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

????

Praca licencjacka
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
mgr. Krzysztofa Wojciecha Ciebiery
Instytut Informatyki

Luty 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 rozwijanym przez tą firmę jest autorski system CMS. Na chwilę obecną ma on architekturę monolitu. Naszym zadaniem było napisanie mikroserwisów umożliwiających dalszą modernizację i modularyzację tego produktu.

Słowa kluczowe

blabaliza różnicowa, fetory σ-ρ, fooizm, blarbarucja, blaba, fetoryka, baleronik, ??TODO

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

Początkowo mieliśmy zrealizować inny projekt - kategoryzację użytkowników na podstawie aktywności na stronach internetowych. Zdążyliśmy zaimplementować podstawową architekturę, a także skonfigurować narzędzia deweloperskie na AWS. Niestety, nastąpiła zmiana projektu wymuszona okolicznościami zewnętrznymi na realizację pierwszego etapu modernizacji produktu e-pointu na nowe technologie oraz architekturę.

Zajmowaliśmy się implementacją dwóch mikroserwisów:

1. Element integrujący między bazą danych aktualnego CMSa a nowym mechanizmem renderingu. Jest to RESTowa aplikacja udostępniająca niektóre dane z bazy danych potrzebne modułowi renderingu.

2. Nowy mechanizm renderujący strony wraz z rejestrami usług. Gdzie przez usługi rozumiane są kroki/akcje wykonywane w trakcie tworzenia strony. Np. podmiana tagów, (??TODO). W docelowej architekturze mają to być niezależne mikroserwisy.

Rozdział 1

Opis modernizacji architektury

1.1. Dotychczasowy monolit

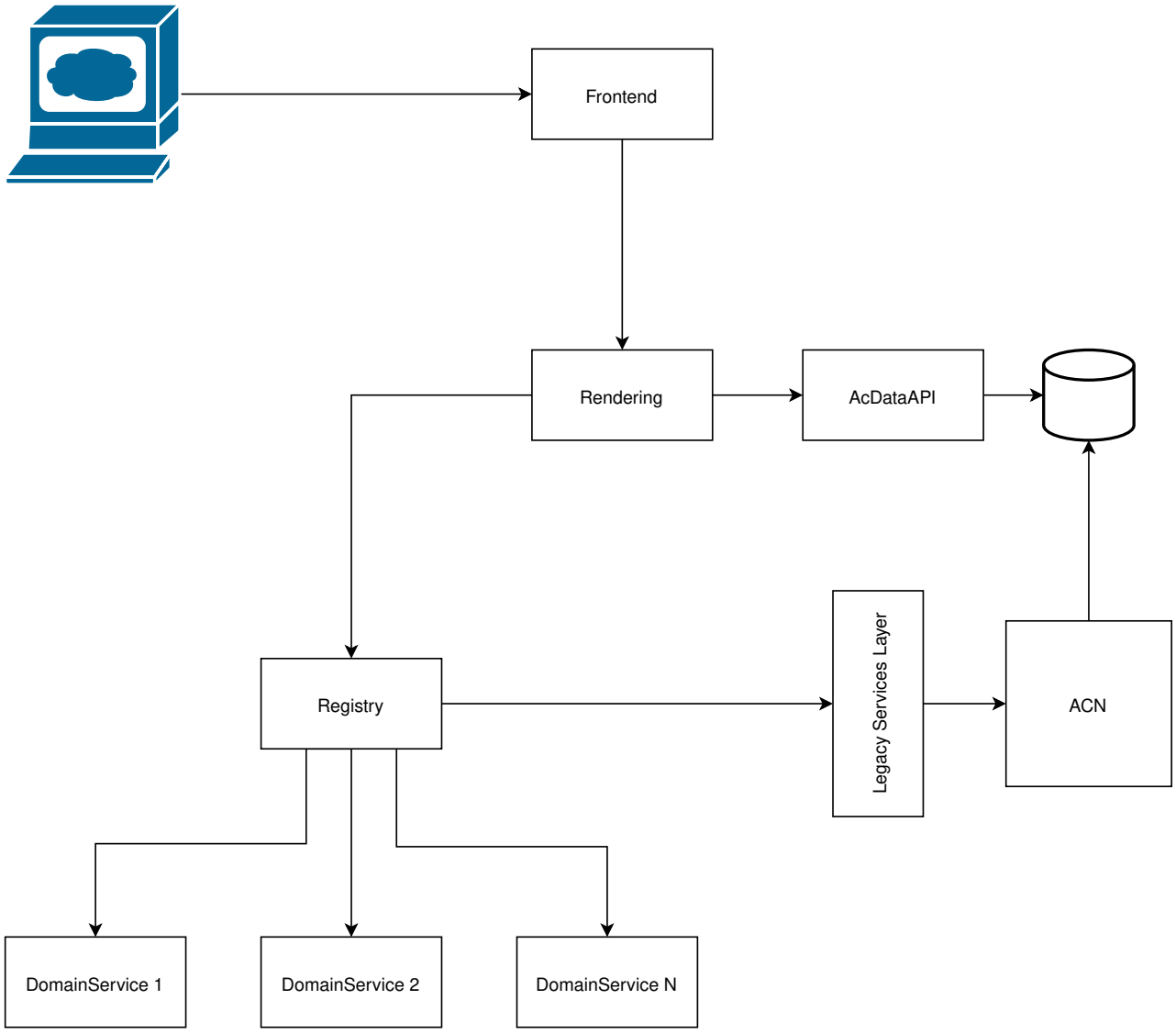
Do tej pory jedna aplikacja(a.k.a. ACN) obsługiwała zapytania HTTP od momentu pojawienia się do końcowego serwowania strony WWW. Taka struktura powoduje skomplikowanie kodu, utrudnia wprowadzanie zmian a w przypadku awarii serwisu komplikuje lokalizację błędu.

1.2. Nowa architektura

Rozwiązanie zaproponowane przez firmę i wdrażane przez nas to rozdzielenie czynności wykonywanych w ACN na kilka mikroserwisów RESTowych. Jak widać na rysunku 1.1 wydzielone zostały następujące aplikacje:

- Frontend - przyjmowanie zapytań i serwowanie stron WWW
- Rendering - tworzenie stron na podstawie szablonów, wykorzystuje usługi udostępniane przez registry
- AcDataAPI - abstrakcja bazy danych, udostępnia tylko niezbędne renderingowi dane
- Registry - rejestr usług tj. podmiana tagów, personalizacja treści itp.
- ACN - dotychczasowa aplikacja implementująca niektóre usługi, w końcowym etapie ma zniknąć

W tej pracy opisujemy naszą pracę nad renderingiem i dataAPI.



Rysunek 1.1: Schemat wydzielonych aplikacji

Rozdział 2

Implementacja elementu integrującego z bazą danych

2.1. Użyte technologie

1. JAVA w wersji 8
2. Spring - framework do tworzenia aplikacji Javowych
3. Jooq(Java Object Oriented Querying) - lekka biblioteka do mapowania baz danych. Posiadającą funkcję generowania klas Javy na podstawie bazy danych
4. Redis - wykorzystany do cachowania wyników zapytań
5. Elastic Stack - Logstash, Metricbeat, ElasticSearch i Kibana użyte do monitorowania aplikacji
6. Gatling - framework do tworzenia i wykonywania scenariuszy testów wdajnościowych
7. Docker - uruchamianie aplikacji w kontenerze

2.2. Założenia aplikacji

1. Aplikacja ma specyfikować oraz implementować kontrakt będący endpointami RESTowymi. Odpowiedzią na dane zapytanie jest JSON zawierający dane z bazy danych. Nazwy endpointów składają się z wersji API oraz nazwy. Np. */v1/language-version*
2. Odpowiedzi na zapytania mają być cachowane. Inwalidacja ??TODO
3. Częścią projektu jest napisanie testów poprawnościowych i wydajnościowych.

2.3. Iteracja 1

Wstępnie zrealizowany przykład oparcia rozwiązania w sposób bardzo bezpośrednio o pakiet Jooq został odrzucony po zaprezentowaniu przykładowej implementacji ze względu na ujawnione braki w architekturze rozwiązania. Ujawniła się konieczność zapewnienia ścisłego kontraktu poprzez API, mimo zmian schematu bazy danych z której pobierane są dane.

Zwłaszcza nad tą fazą projektu trwała najdłuższa dyskusja oraz analiza dostępnych narzędzi. Po rozpatrzeniu możliwości zaprezentowanych kawałkami kodu, osoba odpowiedzialna

za projekt z e-pointu zdecydowała się właśnie na pakiet Jooq w połączeniu z generacją klas na podstawie aktualnego stanu bazy danych, wydzielając jednak warstwę zależną od Jooq od reszty kodu, który nie zależy od sposobu dostępu do bazy danych.

Do monitorowania aktywnych instancji aplikacji użyliśmy gotowego, dedykowanego narzędzia spring-boot-admin. Niestety okazało się że konfiguracja jest równie łatwa co ograniczona i nie spełnia wymogów postawionych przez klienta.

2.4. Iteracja 2

Do realizacji zatem przyjęliśmy architekturę z kilkoma warstwami pośredniczącymi, każda wykonuje oddzielne zadania. I tak zaczynając od warstwy najbliżej bazy danych:

1. Umożliwia dostęp do bazy poprzez Jooqa
2. Zajmuje się sprawą podzielenia danych klientów na schematy w bazie danych (zarządzanie aktualnym schematem)
3. Ukrywa Jooq poprzez obiekty domenowe
4. Umożliwia zapewnienie kontraktu kompatybilnego wstecz pomimo zmian schematu bazy danych

Ostatni punkt zapewniony jest przez wersjonowanie API pakietami java, z wydzielonym wspólnym kodem - czyli zdecydowaną większością.

Konfiguracja cache sprowadzała się dodania kilku adnotacji w kodzie oraz konfiguracji Redisa.

Tym razem do monitorowania instancji aplikacji skorzystaliśmy z Elastic Stacka, który co prawda wymaga ręcznego wprowadzenia zasad dla wymaganych statystyk, ale w zamian daje możliwość niemalże dowolnej ich konfiguracji.

Kolejnym wymaganiem klienta było przeprowadzenie testów wydajnościowych zaproponowanego rozwiązania. W tym celu napisaliśmy scenariusze korzystając z frameworku Gatling napisanego w Scali.