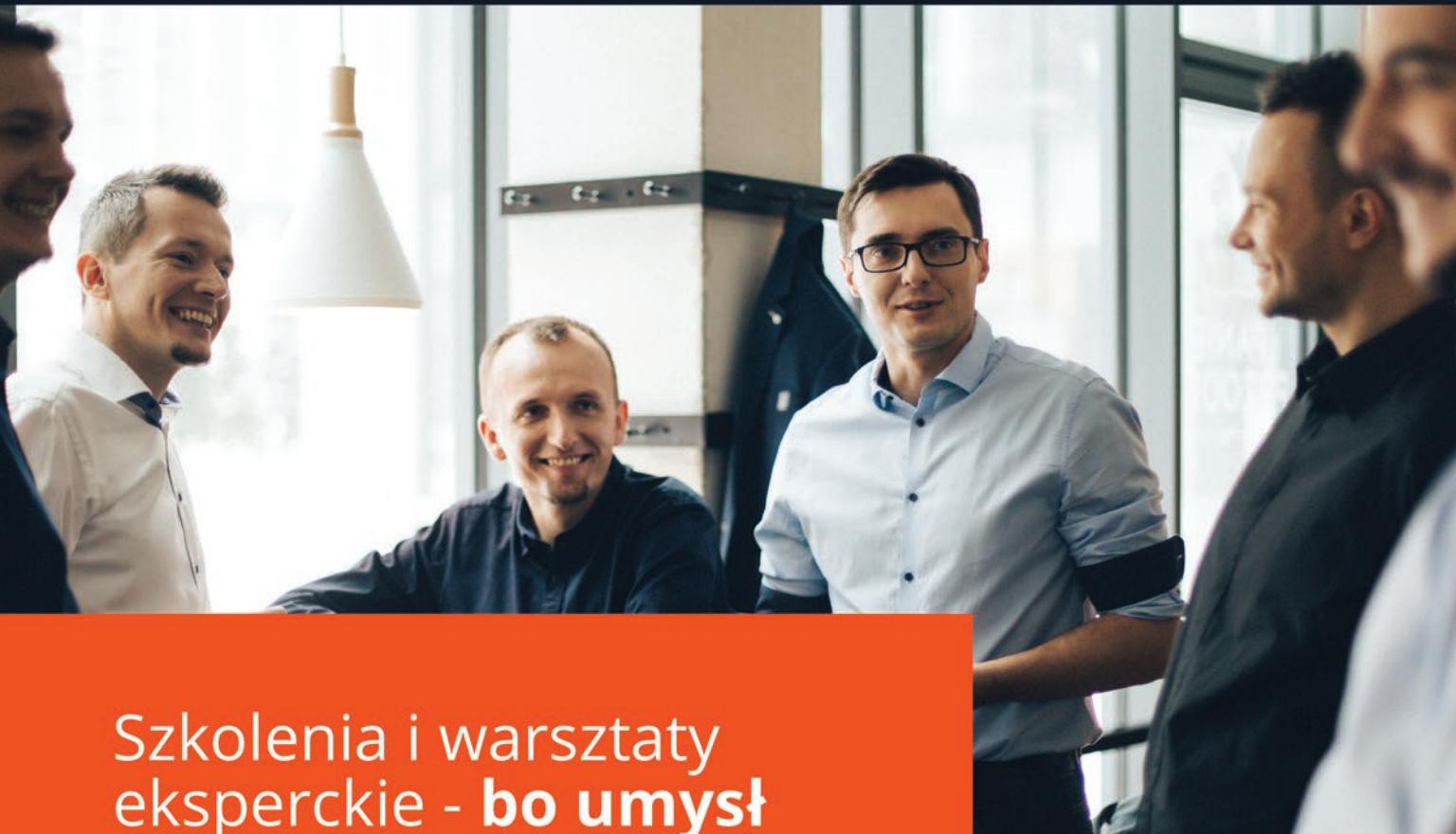
Kiedy spotykam kolegów, z którymi rozmawiałem o wychowywaniu dzieci i pracowaniu pełną parą, dostrzegam na ich twarzach wyrazy zadowolenia i radości z życia, jakie czerpię z tego, że otaczają się najbliższymi, a mimo to skutecznie udaje im się rozwijać i pracować na wysokich obrotach. Jednak życie pokazuje też od czasu do czasu na ich twarzach oznaki zmęczenia po ciężkiej nocy lub widać, jak po południu po ciężkim dniu są już lekko zmęczeni i najchętniej udaliby się na krótką drzemkę. Dzień zaplanowany pod korek czasami może się rozsypać.

Na sam koniec trzeba zadać sobie pytanie: Czy warto? Czy warto zmienić swoje życie na model zadaniowy i pracować jak maszyna? Niewątpliwie efektywność, a zarazem oszczędność czasu, jaką możemy dzięki temu uzyskać, daje nam możliwość spędzenia tego czasu albo z rodziną, albo oddając się naszej pasji, jaką jest programowanie. Niestety przez większość czasu tracimy lekkość i swobodę w podejmowaniu działań z dnia na dzień, dlatego tak ważne jest robienie sobie przerw od ściśle ułożonego planu. Swoboda i spontaniczność z pewnością obudzą w nas kreatywność.

Wystrzegajmy się tylko jednego. Nie traktujmy rodziny i dzieci jak obiekty, które trzeba obsłużyć. Dzieci to nie przedmioty, które trzeba: ubrać, nakarmić, wysłać do przedszkola, umyć i położyć spać. Nie są to kolejne pozycje na naszej liście zadań. Projekt pod tytułem „dzieci” wymaga dużo miłości. Tylko wtedy nieco już zmęczony późnym wieczorem mogę usiąść do czytania książki lub do programowania. Daje mi to energię na kolejny wspaniały dzień.

MICHAŁ LEWANDOWSKI

Software developer. Przez ostatnie kilka lat zajmował się różnymi rzeczami związanymi z wytwarzaniem oprogramowania, począwszy od analizy wymagań, przez prowadzenie zespołu, samo kodowanie, skończywszy na utrzymaniu i odpalaniu systemu. Jest przekonany, że każdy profesjonalista, oprócz twardych umiejętności związanych z kodowaniem, powinien mieć rozwinięte umiejętności miękkie.



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



Dołącz do Ericsson i mają wpływ na to,
w którym kierunku zmierza postęp
technologiczny. Nasze zespoły biorą udział
w prestiżowych projektach ICT, które
pozwala nam wszystkim swobodniej
pracować, studiować i żyć w zrównoważonych
społecznościach na całym świecie.

Obecnie ponad 40% światowego ruchu
mobilnego przechodzi przez sieci firmy
Ericsson, z których korzysta ponad
2,5 miliarda abonentów.

Aplikuj już dziś!

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



Zmieniaj
z nami
świat