Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Daniel Gutowski
Nr albumu: 372207

Jakub Kuklis
Nr albumu: 371125

Adrian Naruszko
Nr albumu: 371233

Filip Plata
Nr albumu: 371335

Modernizacja architektury autorskiego CMS firmy E-point

Praca licencjacka
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
mgr. Krzysztofa Wojciecha Ciebiery
Instytut Informatyki

Kwiecień 2018

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpisy autorów pracy

Streszczenie

W pracy przedstawiamy opis współpracy w firmą e-point. Produktem nad którym pracowaliśmy, jest autorski system CMS. W momencie rozpoczęcia współpracy miał on strukturę monolitu. Naszym zadaniem było napisanie mikroserwisów umożliwiających dalszą modernizację i modularyzację tego produktu. Ostatecznie zajmowaliśmy się komponentami odpowiadającymi za dostęp do bazy danych oraz rendering.

Słowa kluczowe

CMS, Java, ORM, rendering, modernizacja, Jooq, Gatling, Caffeine, cache

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.127. Blabalgorithms
D.127.6. Numerical blabalysis
??TODO

Spis treści

Wprowadzenie

Z biegiem lat technologia rozwija się coraz szybiej. Stwierdzenie to sprawdza się od wieków, ale jego implikacje stają się znacznie poważniejsze w miarę rozwoju cywilzacji. Powszechne wcześniej rozwiązania i reguły prędko stają się przestarzałe i wymagają renowacji, co często oznacza całkowitą zmianę podejścia czy redefinicję problemu.

Nie inaczej jest w przypadku usług sieciowych, szczególnie żywej dziedziny w czasach powszechnego Internetu i postępującej globalizacji. Świadczy o tym chociażby mnogość frameworków i technik wykorzystywanych przy ich tworzeniu, nieustannie ewoluujących w miarę odkrywania wad w używanych wcześniej wersjach. Przed problemem modernizacji swojego systemu stanęła m.in. firma E-point, które to zadanie powierzyła w części naszemu zespołowi.

W szczególności, zajmowaliśmy się implementacją dwóch mikroserwisów:

1. DataAPI - RESTowa aplikacja integrująca bazę danych aktualnego CMSa z nowym mechanizmem renderingu.

2. Element odpowiedzialny za rendering serwisów internetowych i związana z nim architektura.

Rozdział 1

Zarys sytuacji i problemu

1.1. Firma E-point

E-point to polskie przedsiębiorstwo zajmujące się tworzeniem oprogramowania, dostarczające systemy internetowe dla międzynarodowych korporacji oraz usługi w modelu SaaS (software as a service). Założone w 1996 roku, działa nieprzerwanie od ponad 20 lat, otrzymując za swoją działalność liczne nagrody i wyróżnienia od przedstawicieli branży technologicznej. Jej klientami są największe firmy polskie i zagraniczne, takie jak ING Bank Śląski, Bank BGŻ BNP Paribas czy Amway.

Firma koncentruje się na następujących sektorach:

- systemy e-business – aplikacje mające na celu sprzedaż produktów i usług oraz zapewnienie obsługi klientów i partnerów przez internet; sklepy internetowe, bankowość elektroniczna, eCRM, systemy obsługi zamówień;
- portale korporacyjne – systemy mające na celu udostępnianie informacji na zewnątrz i wewnątrz organizacji – serwisy WWW, portale wewnętrzne, extranety;
- aplikacje mobilne aplikacje i serwisy na smartfony i tablety;
- systemy e-wniosków – rozwiązania zorganizowane wokół zagadnienia zbierania i przetwarzania wniosków (formularzy elektronicznych);
- systemy mające na celu obsługę wybranych procesów biznesowych w organizacji – rejestry, wsparcie sprzedaży, operacyjne CRM, e-learning.

W czasie tylu lat istnienia E-point wypuścił na rynek wiele swoich własnych produktów, m.in. kreator stron mobilnych ActiveMobi, modeler baz danych Vertabelo czy OneWebSQL, pierwsze polskie narzędzie do mapowania obiektowo-relacyjnego – sposobu odwzrowoania obiektowej architektury systemu informatycznego na bazę danych o relacyjnym charakterze. Nasze zadanie związane były z wewnętrznym CMS, używanym do zarządzania danymi i renderowania odpowiednich treści w serwisach internetowych klientów firmy.

1.2. ACN

ACN - wewnętrzny CMS (Content Management System) - to jeden z największych produktów firmy. Odpowiada za cały proces utrzymywania, przygotowania i serwowania danych serwisom internetowym. W momencie otrzymania zapytania load balancer decyduje o przekierowaniu go do odpowiednio odciążonego serwera. Request trafia do ACN, który sprawdza uprawnienia dostępu, a po ich pomyślnej weryfikacji wyciąga dostępne komponenty z cache'u, a o informacje potrzebne pozostałym odpytuje bazę danych. W dalszej kolejności renderowane są niecache'owalne elementy strony oraz te nieobecne w danym momencie w cache'u. Po wygenerowaniu wszystkich potrzebnych komponentów strona jest serwowana użytkownikowi.

[schemat ACN]

1.3. Problemy obecnej architektury

Dotychczas ACN stanowił monolit. Jest to produkt tworzony od wielu lat, stale rozwijany i rosnący. W samej bazie danych część zmian podyktowana była indywidualnymi wymaganiami klientów, przez co stała się ona mało przejrzysta i trudna w utrzymaniu. Duża wielkość systemu sprawia, że jego modyfikacja jest uciążliwa – zachowanie spójności wszystkich usług przy nawet pozornie małych zmianach wymaga wielu godzin pracy nad kodem. Utrudniona jest także lokalizacja błędów w systemie, brak modułowości oznacza konieczność przeszukiwania znacznie większej ilości kodu w celu wykrycia usterki.

W związku z powyższym, firma postanowiła dostosować swój produkt do bardziej nowoczesnego podejścia do problemu. Tak pojawił się pomysł przynajmniej częściowego wydzielenia funkcjonalności ACN do osobnych mikroserwisów. Implementacja dwóch z nich, elementu łączącego aplikację z bazą danych oraz serwisu renderującego komponenty strony, została przydzielona naszemu zespołowi.

Rozdział 2

Koncepcja

2.1. Początek prac – inny projekt

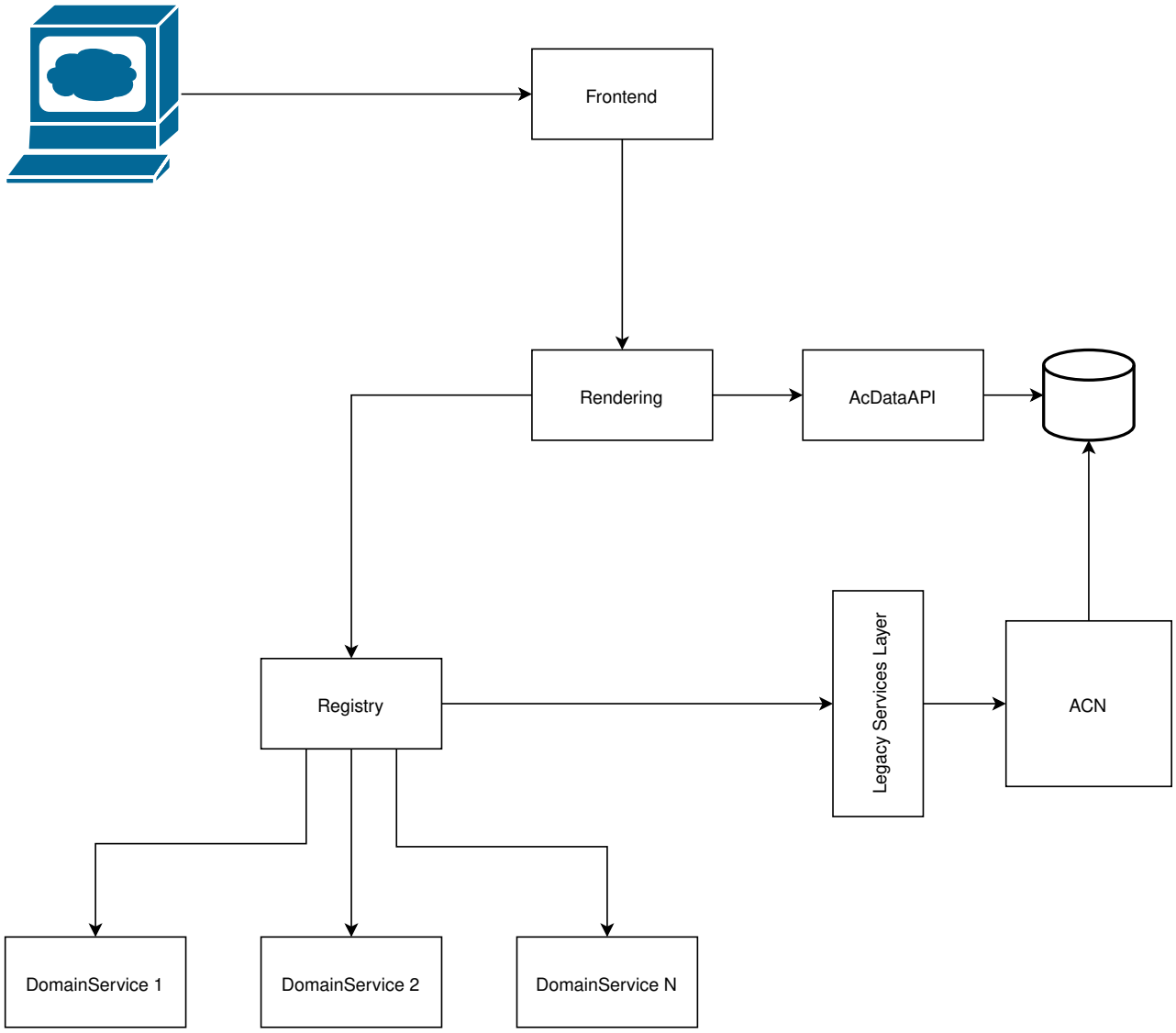
Założenia naszego projektu nie były od początku jasne. Pierwotnie mieliśmy zająć się w zasadzie innym zadaniem – zaimplementowaniem personalizacji treści serwisów internetowych na podstawie aktywności ich użytkowników. Miało to pozwolić przede wszystkim na bardziej trafne wyświetlanie ofert i reklam. Po wstępnej konfiguracji dostępów i tunelowania zdążyliśmy stworzyć podstawową architekturę aplikacji i skonfigurować narzędzie deweloperskie na AWS, kiedy okazało się, że rozwój produktu zostaje wstrzymany ze względu na problemy z jego sprzedażą. Wtedy też wyklarowało się, że nasz projekt będzie w całości poświęcony modernizacji aktualnie funkcjonującego systemu.

2.2. Ostateczny projekt

Rozwiązanie zaproponowane i wdrażane przez firmę ma opierać się na stylu programowania REST, tj. wykorzystującym predefiniowane bezstanowe operacje, co pozwala osiągać dużą wydajność, modułowość i niezawodność aplikacji. Docelowo pojawić mają się następujące mikroserwisy:

- frontend - przyjmowanie zapytań i serwowanie stron WWW;
- rendering - tworzenie stron na podstawie szablonów, wykorzystuje usługi udostępniane przy pomocy serwisu Registry;
- AC Data API - abstrakcja bazy danych, udostępnia tylko niezbędne renderingowi dane;
- registry - rejestr wewnętrznych i zewnętrznych usług takich jak podmiana tagów, personalizacja treści;
- ACN - dotychczasowa aplikacja implementująca niektóre usługi, w końcowym etapie ma zniknąć;

Nasza grupa zajmowała się elementami Data API oraz serwisem renderującym. W przypadku tego drugiego, część pracy stanowiło wprowadzenie zmian w samym ACN, związanych z dostosowywaniem narzędzia do nowej koncepcji schematu renderowania.



Rysunek 2.1: Schemat wydzielonych aplikacji

Rozdział 3

AC Data API

3.1. Mikroserwis

Obsługa serwisów internetowych wymaga niezawodnego i bezpiecznego sposobu łączenia z bazą danych, stąd pojawia się potrzeba wydzielenia odpowiednigo mikroserwisu odpowiedzialnego za to zadanie. E-point korzysta z tradycyjnych relacyjnych baz danych, co pociąga za sobą stosowanie ORM przy wyciąganiu danych do dalszego przetwarzania.

Aplikacja Data API ma udostępniać dane z bazy danych ACN dla komponentów w nowej architekturze. Będzie to jeden z najczęściej odpytywanych serwisów, w związku z czym jego wydajność oraz dostępność jest krytyczna dla sprawnego i poprawnego działania całej usługi.

3.2. Założenia

Aplikacja docelowo ma specyfikować oraz implementować kontrakt obslugi REST'owych endpointów, których nazwy składają się z wersji API oraz tytułu zapytania (np. /v1/languageversion). Odpowiedzią na dane żądanie jest JSON zawierający odpowiednie dane z bazy.

Dane z bazy danych mają być zapisywane w odpowiednio inwalidowanym cache'u. Powodem inwalidacji może być upływ wyznaczonego czasu od zapisu czy powiadomienie o zmianie z innego serwisu połączonego z bazą danych (na ten moment funkcję tę pełni ACN).

API ma być dokładnie pokryte testami, zarówno poprawnościowymi, jak i wydajnościowymi. W pierwszej wersji nie jest wymagana żadna forma uwierzytelniania.

3.3. Struktura bazy

Baza danych składa się z wielu schematów. Głównym schematem jest schemat 'master'. Przechowuje on informacje o klientach – organizacjach – korzystających z systemu firmy E-point. Dla każdego klienta tworzony jest schemat indentyfikowany jego kodem, zawierający np. informacje o użytkownikach zarejestrowanych w serwisach tegoż klienta. Dla każdego portalu zarządzanego przez organizację tworzony jest osobny schemat, w którym przechowywane są wszystkie dane dotyczące instancji portalu.

3.4. Użyte technologie

Aplikacja została napisana w Javie w wersji 8 przy użyciu platformy Spring. Ostatecznie przy jej tworzeniu korzystaliśmy m.in. z:

- systemu automatycznego budowania Gradle;
- pakietu Jooq (Java Object Oriented Querying), lekkiej biblioteki stworzonej pod ORM;
- biblioteki Caffeine, służącej do cache'owania;
- produktu Logstash i powiązanych z nim narzędzi takich jak Elasticsearch czy Kibana, służących do monitorowania działania aplikacji;
- platformy Docker z postgresem do testowego deploymentu aplikacji;
- frameworku Gatling, stosowanego przy tworzeniu i wykonywaniu scenariuszy testów wydajnościowych;
- funkcjonalności związanych z IntelliJ IDEA.

Rozdział 4

Praca nad AC Data API

4.1. Iteracja nr 1

W pierwszej iteracji do generowania klas Javy na podstawie schematów z bazy danych użyliśmy pakietu Jooq. Zaprezentowany przez nas przykład rozwiązania oparty bardzo bezpośrednio na Jooqu został odrzucony przez naszych przełożonych ze względu na ujawnione braki w architekturze rozwiązania. Problematyczne okazało się m.in. zapewnienie ścisłego, sztywnego kontraktu przez API przy zmianach schematów w bazie danych.

Ujawnione wady wcześniejszej koncepcji spowodowały przestój w pracach związany z dyskusją nad możliwościami rozwiązania wykrytych problemów. Po dogłębnej analizie dostępnych narzędzi i prezentowanych przez nas demonstracyjnych kawałków kodu nasz przełożony zdecydował się na pozostaniu przy pakiecie Jooq, z tą różnicą, że warstwa aplikacji zależna od Jooq miała zostać wydzielona od pozostałej części kodu, niezależnej od sposobu dostępu do bazy danych.

W tej wersji do monitorowania aktywnych instancji aplikacji użyliśmy gotowego, dedykowanego narzędzia spring-boot-admin. Niestety, konfiguracja narzędzia okazała się tyleż prosta, co ograniczona. Nie mogąc spełnić wymogów firmy przy jego pomocy, zdecydowaliśmy się na inny produkt do monitorowania aplikacji.

4.2. Iteracja nr 2

Ta iteracja poświęcona była przede wszystkim na poprawne zaimplementowanie ORM oraz stworzenie potrzebnych endpointów. Przyjęliśmy architekturę podzieloną na kilka warstw odpowiedzialnych za wydzielone zadania.

Najniższa z nich zapewnia bezpośredni dostęp do bazy danych poprzez Jooq. Druga warstwa zajmuje się dzieleniem danych klientów na schematy w bazie, zarządzaniem aktualnymi schematami. Następny poziom ukrywa działanie Jooq'a poprzez utworzenie obiektów domenowych. Ostatecznie wersjonowanie API poprzez javowe pakietowanie umożliwia zapewnienie kontraktu kompatybilnego wstecz nawet pomimo zmian schematu bazy daych.

Odpowiednie endpointy realizują zapytania potrzebne wyższym poziomom obsługi żądań do stron. Listę koniecznych na ten moment zapytań przygotowali nasi przełożeni. Wśród zaimplentowanych endpointów znajdują się m.in. zapytania pobierające wersje językowe, informacje o kliencie o zadanym identyfikatorze czy własności strony o zadanym id. Wszystkie te metody są obłożone testami poprawnościowymi w JUnit.

4.3. Iteracja nr 3

Mając zaimplementowaną strukturę aplikacji, przystąpiliśmy do zadań związanych z jej wydajnością. Pierwszym krokiem była konfiguracja cache'u. Na podstawie analizy dostępnych w internecie źródeł i benchmarków zdecydowaliśmy się na Caffeine, bardzo wydajną bibliotekę do cache'owania oparta na Javie 8.

Spring dobrze współdziała z różnymi frameworkami i Caffeine nie jest tu wyjątkiem. Wybór elementów do cache'owania odbywa się poprzez dodanie odpowiednich adnotacji przy cache'owanych funkcjach – model zmian w kodzie przy pomocy adnotacji to zresztą powszechny w Springu mechanizm.

Gotowy cache poddaliśmy testom wydajnościowym generowanym przy pomocy scenariuszy korzystających z Gatlinga.

Działanie całej aplikacji możemy monitorować, korzystając z Elastic Stacka. Wymaga on wprawdzie ręcznego wprowadzania reguł dla oczekiwanych statystyk i nie radzi sobie najlepiej z agregowanymi danymi, zapewnia za to pełną konfigurowalność. Narzędzia związane z Elastic Stackiem takie jak Kibana pozwalają nam na graficzną reprezentację zebranych statystyk, poniżej przykładowe wynikowe wykresy.

[Wykresy z Kibany]

Rozdział 5

Rendering

5.1. Spotkanie

Po oddaniu ostatnich elementów Data API i przejściu przez etap recenzji odbyliśmy spotkanie w firmie dotyczące kontynuacji naszej współpracy. Przedstawiona została na nim wizja dalszego podziału ACN i wydzielenia z niego elementu renderującego.

5.2. Dotychczasowy rendering

W poprzedniej wersji renderowanie miało formę dwuetapową. W jednym z etapów renderowane były komponenty z ustawionym w bazie danych parametrem 'noncacheable attributes'. Elementy te miały swoje wersje w cache'u zależne od parametru żądania (zazwyczaj) czy też ciasteczka pochodzącego z sesji. Całość renderowania wykonywania była przez ACN, co ograniczało wydajność systemu.

W rozwiązaniu wdrażanym przez firmę opisany wyżej postprocessing ma docelowo zniknąć. Naszym zadaniem było takie przystosowanie ACN, które pozwoliłoby na pozostawienie części renderingu przeglądarce użytkownika. Elementem tej pracy było zastąpienie wspomnianego wcześniej podejścia przez coś lepszego.

[co lepszego? Na razie nie wiadomo]

Dopiero po wykonaniu powyższego zadania mogliśmy przystąpić do pisania nowego komponentu renderującego.

5.3. Iteracja nr 1

Pierwszym etapem naszej pracy była konfiguracja środowiska potrzebnego do testowania wprowadzanych przez nas zmian. Ostatecznie postanowiliśmy trzymać całe środowisko na maszynie wirtualnej. Do obsługi serwera używany jest Apache oraz JBoss. Dla usprawnienia pracy korzystamy także z JRebela, narzędzia pozwalającego na aktualizowanie kodu i obserowanie zmian bez restartowania serwera.

W tym czasie nasi przełożeni przygotowywali zmiany w komponentach powiązanych z ACN, takich jak OneWeb, by umożliwić nam pracę nad serwisem renderującym. Jedną z istotnych zmian było umożliwienie cache'owania formularzy. Nauka OneWeba była kolejną rzeczą, która zajmowała nas w tej części.

5.4. Iteracja nr 2

W tej iteracji naszym zadaniem było wyczyszczenie ACN z elementów uznanych przez naszych przełożonych za zbędne – bardzo rzadko używanych i stosunkowo łatwo podmienialnych lub niecache'owalnych. Wyspecyfikowanych zostało kilkadziesiąt niedużych komponentów wymagających poprawek.

[fragment kodu z opisem ?]

[co kto robił ?]

[bibliografia ?]