Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka techniczna**
Specjalno: **Inżynieria systemów informatycznych**

PRACA DYPLOMOWA MAGISTERSKA

Analiza wydajnościowa interfejsów API w technologiach C# oraz NodeJS

Performance analysis of C# and NodeJS APIs

Maciej Grzela

Opiekun pracy
dr inż., Michał Kucharzak

WROCAW, 2022

Streszczenie

Streszczenie w jzyku polskim powininno zmieci si na poowie strony (drug poow powinien zaj abstract w jzyku angielskim).

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Nam id nulla a adipiscing tortor, dictum ut, lobortis urna. Donec non dui. Cras tempus orci ipsum, molestie quis, lacinia varius nunc, rhoncus purus, consectetuer congue risus.

Słowa kluczowe: raz, dwa, trzy, cztery

Abstract

Streszczenie in Polish should fit on the half of the page (the other half should be covered by the abstract in English).

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Nam id nulla a adipiscing tortor, dictum ut, lobortis urna. Donec non dui. Cras tempus orci ipsum, molestie quis, lacinia varius nunc, rhoncus purus, consectetuer congue risus.

Keywords: one, two, three, four

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Rozdział 1

Wstęp

1.1. Geneza pracy

Usługi sieciowe, zarówno te dostępne publicznie jak i te realizowane dla celów prywatnych, pełnią kluczową rolę w kontekście funkcjonowania współczesnej sieci internetowej. Zapewne nikt z nas, nie jest w stanie wyobrazić sobie kształtu obecnego Internetu bez takich rozwiązań sieciowych jak obsługa poczty elektronicznej, realizacja transferu plików, czy też przede wszystkim dostęp do aplikacji oraz witryn internetowych. Szczególnie w obrębie ostatniej spośród wymienionych usług, na przestrzeni ostatnich lat zauważyć można bardzo dużą liczbę zmian dotyczących sposobu ich definiowania oraz realizacji. Powodem pojawiania się tych zmian, jest niewątpliwie konieczność zachowania bądź też zwiększenia poziomów wydajności, niezawodności oraz bezpieczeństwa oferowanych rozwiązań, uwzględniając coraz to większy ruch sieciowy, generowany przez nieustannie zwiększającą się liczbę użytkowników Internetu. Ponadto, od nowoczesnego systemu internetowego, wymaga się coraz to większego poziomu skalowalności, a także płynności działania.

Poparciem niniejszych słów, może być treść wydawanego w kilkuletnich odstępach czasu raportu firmy Cisco, dotyczącego przewidywań oraz trendów sieciowych (tj. Cisco Annual Internet Report). Zgodnie z przedstawionymi w przytoczonym raporcie informacjami, a także porównując informacje te, z faktycznymi wartościami wskaźników dotyczących ruchu w internecie, zaobserwować możemy niemalże trzykrotny wzrost globalnego ruchu sieciowego na przestrzeni ostatnich pięciu lat. Ponadto, liczba klienckich urządzeń sieciowych, wykorzystywanych w celu uzyskania dostępu do usług udostępnianych w Internecie, na przestrzeni analogicznego przedziału czasowego, zwiększyła się z wartości 2,4 urządzenia na osobę, do poziomu niemalże czterech hostów sieciowych przypadających na pojedynczego reprezentanta globalnej populacji.

Należy także zwrócić uwagę, jakiego typu ruch sieciowy pełni dominującą rolę w kontekście dzisiejszego Internetu. Ponad 80% globalnego konsumenckiego ruchu internetowego stanowią dane dotyczące usług wideo, około dziesięciu procent światowego ruchu obejmują pozostałe treści udostępniane w ramach aplikacji oraz witryn internetowych, a pozostałe 10% to ruch generowany m.in. przez usługi transferu plików, poczty elektronicznej, czy też gier online. Na podstawie tych informacji, zauważyć można, że ponad 90% całości danych, przesyłanych w ramach globalnej sieci, musi być przetwarzanych przez aplikacje internetowe, bądź usługi sieciowe z nimi powiązane. Dlatego też, zaawansowane witryny internetowe komunikujące się z usługami sieciowymi, zwane dziś systemami internetowymi, tworzone są z wykorzystaniem coraz to bardziej udoskonalonych modeli architektonicznych, pozwalających na coraz to łatwiejszą budowę i rozwój rozwiązań przystosowanych do potrzeb aktualnego ruchu sieciowego [7].

Jednym z pierwszych, a także najbardziej podstawowych podejść do projektowania i implementacji systemów internetowych było wprowadzenie modelu architektury definiującego aplika-

cje monolityczne. W modelu tym, użytkownik aplikacji, wykorzystując oprogramowanie klienckie, którym w tym przypadku jest przeglądarka internetowa, wysyłał żądanie uzyskania zasobu definiując odpowiedni adres url (*(ang. Uniform Resource Locator)*). Żądanie to, odwoływało się bezpośrednio do fizycznego zasobu zlokalizowanego na serwerze, który przed dostarczeniem do klienta był przetwarzany przez serwer w celu uzupełnienia go danymi uzyskanymi z zewnętrznych źródeł - m.in. z systemu bazodanowego. Odpowiednio przygotowana statyczna zawartość odpowiedzi serwera, przybierająca postać pliku HTML (*ang. HyperText Markup Language*) była następnie przesyłana bezpośrednio do przeglądarki internetowej. Podejście to, wyróżniało się całkowitym brakiem dynamiki działania systemu internetowego, ponieważ każde zdarzenie wywoływane przez oprogramowanie klienta, wymagało zaadresowania i wygenerowania nowego żądania w kierunku serwera, którego odpowiedzią była nowa zawartość warstwy prezentacyjnej systemu.

W związku z zauważeniem pewnej regularności dotyczącej funkcjonowania większości systemów internetowych, związanej z faktem niejednokrotnego generowania nieznacznie różniących się od siebie odpowiedzi serwera, a także w związku z rozwojem języka skryptowego JavaScript oraz technologii Flash, aplikacje w ramach architektury monolitycznej zaczęły uwzględniać obsługę żądań zawierających przetworzone fragmenty warstwy prezentacyjnej. Ponadto, możliwa stała się dynamiczna podmiana określonych fragmentów treści, bez konieczności ponownego pozyskiwania pozostałej zawartości widoku. Usprawnienie to, opierające się na technice realizacji żądań asynchronicznych w ramach JavaScript (*ang. AJAX - Asynchronous JavaScript and XML*) pozwoliło na poprawę wydajności działania aplikacji internetowych przyczyniając się do zmniejszenia częstotliwości generowania zapytań, a także redukcji rozmiaru pojedynczej odpowiedzi serwera. Rozwiązanie to, nie wpływało jednakże bezpośrednio na strukturę systemu, której głównymi mankamentami były: pojedynczy centralny punkt przetwarzania żądań, a także brak separacji logiki działania systemu od warstwy prezentacyjnej.

Niedoskonałości omówionego powyżej modelu zostały zniwelowane poprzez wprowadzenie architektury zorientowanej na serwisy (*ang. SOA - Service Oriented Architecture*). W podejściu tym, dokonano separacji warstwy prezentacyjnej systemu, a także wszystkich pozostałych funkcjonalności dotyczących logiki biznesowej oraz przetwarzania danych. Reużywalne oraz autonomiczne usługi sieciowe pozwalały na realizację określonych funkcji systemu, a sposób komunikacji klienta z usługą, jak i komunikacji pomiędzy poszczególnymi serwisami definiowany był przez standaryzowane kontrakty. Zdefiniowanie architektury zorientowanej na serwisy umożliwiło budowę skalowalnych systemów internetowych, których poszczególne części mogły być realizowane w dowolnej technologii, a implementacja nowej funkcjonalności nie wymagała przebudowy pozostałych komponentów. Rozwiązanie to, wprowadzało jednak dodatkowy narzut dla każdej z przesyłanych wiadomości, wynikający ze ściśle określonej struktury żądania, tworzonej z wykorzystaniem języka XML (*ang. Extensible Markup Language*). Ponadto, wraz ze wzrostem poziomu zaawansowania systemu internetowego, autonomiczność oraz reużywalność poszczególnych komponentów malała ze względu na powstawanie specyficznych dla określonego rozwiązania zależności [36].

W związku z coraz to większymi wymaganiami dotyczącymi aplikacji internetowych, dominująca ówcześnie architektura rozproszonych usług sieciowych zastąpiona została poprzez model uwzględniający warstwę kliencką oraz interfejs programowania aplikacji (*ang. Application Programming Interface*). W przypadku nowoczesnych systemów internetowych, oba z tych komponentów budowane są w oparciu o architekturę n-warstwową (*ang. N-Tier Architecture Application*). W ramach niniejszego modelu, klient wysyła żądanie do interfejsu API, który na początku przetwarza jego treść, a następnie wywołuje usługę utworzoną w celu realizacji określonego zadania. Celem serwisu jest przetworzenie logiki biznesowej dla danej funkcjonalności, a także odwołanie się do usług dostępu do danych w celu ich uzyskania z zewnętrznego źródła informacji. Odpowiednio przygotowana odpowiedź jest następnie przekazywana do warstwy

obsługi żądania, która zwraca ją określonemu klientowi. W przeciwieństwie do pierwszego z przytoczonych modeli, odpowiedzią API nie jest dokument HTML, a jedynie dane dotyczące zasobu, które chce uzyskać klient. Sam zasób natomiast, nie jest elementem warstwy prezentacji systemu a zbiorem danych lub typem operacji, które można na tym zbiorze wykonać. Upraszczając, stwierdzić można, że API pełni rolę pośrednika pomiędzy warstwą prezentacji a zbiorem danych oraz operacji ich przetwarzania, a także dostarczania. Poszczególne usługi realizujące logikę biznesową aplikacji zawarte są bezpośrednio wewnątrz API, co nie oznacza jednakże, że nie mogą odwoływać się do serwisów zewnętrznych. Takie podejście do budowania systemów internetowych zapewnia zarówno skalowalność poszczególnych aplikacji wchodzących w skład systemu, jak i rozwiązuje problemy architektury SOA związane z zależnościami występującymi pomiędzy usługami. Dlatego też, architektura ta jest powszechnie wykorzystywana w celu budowy i zarządzania nowoczesnymi oraz zaawansowanymi systemami internetowymi [29].

Zarówno zdecentralizowana architektura zorientowana na serwisy, jak i centralna architektura oparta o interfejs programowania aplikacji, w przeciwieństwie do architektury monolitycznej, dostarcza zdecydowanie więcej możliwości związanych z ewaluacją działania poszczególnych komponentów systemu. Dzięki powstaniu ostatnich dwóch, spośród trzech przedstawionych modeli architektonicznych, możliwe jest nie tylko zbudowanie efektywnie działającej aplikacji internetowej, ale także ciągła ocena poprawności implementacji jej komponentów, w celu ustawicznego doskonalenia całego systemu.

Niniejsza praca, traktować będzie o ewaluacji efektywności działania interfejsów programowania aplikacji, w kontekście jednych z dwóch najpopularniejszych środowisk rozwoju oraz uruchamiania api. Ponadto, porównane zostaną parametry wydajnościowe w kontekście określonych przypadków użycia interfejsu API, będącego niezbędną częścią powszechnie wykorzystywanej architektury systemów internetowych.

1.2. Cel i zakres pracy

Celem pracy jest porównanie wydajności działania interfejsów programowania aplikacji, tworzonych z wykorzystaniem języków programowania C# oraz JavaScript. Interfejsy, wykonywane są w dwóch różnych środowiskach uruchomieniowych. Dla języka C#, środowiskiem tym jest platforma .NET, natomiast dla języka JavaScript – platforma NodeJS. Analiza porównawcza, obejmuje zarówno aspekty dotyczące efektywności działania samego interfejsu programowania aplikacji, jaki i elementów wchodzących w skład tworzonego systemu. Wśród omawianych rozwiązań, wyróżnić należy mappery obiektowo-relacyjne, systemy bazodanowe, czy też mechanizmy zarządzania pamięcią podręczną. Niektóre spośród wymienionych modułów stanowią integralną część API, natomiast pozostałe służą do rozszerzenia jego funkcjonalności.

Zakres pracy obejmuje: przegląd literaturowy, implementację środowiska badawczego, realizację badań oraz opracowanie wyników. Przegląd literatury tyczy się aspektów związanych ze strukturą i zasadą działania interfejsów programowania aplikacji, a także kwestii dotyczących wykonywania pomiarów wydajności dla poszczególnych operacji sieciowych. Operacje sieciowe, realizowane są w ramach obsługi żądania przez API. Etap implementacji środowisk badawczych składa się z budowy interfejsów w oparciu o porównywane środowiska rozwoju i uruchamiania aplikacji, a także konfiguracji platformy lokalnej oraz platform chmurowych, pozwalających na przeprowadzanie analizy działania systemów. Realizacja badań, przeprowadzona została pod kątem pomiaru czasu odpowiedzi na żądania użytkownika końcowego biorąc pod uwagę aspekty: wywołania serii żądań, obsługi współbieżności procesów, dostępności zasobów platformy hostingowej, a także możliwości oferowanych przez mappery obiektowo-relacyjne oraz systemy bazodanowe. Celem etapu opracowania wyników jest przedstawienie, wizualizacja oraz analiza różnic wartości czasów odpowiedzi interfejów API na poszczególne

żądania, w odniesieniu do przeprowadzonych badań. Zastosowanymi kryteriami oceny podczas przeprowadzanej analizy jest czas odpowiedzi interfejsu programowania aplikacji dla wygenerowanego żądania, a także maksymalna liczba żądań jakie jest w stanie obsłużyć określone API. Przedstawione kryteria, uwzględniane zostały w odniesieniu do wykorzystywanego środowiska uruchomieniowego oraz technologii implementacyjnej. Przeprowadzone badania, mają służyć wskazaniu zarówno pozytywnych aspektów, jak i problemów dotyczących wydajności działania aplikacji tworzonych z wykorzystaniem porównywanych technologii. Ponadto, celem jest także przedstawienie możliwości zwiększenia efektywności implementowanych interfejsów programowania aplikacji.

1.3. Struktura pracy

Niniejsza praca, podzielona została na sześć rozdziałów.

Pierwszy z nich, napisany został w celu zobrazowania dziedziny rozważanego problemu, a także podkreślenia jego wagi w kontekście zagadnienia usług sieciowych. Ponadto, w rozdziale tym zdefiniowano cel popełnionej pracy oraz przedstawiono zakres czynności realizowanych w ramach przeprowadzonych badań.

W rozdziale drugim dokonano wprowadzenia teoretycznego do tematyki interfejsów programowania aplikacji oraz testowania usług sieciowych. Wprowadzenie to, w odniesieniu do interfejsów API dotyczy zarówno struktury i zasady działania omawianej usługi sieciowej, jak i sposobu realizacji połączeń tej usługi z zewnętrznymi źródłami danych. W ramach wprowadzenia do tematyki testowania usług sieciowych wyjaśniono fundamentalne pojęcia teorii testowania oraz omówiono dostępne modele realizacji testów. Co więcej, nakreślono strategię wykonywania pomiarów wydajności w kontekście usług pracujących w sieciach komputerowych. W niniejszym rozdziale, zawarto również przegląd pozycji literaturowych, pomocnych w trakcie realizacji badań, a także przegląd technologii informatycznych, zastosowanych w procesie implementacji środowiska badawczego oraz wykonania pomiarów.

W ramach trzeciego z rozdziałów, zdefiniowano i omówiono każdy z aspektów rozważanego problemu badawczego. Dzięki temu, możliwe stało się sformułowanie zbioru rozważanych scenariuszy badawczych.

W celu realizacji badań opartych o zdefiniowane w rozdziale trzecim scenariusze badawcze, należy zaprojektować oraz zaimplementować odpowiednio dostosowane środowisko badań. Poszczególne kroki realizacji tego środowiska, zarówno te, dotyczące jego fizycznej struktury, jak i te, które tyczą się implementacji interfejsów programowania aplikacji, opisane zostały w rozdziale czwartym niniejszej pracy.

Piąty z rozdziałów, ma na celu przedstawienie rezultatów wynikających z przeprowadzonych prac naukowych. Rezultaty te, w obrębie niniejszego rozdziału zostały zgrupowane względem zdefiniowanych uprzednio scenariuszy badawczych, realizowanych w odpowiednio przystosowanym środowisku. Ponadto, dla uzyskanych wartości pomiarowych, dotyczących kryteriów poszczególnych badań, wykonano testy parametryczne, dzięki którym możliwa jest ocena istotności statystycznej zaobserwowanych różnic wynikowych. Co więcej, wyniki każdego z realizowanych scenariuszy badawczych poddane zostały krytycznej analizie.

Ostatni z rozdziałów pełni rolę podsumowania. Autor przedstawia w nim uzyskane efekty wykonanej pracy, a także nakreśla możliwości związane z dalszym rozwojem badań.

Rozdział 2

Wprowadzenie teoretyczne

2.1. Wykorzystywane terminy

W niniejszej pracy, posłużono się terminologią dystynktywną z punktu widzenia realizacji, rozwoju oraz ewaluacji usług sieciowych. Najbardziej istotne spośród wykorzystywanych terminów wymieniono poniżej. Dla każdego z pojęć, przedstawiono obcojęzyczne tłumaczenie, a także zdefiniowano spójny oraz zwięzły opis.

Usługa sieciowa

Web Service

Rodzaj systemu informatycznego cechującego się permanentnym wykonywaniem zdefiniowanych funkcji, tuż po uzyskaniu żądania. Żądanie to, przybiera postać danych, przekazanych w ramach systematycznej struktury. Sposób dostarczenia żądania, jego format, a także metoda odpowiedzi na żądanie, definiowane są poprzez protokół sieciowy z którego korzysta dana usługa.

Interfejs Programowania Aplikacji (API) - OPIS Z INŻ.

Application Programming Interface

Zbiór zasad oraz procedur determinujący sposób komunikacji pomiędzy wieloma aplikacjami. Aplikacjami tymi mogą być zarówno programy klienckie (np. strona webowa), jak i serwery danych.

API wykonane w technologii REST - OPIS Z INŻ.

RESTful API

Interfejs programowania aplikacji opierający swoją budowę oraz sposób funkcjonowania o zbiór ustalonych reguł. Reguły te, dotyczą między innymi: struktury żądań wysyłanych od klienta do serwera, budowy zasobu odpowiedzi serwera, a także kodów statusów zwracanych z chwilą odpowiedzi w zależności od wykonanej akcji.

Kontroler - OPIS Z INŻ.

Controller

Klasa, której zadaniem jest obsłużenie żądania aplikacji klienckiej, weryfikacja jego poprawności, a następnie wywołanie kodu logiki biznesowej w ramach struktur serwisów. Po otrzyma-

niu rezultatu obliczeń z warstwy logiki biznesowej, odpowiedzialnością metody kontrolera jest zwrócenie przetworzonego zasobu do systemu klienta.

Serwis - OPIS Z INŻ.

Service

Klasa, zawierająca metody odpowiedzialne za realizację logiki biznesowej w ramach interfejsu programowania aplikacji. Obiekt tej struktury danych, wywoływany jest bezpośrednio przed metody klas kontrolerów.

Repozytorium - OPIS Z INŻ.

Repository

Struktura danych, wykorzystywana do komunikacji interfejsu API z serwerem bazodanowym. Metody, w ramach klas repozytoriów, operują na modelu danych, przechowywanym w ramach API, a następnie, odwzorowują ten model za pomocą narzędzia ORM, na fizyczną zawartość bazodanową.

Mapper obiektowo-relacyjny (ORM)

Object-relational mapper (ORM)

Oprogramowanie, którego głównym zadaniem jest konwersja struktury klas modelu danych do fizycznej organizacji tabel w ramach systemu bazodanowego. Ponadto, mapper obiektowo-relacyjny dostarcza zbiór właściwości oraz metod stanowiących fasadę dla niskopoziomowych procedur dostępu do bazy danych, a także modyfikacji danych w niej zawartych.

Pamięć podręczna

Cache

Wydzielony fragment pamięci cechujący się szybkim czasem dostępu, wysoką przepustowością transmisji, a także ograniczonym okresem trwałego przechowywania danych. Pamięć ta, w kontekście webowego interfejsu programowania aplikacji, wykorzystywana jest w celu przechowywania wyników często realizowanych operacji, a także magazynowania uprzednio dostarczonych do klienta fragmentów odpowiedzi na żądania.

Wielowątkowość - OPIS Z INŻ.

Multithreading

Technika programowania, zakładająca wykorzystanie wielu odrębnie wykonywanych procesów w ramach jednej aplikacji. W przypadku interfejsu API implementującego tę technikę, każdy z punktów końcowych stanowi osobny wątek, będący częścią składową pojedynczego procesu. Dzięki temu, aplikacja jest dostępna, niezależnie od wywołanych, niekiedy długo trwających zadań.

Algorytm metaheurystyczny

Metaheuristic algorithm

Technika projektowania algorytmów nie zapewniających gwarancji uzyskania optimum dla rozważanego problemu, jednakże pozwalająca na zbudowanie systemu, dostarczającego rozwiązanie złożonego zagadnienia w akceptowalnym czasie, a także uzyskiwanego przy wykorzystaniu akceptowalnej ilości zasobów sprzętowych. Algorytm metaheurystyczny, poza konwencjonalnymi regułami stosowanymi w ramach standardowych wzorców programowania, implementuje reguły rozwiązywania problemów oparte na losowości, bądź też wywnioskowane na podstawie zjawisk fizycznych.

Punkt końcowy usługi sieci Web - OPIS Z INŻ.

Endpoint

Metoda klasy kontrolera, uruchamiana w momencie określenia żądania klienta. Do każdego z punktów końcowych, przypisany jest adres wywołania, zbiór wymaganych parametrów, a także obsługiwany typ metody protokołu hipertekstowego. Dzięki temu, bazując na strukturze otrzymanego żądania, interfejs API jest w stanie stwierdzić który z punktów końcowych powinien zostać wywołany.

Żądanie realizowane w ramach usługi protokołu hipertekstowego

HTTP Request

Struktura danych, wysyłana od aplikacji klienckiej (tj. aplikacji internetowej, przeglądarki, czy też programu klienta HTTP) w kierunku usługi sieciowej. Żądanie protokołu hipertekstowego charakteryzuje się jednoznacznie zdefiniowaną strukturą, uwzględniającą m.in. unikalny identyfikator zasobu, listę zdefiniowanych nagłówków, ciało żądania oraz jedną z dziewięciu dopuszczalnych metod HTTP.

Odpowiedź usługi protokołu hipertekstowego

HTTP Response

Struktura danych, wysyłana przez usługę sieciową w kierunku aplikacji klienckiej. Odpowiedź HTTP, ma na celu poinformowanie klienta serwisu webowego o statusie realizacji, wysłanego przez niego uprzednio żądania. Podstawowymi elementami odpowiedzi usługi protokołu hipertekstowego są: ciało odpowiedzi (zdefiniowane najczęściej z wykorzystaniem notacji JSON lub języka XML), kod odpowiedzi (liczba determinująca stan wykonania żądania), a także zbiór informacji nagłówkowych dotyczących typu danych zawartych w odpowiedzi, czy też fizycznych informacji o serwerze usługi sieciowej.

Kod odpowiedzi usługi protokołu hipertekstowego

HTTP Response Code

Liczba determinująca status realizacji żądania wysłanego przez aplikację kliencką. Kod odpowiedzi stanowi jedną z wymaganych składowych dotyczących standardowego rezultatu zwracanego w ramach usługi opartej o protokół hipertekstowy. Wyróżnić możemy pięć kategorii kodów odpowiedzi, niosących ze sobą odmienną informacje. Kategoriami tymi są: kody informacyjnej odpowiedzi (100-199), kody poprawnej odpowiedzi (200-299), kody wiadomości o

przekierowaniu (300-399), kody błędu aplikacji klienckiej (400-499), oraz kody błędu aplikacji serwerowej (500-599).

Czas odpowiedzi usługi protokołu hipertekstowego

HTTP Response Time

Wyrażony w milisekundach, przedział czasu od momentu otrzymania żądania wygenerowanego przez aplikację kliencką, do chwili zwrócenia rezultatu wykonywanych przez usługę sieciową obliczeń. Liczba ta, stanowi jedną z wartości pomiarowych, w kontekście efektywności działania interfejsu programowania aplikacji.

Obiektowa notacja JavaScript (JSON) - OPIS Z INŻ.

JavaScript Object Notation

Niezależny od języka programowania format prezentacji, definicji oraz wymiany danych w formie obiektów. Powszechnie stosowany jako sposób generowania ciała żądania wysyłanego do interfejsu API, a także odpowiedzi od niego uzyskiwanej.

Testy wzorcowe

Benchmark

Rodzaj ewaluacji oprogramowania, której zadaniem jest określenie referencyjnego poziomu wydajności dla testowanego systemu. Metryki, uzyskane w ramach testów wzorcowych, mogą zostać wykorzystane jako wartości ograniczeń względem testów obciążeniowych oraz przeciążeniowych.

Testy dymne

Smoke testing

Metoda testowania oprogramowania, której celem jest sprawdzenie poprawności funkcjonowania poszczególnych elementów systemu. Testy dymne, wykonywane są przed testami wydajnościowymi, po to aby upewnić się co do braku błędów implementacyjnych w ramach analizowanego oprogramowania.

Testy wydajności podstawowej

Baseline performance testing

Metoda ewaluacji oprogramowania, pozwalająca na weryfikację działania systemu w warunkach analogicznych do realiów standardowego działania. Na podstawie testów wydajności podstawowej, określić można wartości metryk, które będą miały zastosowanie jako punkt odniesienia dla kolejnych rodzajów testów. Ponadto, wykorzystując standard pomiaru wydajności aplikacji internetowych (taki jak np. APDEX), wartości uzyskane w ramach ewaluacji podstawowych, mogą posłużyć w celu określenia punktów satysfakcji, tolerancji oraz frustracji.

Testy obciążeniowe

Load testing

Rodzaj testów, które mają na celu określenie maksymalnego poziomu natężenia operacji, jakie mogą być generowane w kierunku oprogramowania. W kontekście niniejszej pracy, operacjami tymi są żądania wysyłane do interfejsu programowania aplikacji. Kluczowym aspektem testu obciążeniowego jest zdefiniowanie progu obciążenia aplikacji, powyżej którego system jest nie w stanie generować poprawnych odpowiedzi w akceptowalnym czasie.

Testy przeciążeniowe

Stress testing

Metoda ewaluacji oprogramowania, w ramach której natężenie operacji generowanych w kierunku testowanego oprogramowania zwiększone jest ponad ustalony próg tolerancji. Celem testu przeciążeniowego jest obserwacja sposobu działania systemu, w momencie, w którym nie jest on w stanie przetwarzać otrzymywanych żądań w sposób poprawny.

Asercja

Assertion

Wyrażenie typu prawda/fałsz, zdefiniowane w dowolnym miejscu programu, które przyjmuje wartość prawdziwą w momencie spełnienia hipotezy zawartej w ramach określonego przypadku testowego. Praktyczne podejście do procesu testowania funkcjonalności oprogramowania, sprowadza się do definiowania hipotez oraz ciągów operacji w kontekście przypadków testowych, a następnie weryfikacji tych hipotez z wykorzystaniem asercji.

2.2. Interfejsy programowania aplikacji

Webowy interfejs programowania aplikacji to usługa sieciowa, której celem jest realizacja zadań zleconych przez oprogramowanie klienta. Zadania te, dotyczą operacji wykonywanych w kontekście określonych zasobów. Wyróżnić możemy operacje zwane zapytaniami (tj. dotyczące pozyskiwania danych z ich źródeł), a także komendami (tj. związane z wykonywaniem operacji na danych).

Interfejsy API, budowane są z wykorzystaniem protokołu HTTP, dlatego też w ich kontekście możemy mówić o komunikacji bezstanowej definiującej pojęcia żądania oraz odpowiedzi. W związku z charakterystyką protokołu hipertekstowego, zarówno żądanie jak i odpowiedź cechuje się regularną strukturą zawierającą predefiniowane elementy.

Żądanie protokołu http wysyłane jest od aplikacji klienta do interfejsu API. Podstawową składową tego polecenia stanowi unikalny identyfikator zasobu URI (*ang. Uniform Resource Identifier*), na podstawie którego możliwe jest określenie fragmentu dziedziny obsługiwanego modelu danych. Informacja ta jednak, nie jest wystarczająca w kontekście realizacji jednej z funkcjonalności, zdefiniowanych w ramach API. Żądanie klienta, musi zostać uzupełnione o jedną z dziewięciu ustalonych metod http, obsługiwaną wersję protokołu, a także zbiór linii nagłówkowych. Opcjonalnie, informacja wysyłana w kierunku interfejsu, może zostać wzbogacona o zawartość tekstową określaną ciałem żądania (*ang. Request body*). Taki zbiór informacji, pozwala na jednoznaczną identyfikacje fragmentu kodu programu, który ma zostać wykonany wewnątrz interfejsu programowania aplikacji. W tabelach 2.1 oraz 2.2 przedstawiono kolejno

listę zdefiniowanych metod protokołu hipertekstowego wraz z wyjaśnieniem ich przeznaczenia, a także zbiór najczęściej wykorzystywanych linii nagłówkowych, w kontekście realizacji żądań.

Tab. 2.1: Zbiór dozwolonych metod protokołu hipertekstowego

Nazwa metody	Opis
GET	Pozyskanie danych dotyczących pojedynczej instancji określonego zasobu lub grupy instancji z opcjonalnym uwzględnieniem warunków kwalifikacji poszczególnej instancji do grupy.
POST	Definiowanie nowej instancji dotyczącej określonego typu zasobu. Przy zastosowaniu metody POST, wymagane jest zdefiniowanie ciała żądania, jako części składowej generowanej instrukcji.
PUT	Aktualizacja pełni zawartości instancji występującej w ramach odwołania się do określonego zasobu. Przy zastosowaniu metody PUT, wymagane jest zdefiniowanie ciała żądania, jako części składowej generowanej instrukcji.
DELETE	Usunięcie istniejącej instancji dotyczącej określonego typu zasobu.
PATCH	Aktualizacja fragmentu zawartości instancji występującej w ramach odwołania się do określonego zasobu. Przy zastosowaniu metody PATCH, wymagane jest zdefiniowanie ciała żądania, jako części składowej generowanej instrukcji.
HEAD	Pozyskanie zbioru linii nagłówkowych, które byłyby dostarczone wraz z ciałem odpowiedzi w ramach żądania wykorzystującego metodę GET. Wygenerowanie żądania HEAD umożliwia określenie charakteru danych, przed ich ewentualnym pozyskaniem.
OPTIONS	Pozyskanie informacji dotyczących charakterystyki oraz struktury serwera. Definiując żądanie typu OPTIONS, klient może dowiedzieć się o dopuszczalnych metodach HTTP obsługiwanych przez serwer, czy też uzyskać informacje o nazwie serwera oraz wykorzystywanym systemie operacyjnym.
CONNECT	Ustanowienie dwukierunkowej komunikacji pomiędzy klientem a serwerem. W przypadku realizacji komunikacji szyfrowanej, żądanie typu CONNECT pozwala na zestawienie zabezpieczonego tunelu pomiędzy hostami.
TRACE	Wygenerowanie komunikatu diagnostycznego w ramach pętli zwrotnej, którego celem jest osiągnięcie każdego z hostów, biorących udział w komunikacji.

Po wykonaniu kodu programu przypisanego do określonego rodzaju polecenia generowanego przez aplikacje kliencką, z interfejsu programowania aplikacji zwracana jest odpowiedź na żądanie (*ang. HTTP response*). Analogicznie do instrukcji realizacji danej czynności, także odpowiedź dotycząca statusu jej wykonania jest ustrukturyzowana zgodnie z wytycznymi zawartymi w definicji protokołu hipertekstowego. W ramach rezultatu zwróconego przez API wyróżnić należy: adres docelowy klienta, kod statusu, ciało odpowiedzi, a także zbiór linii nagłówkowych. Informacja zawarta w ramach kodu statusu, determinuje powodzenie realizowanej operacji, a treść dostarczanych linii nagłówkowych, może zostać wykorzystana w celu wnioskowania o charakterystyce odbywającej się komunikacji. Ciało odpowiedzi powinno zawierać dane dotyczące definiowanego w ramach identyfikatora żądania zasobu, w przypadku żądań wykorzystujących metodę GET. W kontekście pozostałych żądań, zgodnie z wytycznymi dokumentu RFC (*ang. Request For Comments*) o numerze 7230, powinno ono posiadać charakter informacji pomocniczej, bądź też pozostać puste [9]. W ramach tabel 2.3 oraz 2.4, wymienione zostały kolejno: zbiór najczęściej zwracanych linii nagłówkowych w kontekście odpowiedzi na żądanie, a także przedziały liczbowe dla kodów statusu odpowiedzi, wraz z ich semantyką.

Przedstawiony w niniejszy sposób interfejs programowania aplikacji scharakteryzować należy jako deterministyczny system wejściowo-wyjściowy. Ponadto, należy zauważyć, że w ramach systemu tego, występuje zjawisko inercji, powodowane koniecznością realizacji zdefiniowanego w ramach API kodu programu. Na podstawie tego założenia, ewaluację działania oraz

Tab. 2.2: Zbiór najczęściej wykorzystywanych linii nagłówkowych w kontekście żądania protokołu hipertekstowego

Linia nagłówkowa	Znaczenie	Dopuszczalna zawartość
accept	Typ zawartości, którą jest w stanie przetwarzać aplikacja kliencka	Identyfikator typu MIME (*ang. Multipurpose Internet Mail Extensions*) lub zapis */* oznaczający dowolną zawartość
accept-encoding	Sposób kodowania znaków, rozumiany przez stronę klienta	Zbiór formatów kodowania zdefiniowany w ramach rejestru formatów IANA
accept-language	Język naturalny, preferowany przez stronę kliencką	Pojedyncza wartość reprezentująca określony kraj lub region, bądź też lista niniejszych wartości wraz z parametrem istotności poszczególnego kodu lokalizacji
content-length	Długość ciała żądania wyrażona w bajtach	Liczba naturalna
content-type	Format zawartości ciała żądania	Identyfikator typu MIME wraz ze sposobem kodowania wiadomości
cookie	Zbiór informacji pozwalających na wprowadzenie oraz utrzymanie stanowego charakteru transmisji	Zestaw par klucz-wartość, gdzie klucz jest wartością tekstową, a wartość przyjmuje postać dowolną
origin	Informacja determinująca pochodzenie żądania	Ciąg tekstowy składający się z nazwy protokołu, nazwy hosta oraz numeru portu
user-agent	Specyfikacja techniczna oprogramowania klienta	Ciąg znaków zawierający informacje o nazwie produktu, jego wersji, platformie sprzętowej, czy też systemie operacyjnym

Tab. 2.3: Zbiór najczęściej zwracanych linii nagłówkowych w kontekście odpowiedzi protokołu hipertekstowego

Linia nagłówkowa	Znaczenie	Dopuszczalna zawartość
access-control-allow-credentials	Określenie, czy odpowiedź serwera ma być osiągalna z kodu JavaScript aplikacji klienckiej, w momencie gdy nagłówek żądania dotyczący poświadczeń, zezwala na ich dołączenie	Wartość prawda/fałsz
access-control-allow-origin	Informacja dotycząca pochodzenia klienta, który może ubiegać się o otrzymanie odpowiedzi od serwera	adres hosta klienckiego lub symbol gwiazdki oznaczający zezwolenie dla wszystkich hostów
cache-control	Dane konfiguracyjne dotyczące obsługi pamięci podręcznej	Zbiór par klucz-wartość określających zachowanie pamięci cache w kontekście określonej komunikacji
content-length	Długość ciała odpowiedzi wyrażona w bajtach	Liczba naturalna
content-type	Format zawartości ciała odpowiedzi	Identyfikator typu MIME wraz ze sposobem kodowania wiadomości
cross-origin-resource-policy	Polecenie ignorowania żądań realizowanych pomiędzy źródłami bądź witrynami w kontekście określonego zasobu	Wartość prawda/fałsz
expires	Data wygaśnięcia ważności niniejszej odpowiedzi	Określona data
server	Nazwa hosta dostarczającego odpowiedź klientowi	Ciąg znaków

Tab. 2.4: Zbiór kodów statusu odpowiedzi protokołu hipertekstowego

Przedział liczbowy	Semantyka w kontekście odpowiedzi
100 - 199	Zbiór kodów informacyjnych - żądanie jest aktualnie przetwarzanie
200 - 299	Zbiór kodów poprawnej odpowiedzi - wystosowane żądanie zostało zrealizowane poprawnie
300 - 399	Zbiór kodów przekierowań - istnieć może wiele akceptowalnych odpowiedzi dla żądania bądź realizacja określonej operacji wymusza odwołanie się pod adres identyfikujący odmienny zasób
400 - 499	Zbiór kodów błędu po stronie klienta - wygenerowane żądanie zawiera błędy, oczekiwany zasób nie istnieje, klient nie jest uwierzytelniony lub nie posiada określonego poziomu uprawnień
500 - 599	Zbiór kodów błędu po stronie serwera - pomimo poprawnej struktury wygenerowanego żądania, serwer nie jest w stanie zrealizować przydzielonej mu operacji

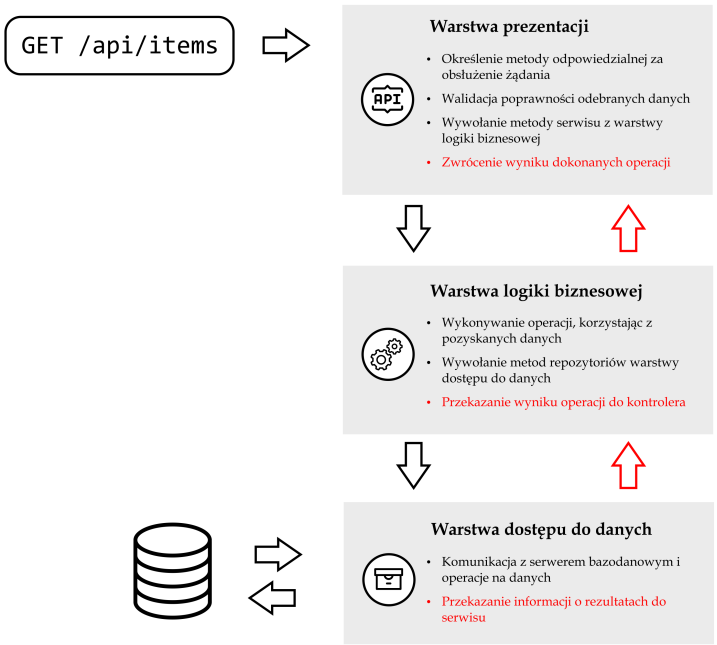
wydajności interfejsu programowania aplikacji przeprowadzić można poprzez wprowadzanie określonego wejścia (tj. generowanie żądania) oraz obserwację wartości zwróconej na wyjściu (tj. uzyskana odpowiedź).

Proces przetwarzania żądania wewnątrz interfejsu API

Po uzyskaniu żądania otrzymanego od strony klienta, zadaniem interfejsu programowania aplikacji jest wybór określonej klasy kontrolera, a także zawartej w niej metody. Każda z klas kontrolerów stworzona jest w celu obsługi operacji związanych z konkretnym zasobem, a poszczególna metoda tej klasy implementuje zachowanie które ma zostać wywołane w kontekście dostarczonego typu oraz identyfikatora polecenia.

Wewnątrz metody klasy warstwy kontrolerów, wywoływane zostają operacje zdefiniowane w usługach warstwy biznesowej. Usługi te, realizowane mogą być zarówno wewnątrz api jak i stanowić odrębny system internetowy. Klasy warstwy logiki biznesowej, zwane serwisami, złożone są z metod, których głównym celem jest weryfikacja poprawności otrzymanych informacji w kontekście obsługiwanych zasobów, a także pozyskiwanie danych oraz wykonywanie operacji na nich, poprzez odwoływanie się do metod warstwy dostępu do danych.

Zbiór klas warstwy dostępu do danych, stanowi ostatni z logicznych poziomów, definiowanych w ramach architektury API. Fragmenty kodu zdefiniowane w tej warstwie, zwane repozytoriami, mają za zadanie obsłużyć komunikację pomiędzy interfejsem programowania aplikacji, a określonym źródłem danych. Ponadto, metody klas repozytoriów, dostarczają warstwie logiki biznesowej interfejs operacji na danych. Dzięki temu, żądanie może być przetwarzane od warstw najwyższych (tj. warstwy kontrolerów) do warstwy najniższej (tj. warstwy dostępu do danych), natomiast odpowiedź na żądanie jest konsolidowana w kierunku odwrotnym [18]. Na ilustracji 2.1 przedstawiono przepływ informacji wewnątrz interfejsu API, od momentu wygenerowania żądania do chwili uzyskania odpowiedzi.



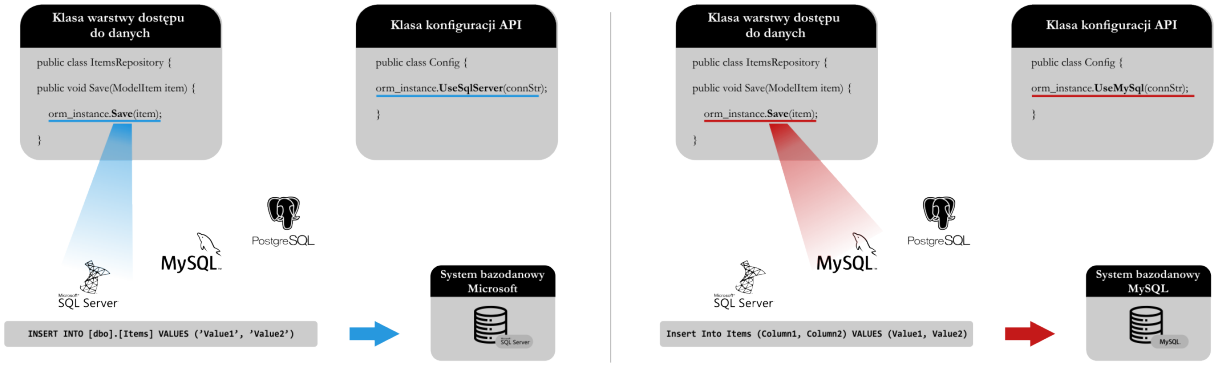
Rys. 2.1: Proces przetwarzania żądania wewnątrz interfejsu API

Konwersja obiektowo-relacyjna

W celu uproszczenia procesu pozyskiwania oraz modyfikacji danych z zewnętrznych źródeł, a także unifikacji sposobu interakcji z nimi, w ramach interfejsów programowania aplikacji, powszechnie wykorzystywane jest oprogramowanie zwane mapperem obiektowo-relacyjnym (*ang. Object-Relational Mapper*). Założeniem oprogramowania tego, jest zdefiniowanie warstwy abstrakcji pomiędzy interfejsem programowania aplikacji a językiem programowania bądź zbiorem poleceń, wykorzystywanym w ramach obsługi źródła danych.

Podstawowe składowe oprogramowania typu ORM to jednolity interfejs operacji na zbiorze danych, klasy kontekstu bazodanowego, a także metody obsługi komunikacji z bazą danych.

Dzięki wprowadzeniu jednolitego interfejsu operacji na danych, niezależnie od źródła informacji z jakim komunikuje się API, wydanie konkretnego polecenia do dowolnego systemu bazodanowego równoznaczne jest z każdorazowym wywołaniem funkcji o takiej samej sygnaturze. Stosując takie podejście, konstruktor interfejsu programowania aplikacji nie staje się uzależniony od źródła danych z którym pracuje. Ponadto, istnieje możliwość zamiany lub połączenia dodatkowego systemu bazodanowego, a operacja ta, nie wpływa w jakikolwiek sposób na działanie interfejsu API. Niniejsza zależność została zilustrowana na rysunku 2.2



Rys. 2.2: Zasada działania oprogramowania mappera obiektowo-relacyjnego w kontekście jednolitego interfejsu operacji na zbiorze danych

Dystynktywnym elementem oprogramowania mappera obiektowo-relacyjnego jest klasa kontekstu bazodanowego. Klasa ta, jest kontenerem struktur w ramach których wyróżnić możemy zbiory elementów modelu danych, a także konfigurację poszczególnych ich właściwości. Podstawową ideą omawianej konwersji dziedziny obiektowej do domeny relacyjnej jest zdefiniowanie zbioru klas, opisujących wykorzystywane zasoby, a następnie odwzorowanie ich w relacyjnym modelu danych, obsługiwanym przez wybrany system bazodanowy. Klasa kontekstu pozwala na określenie, które spośród struktur danych zdefiniowanych w ramach API powinny zostać rzutowane na obiekty tabel generowanych w obrębie bazy danych. Ponadto, dla właściwości każdej z klas modelu danych, zdefiniować należy konfigurację, która zostanie przetransformowana do modelu relacyjnego. W zakresie klasy kontekstu bazy danych, opisywane są także relacje, jakie mają zostać wygenerowane pomiędzy poszczególnymi elementami modelu.

W celu nawiązania, utrzymania, a także zakończenia komunikacji z zewnętrznym źródłem danych, oprogramowanie ORM wykorzystuje klasy zwane konektorami. Klasy te, dostarczają przejrzysty interfejs obsługi połączenia, który następnie jest opakowywany w zunifikowany interfejs, dostępny bezpośrednio dla twórcy API [34].

Uwierzytelnienie oraz autoryzacja

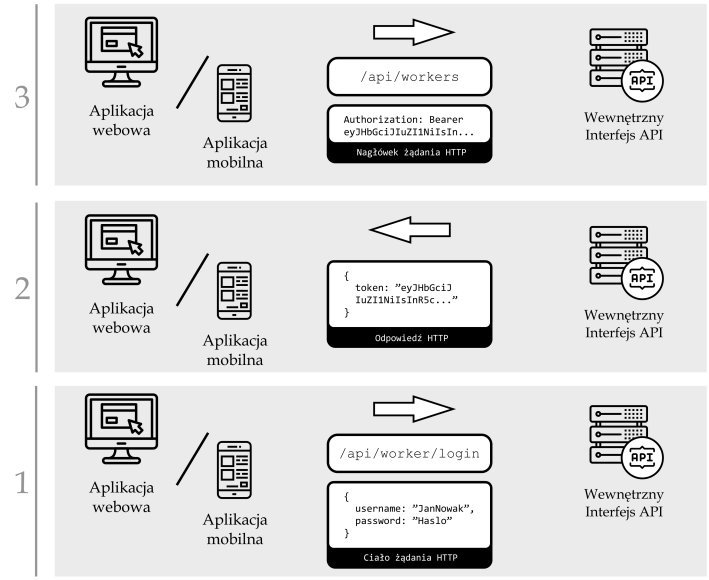
Proces uwierzytelnienia oraz autoryzacji użytkownika odwołującego się do interfejsu programowania aplikacji, przedstawić należy w trzech następujących krokach.

Pierwszym z nich, jest wygenerowanie żądania odwołującego się do punktu końcowego odpowiedzialnego za obsługę uwierzytelnienia wewnątrz API. Żądanie to, musi posiadać ciało, zawierające informacje poświadczające o konkretnymi użytkowniku. Najczęściej, informacją tą, jest nazwa użytkownika oraz hasło.

Następnie, dostarczone referencje są analizowane przez mechanizmy uwierzytelniania implementowane w ramach API. W rezultacie tych operacji, zwrócona zostaje pozytywna odpowiedź zawierająca token autoryzujący bądź też negatywna, posiadająca w sobie informację o błędzie uwierzytelnienia klienta.

Strona kliencka może autoryzować dysponowane operacje przed interfejsem programowania aplikacji, uwzględniając w ramach linii nagłówkowej żądania token uwierzytelniający. Dostarczona w ten sposób informacja, pozwala na identyfikację użytkownika w ramach interfejsu API, a także na określenie przypisanego użytkownikowi poziomu uprawnień. W ramach struktury tokenu, zawarta jest także informacja o jego czasie ważności, dlatego też, procedura uwierzytelniania musi być regularnie ponawiana [19].

Na rysunku 2.3, zilustrowany został proces uwierzytelnienia i autoryzacji aplikacji klienta przez interfejsem programowania aplikacji.



Rys. 2.3: Proces uwierzytelnienia oraz autoryzacji użytkownika przed interfejsem API

Separacja zapytań oraz komend w kontekście odwołań do źródeł danych

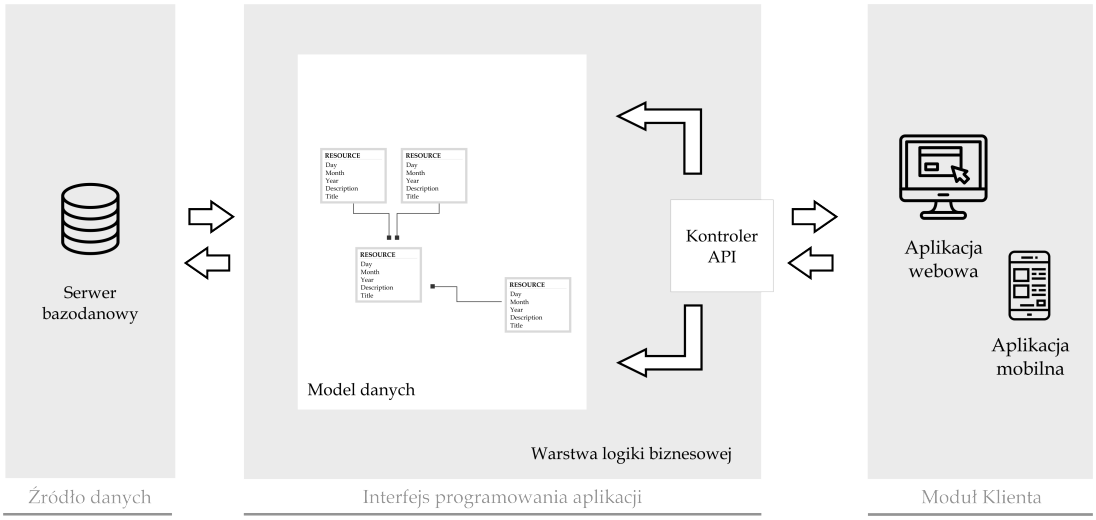
Wraz z rosnącą liczbą żądań obsługiwanych w ramach zaawansowanych interfejsów programowania aplikacji, zauważone zostało zjawisko asymetrii w kontekście typów wiadomości generowanych przez klientów. Zapytania dotyczące pozyskiwania danych z API realizowane jest z nieporównywalnie większą częstością niż operacje ich modyfikacji. Dlatego też, zdefiniowany został wzorzec projektowy dotyczący separacji zapytań oraz komend generowanych względem usługi sieciowej *(ang. Command Query Responsibility Segregation)*.

Zastosowanie przedstawionego powyżej wzorca projektowego wiąże się z koniecznością budowy dwóch osobnych modeli danych. Pierwszy z nich, wykorzystywany jest w kontekście odczytu informacji. Na modelu tym, dokonywana jest najczęściej operacja optymalizacji, której celem jest redukcja rozmiaru składowych modelu, a także szybkości przetwarzania bardziej złożonych struktur będących jego częścią. Drugi z modeli danych, znajduje zastosowanie w

aspekcie modyfikacji określonych zasobów. Biorąc pod uwagę standardowy sposób eksploatacji interfejsu programowania aplikacji, model ten cechować się może niższą wydajnością. W zależności od specyfiki określonej usługi sieciowej, optymalizowany może być albo model odczytu, albo też model zapisu. Ponadto, implementując wzorzec projektowy separacji zapytań oraz komend, wykorzystać można dwa osobne zewnętrzne źródła danych, przystosowane do wydajniejszego wykonywania określonego typu operacji, bądź też dostosowane do obsługi większego ruchu sieciowego.

Niewątpliwymi zaletami, wynikającymi z zastosowania opisywanego wzorca projektowego są: zwiększenie efektywności operacji realizowanych z dużą częstotliwością, możliwość korzystania z osobnych źródeł danych dla operacji odczytu oraz zapisu, zachowanie zasady pojedynczej odpowiedzialności *(ang. Single Responsibility Principle)* względem klas logiki biznesowej API, a także redukcja liczby wstrzykiwanych zależności *(ang. Dependency Injection)* w ramach klas kontrolerów interfejsu [30].

Na ilustracjach 2.4 oraz 2.5 przedstawiono kolejno schemat przetwarzania żądań wewnątrz API z uwzględnieniem wzorca CQRS, a także przy wykorzystaniu pojedynczego modelu danych.



Rys. 2.4: Schemat przetworzenia żądania przez interfejs API dla architektury z jednym modelem danych

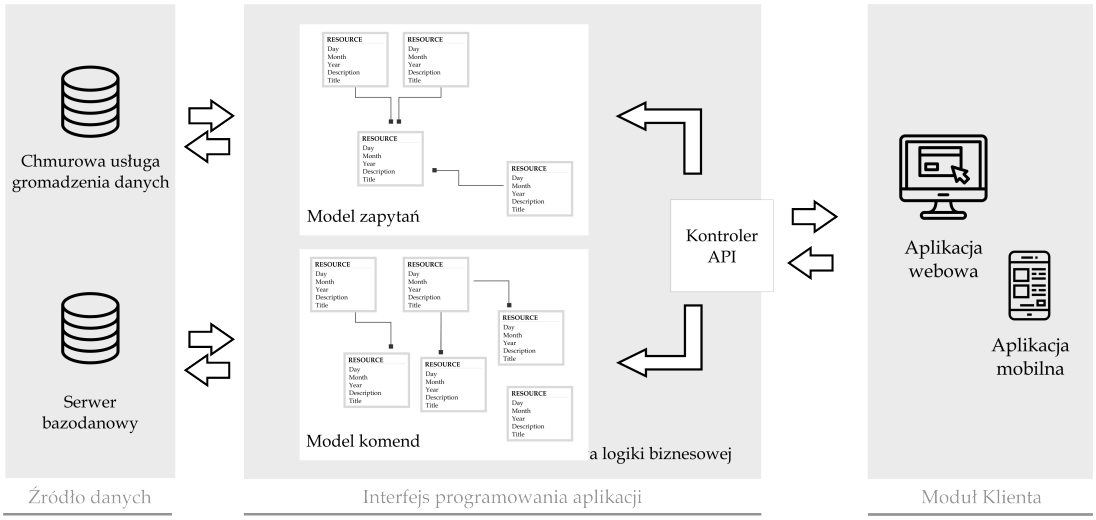
Odwołując się do ilustracji 2.5 należy zaznaczyć, że jest to jedna z możliwości implementacji wzorca CQRS, uwzględniająca wykorzystanie odseparowanych źródeł danych. Architektura separacji zapytań oraz komend w kontekście żądań wysyłanych do interfejsu API, może być także wprowadzona przy wykorzystaniu pojedynczego systemu bazodanowego.

Konwencja REST

Niezależnie od struktury wewnętrznej omawianych usług sieciowych, współczesne interfejsy programowania aplikacji projektowane są tak, aby ich zewnętrzna warstwa (tj. widziana z perspektywy aplikacji klienckiej) cechowała się jednolitą kompozycją.

Jednym z najpopularniejszych sposobów zapewnienia jednolitego interfejsu komunikacyjnego pomiędzy klientami i usługami sieciowymi, przetwarzającymi informacje z wykorzystaniem protokołu HTTP, jest konwencja oraz styl architektoniczny REST *(ang. Representational State Transfer)*.

Konwencja ta, definiuje zbiór zasad dotyczących m.in. zachowania usługi sieciowej w kontekście przetwarzania żądania konkretnego typu, struktury i elementów odpowiedzi na określone



Rys. 2.5: Schemat przetworzenia żądania przez interfejs API dla architektury wykorzystującej wzorzec projektowy CQRS

żądanie, semantyki wykorzystywanych statusów rezultatu przetwarzania, bezstanowego charakteru komunikacji, czy też syntaktyki odwołań do poszczególnych punktów końcowych.

W kontekście stopnia implementacji stylu architektonicznego REST w ramach interfejsu programowania aplikacji, wprowadzić należy pojęcie modelu dojrzałości Richardsona *(ang. Richardson Maturity Model)*. Pojęcie to, definiuje cztery poziomy przystosowania interfejsu API do omawianej w niniejszej sekcji konwencji.

W odniesieniu do poziomu zerowego, powinnością interfejsu programowania aplikacji jest udostępnienie usług w ramach pojedynczego adresu sieciowego, niezależnie od wykorzystywanych metod HTTP. Struktura żadania klienckiego, w sposób jednoznaczny dostarczać ma informację na temat wykonywanego wewnątrz usługi sieciowej działania.

Zasada poziomu pierwszego, odnosi się do charakterystyki interfejsu API jako usługi zorientowanej na zasoby. Niezależnie od czynności, jaka ma zostać wykonana przez omawianą usługę sieciową, opis tej czynności wskazywać ma na zasób którego ona dotyczy.

Reguła stanowiąca definicję poziomu trzeciego, związana jest z semantyką poszczególnych typów żądań protokołu hipertekstowego. Żądanie o takim samym adresie sieciowym, pełnić powinno odmienną rolę, w zależności od rodzaju żądania HTTP.

Ostatnim z poziomów dojrzałości interfejsu programowania aplikacji opartego o konwencję REST jest reguła HATEOAS *(ang. Hypertext As The Engine Of Application State)*. Reguła ta, definiuje interfejs API jako źródło informacji dotyczącej obsługi stanu całego systemu internetowego (tj. usługi sieciowej wraz z aplikacjami klienckimi). Klient, po uzyskaniu odpowiedzi serwera na żądanie, powinien na podstawie zawartości tej odpowiedzi móc zdefiniować przyszłe czynności, które wolno mu wykonać [37].

2.3. Testowanie oprogramowania

Aspekt badawczy niniejszej pracy, związany jest z realizacją procesu ewaluacji wydajności interfejsów programowania aplikacji, pod kątem wykorzystania odmiennych środowisk implementacyjnych oraz uruchomieniowych. Proces ten, jest tylko jednym z wielu elementów domeny testowania oprogramowania, której charakterystyka uwzględnia zbiór sztywno zdefiniowanych reguł cechujących się wysokim poziomem sformalizowania. W następnych sekcjach niniejszego akapitu dokonane zostało wprowadzenie dotyczące zagadnienia ewaluacji oprogramowania, nakreślone zostały zasady testowania systemów informatycznych, zdefiniono taksonomię technik

testowania, a także omówiono proces przeprowadzania ewaluacji wydajności usługi sieciowej jaką jest interfejs programowania aplikacji.

Wprowadzenie do zagadnienia ewaluacji oprogramowania

Ewaluacja poszczególnych składowych tworzonego oprogramowania jest niezbędną częścią procesu budowy systemu informatycznego,niezależnie od jego charakterystyki, czy też wykorzystywanej do jego budowy technologii. W rozumieniu ogólnym, proces testowania często sprowadzany jest do zbioru dwóch czynności. Czynnościami tymi są uruchamianie oprogramowania, a także eksploracja jego funkcjonalności w celu dostrzegania tych, w ramach których zauważyć można niezgodność ich działania w stosunku do specyfikacji. Takie wnioskowanie jednak jest niepełne, i uwzględnia ono tylko jeden z etapów składających się na cały proces testowania. Dziedzinę ewaluacji cech programów komputerowych, poszerzyć należy ponadto o takie elementy jak: planowanie testów, wybór kryteriów oceny oprogramowania, nadzór oraz kontrolę realizacji badań, projektowanie przypadków testowych, czy też analizę spełnienia ustalonych kryteriów zakończenia.

Wyróżnić możemy znaczącą liczbę definicji testowania oprogramowania, a każda z nich wprowadza inny poziom szczegółowości. Ponadto, wiele spośród formułowanych pojęć nawiązuje do różnych aspektów omawianego procesu. Zgodnie z jedną z najbardziej generycznych definicji, wprowadzoną przez Hetzla w publikacji [13], proces testowania oprogramowania określić należy jako zbiór wszystkich czynności, które nakierowane są na weryfikację atrybutów i właściwości programu, a także sprawdzenie tego, czy określony system spełnia założone wymagania. Definicja ta, względem wielu innych popularnych sformułowań dotyczących ewaluacji oprogramowania, uwzględnia możliwość zastosowania statycznych technik testowania. Ponadto, jej autor bierze pod uwagę fakt, że w ramach procesu ewaluacji, oceniany powinien być każdy z artefaktów tworzonych w ramach systemu, a nie tylko i wyłącznie kod źródłowy programu.

Taksonomia technik testowania

Jako jedno z podstawowych kryteriów podziału technik testowania oprogramowania, wskazać należy rodzaj czynności wykonywanej przez stronę testującą, której realizacja prowadzi do uzyskania charakterystyki programu poddanego ewaluacji. Według kryterium tego, wyróżnić należy statyczne oraz dynamiczne techniki testowania.

Pierwsze, spośród przytoczonych metod, opierają się na analizie artefaktów oprogramowania (takich jak m.in.: kod źródłowy, specyfikacja, dokumentacja, czy też lista wymagań) bez ich wykonywania. Jako praktyczne przykłady przedstawionej techniki, zdefiniować należy: generowanie metryk kodu źródłowego programu, analizę przepływu sterowania, formalne dowodzenie poprawności działania, a także interpretację grafów wywołań.

Metody dynamiczne natomiast, związane są z weryfikacją właściwości poszczególnych elementów systemu informatycznego w trakcie jego wykonywania. Ten rodzaj testowania, nie uwzględnia formalnych struktur liczbowych, jakimi są grafy przepływu sterowania, czy też metryki kodu źródłowego programu. Zorientowany jest on, na odbiór systemu z perspektywy korzystającego z niego klienta.

Innym z rozważanych kryteriów podziału technik testowania jest ich umiejscowienie metody względem określonego fragmentu procesu wytwórczego. W nawiązaniu do tego aspektu, zdefiniować należy pojęcie poziomu testów, które jest powiązaniem sposobu ewaluacji oprogramowania z etapem jego realizacji. Istotą rozróżniania danych poziomów testów jest założenie różnorodności celów testowania, a także testowanych obiektów, określanych w kontekście

każdej z warstw. W odniesieniu do najbardziej popularnych systematyk poziomów testowania wyróżnić można następujące elementy:

- testy jednostkowe (zwane także modułowymi lub testami komponentów)
- testy integracyjne
- testy systemowe
- testy akceptacyjne

Pierwszy z poziomów, dotyczy znajdowania niezgodności specyfikacyjnych w obrębie logicznie oddzielonych jednostek oprogramowania *(ang. Software Units)*. Każda z jednostek, powinna być testowana w izolacji od pozostałych elementów systemu. Ze względu na złożoność rozwiązań informatycznych, a także statystycznie wysoki współczynnik wzajemnych zależności modułowych, warunek ten często nie może zostać spełniony. W takich sytuacjach, aby dostarczyć zależność do testowanej jednostki, budowane są moduły zastępcze, imitujące poprawne zachowanie określonego fragmentu programu *(ang. Mocks)*. Omawiany poziom testowania, często postrzegany jest jako jeden z etapów procesu wytwórczego, szczególnie w ramach takich technik jak rozwój oprogramowania napędzany testowaniem *(ang. Test Driven Development)*.

Celem kolejnego z poziomów testowania jest weryfikacja poprawności współoddziaływania indywidualnych komponentów, a także prawidłowości funkcjonowania interfejsów definiowanych pomiędzy nimi. Przykładem współdziałania jednostek oprogramowania może być współpraca interfejsu programowania aplikacji, będącego systemem poddawanym analizie w ramach niniejszej pracy, a także określonego silnika bazodanowego. W zależności od liczności weryfikowanych powiązań pomiędzy poszczególnymi jednostkami oprogramowania, a także w odniesieniu do liczby samych modułów będących częścią testowanego fragmentu systemu, wyróżnić możemy testy małej oraz dużej skali integracji.

Na temat testów systemowych, należy mówić wtedy, gdy wszystkie z elementów rozwiązania informatycznego zostały ze sobą powiązane w sposób spójny. Celem testów, realizowanych w ramach tego poziomu, jest weryfikacja wysokopoziomowej funkcjonalności oprogramowania, a także wykonywanie scenariuszy ewaluacji systemu z poziomu regularnego użytkownika *(ang. End-to-End testing)*.

Ostatnim z wymienionych poziomów ewaluacji są testy akceptacyjne. Przedmiotem oceny w ramach tego rodzaju testów jest gotowe rozwiązanie informatyczne w postaci komercyjnego produktu. Podmiot odpowiedzialny za realizację omawianych testów przygotowuje listę kryteriów akceptacji *(ang. acceptance criteria)*, a następnie na podstawie obsługi testowanego rozwiązania, potwierdza lub odrzuca spełnienie każdego z nich. Celem omawianych ewaluacji nie jest znajdowanie błędów działania systemu, a nabranie zaufania co do jakości jego funkcjonalnych oraz niefunkcjonalnych atrybutów.

Wykonując testy definiowane w ramach kolejnych poziomów ewaluacji, weryfikowane zostają na początek funkcjonalne, a następnie niefunkcjonalne elementy testowanego systemu. Jako weryfikację elementów funkcjonalnych, rozumieć należy wszystkie te czynności, które podejmowane są w ramach wszystkich wymienionych powyżej poziomów testów, z wyjątkiem testów akceptacyjnych. Ewaluacja niefunkcjonalna natomiast, odnosi się tylko do ostatniego spośród wyróżnionych poziomów testowania.

Podział charakteryzujący przedmiot ewaluacji względem omówionych aspektów definiuje pojęcia testów funkcjonalnych oraz niefunkcjonalnych i w kontekście niniejszej pracy jest on podziałem kluczowym.

Badania przeprowadzone w ramach tej pracy posiadają charakter testów niefunkcjonalnych, a ich wykonanie poprzedzone jest weryfikacją funkcjonalną, której poprawność traktowana jest jako wymóg.

Ewaluacja wydajności interfejsów programowania aplikacji

Zgodnie z teorią przedstawioną w sekcji 2.2 niniejszej pracy interfejs programowania aplikacji postrzegać można jako deterministyczny system wejściowo-wyjściowy o charakterze dyskretnym. Takie podejście, w znaczący sposób ułatwia proces ewaluacji wydajności interfejsów API.

Definiowanie interfejsu API jako systemu pobudzanego pojedynczym wejściem, a także generującego pojedynczą wartość wyjściową, pozwala na wykorzystanie sposobu oceny wydajności zwanego testem czarnoskrzynkowym *(ang. Black-box testing)*. W ramach tego rodzaju testu, określone kryterium ewaluacji wyliczane jest jako różnica wartości pomiaru na wyjściu systemu, względem tej, której kalkulacja nastąpiła na jego wejściu. Taki rodzaj testu, umożliwia wyliczenie metryki wydajności, bez konieczności przygotowywania systemu do przeprowadzenia procesu testowania.

Podstawowym kryterium oceny wydajności interfejsu programowania aplikacji jest czas odpowiedzi na żądanie. Metrykę tę, określić należy jako czas od momentu wygenerowania żądania przez stronę klienta, do chwili uzyskania przez niego odpowiedzi. Zauważyć należy również zależność czasu odpowiedzi na żądanie, zarówno od rozmiaru żądania jak i wielkości jego odpowiedzi. Ponadto, czynnikiem wpływającym na uzyskany rezultat pomiaru jest niewątpliwie przepustowość łącza sieciowego pomiędzy klientem a serwerem.

Aby rezultaty uzyskane w ramach oceny wydajności API mogły zostać postrzegane jako rzetelne, omówione powyżej czynniki muszą cechować się statyczną charakterystyką, bądź też zostać całkowicie wyeliminowane z procesu testowego. W ramach niniejszej pracy, rozmiar odpowiedzi generowanej przez interfejsy programowania aplikacji jest stały niezależnie od zastosowanej technologii, a także środowiska uruchomieniowego. Wynika to z zastosowania pojedynczego i ustrukturyzowanego modelu danych, który jest identyczny, niezależnie od API. W odniesieniu do zmienności prędkości łącza internetowego, aspekt ten został wyeliminowany poprzez przeprowadzanie testów w obrębie lokalnej sieci komputerowej, a także umiejscowienie interfejsów oraz systemów bazodanowych w ramach tej właśnie sieci.

Kolejnym z kryteriów oceny wydajności, uwzględnianym w ramach procesu testowania usług sieciowych jest poprawność uzyskanej odpowiedzi w odniesieniu do liczby klientów, równolegle generujących żądania. Kryterium to, charakteryzuje się silną korelacją z czasem odpowiedzi na pojedyncze zapytanie.

Uwzględniając oba opisane powyżej parametry, podstawowy schemat scenariusza badawczego dotyczącego ewaluacji wydajności API składa się z następujących testów:

- testy linii bazowej - pojedynczy klient generuje żadania w kierunku API w celu zdefiniowania średniego czasu odpowiedzi usługi w standardowych warunkach jej działania. W ramach niniejszej pracy, podczas wykonywania omawianego testu, zdefiniowane zostaną ponadto wartości współczynników satysfakcji, toleracji oraz frustracji, będące składowymi wskaźnika jakości APDEX. Wartości te, stanowią uogólnioną ocenę wydajności i posłużą jako punkt odniesienia dla kolejnych testów.
- testy obciążeniowe - uwzględniana zostaje zmienna liczba klientów generujących żądania, w kontekście której ustalane są średnie czasu odpowiedzi na żądanie, a także dokonywane zostaje odniesienie uzyskanego rezultatu do współczynników zdefiniowanych uprzednio w ramach miary APDEX.
- testy przeciążające - liczba klientów generujących żądanie zostaje dobrana w taki sposób, aby doprowadzić do obciążenia testowanej usługi, niepozwalającego na poprawne funkcjonowanie interfejsu programowania aplikacji. Ewaluacja ta, ma na celu znalezienie punktu krytycznego w kontekście działania testowanego oprogramowania.

Poza czarnoskrzynkowymi testami wydajności, cechującymi się omówioną powyżej strkturą, przeprowadzone mogą zostać ewaluacje efektywności działania poszczególnych fragmentów

usługi sieciowej, w ramach których interfejs programowania aplikacji postrzegać należy w odmienny sposób, aniżeli jako system wejściowo-wyjściowy.

Przykładem oceny wydajności określonego fragmentu interfejsu programowania aplikacji może być ewaluacja modułu realizacji zaawansowanych operacji obliczeniowych, dostępnego z poziomu punktu końcowego API. W takim przypadku, oprogramowanie musi zostać dostosowane do przeprowadzenia procedury testowej, a metryka wydajności - powiązana z postępem realizacji obliczeń.

2.4. Wykorzystywane narzędzia i technologie

Zarówno w trakcie procesu implementacji badanych interfejsów programowania aplikacji, jak i procedurze przeprowadzenia badań pod kontem ich wydajności, wykorzystano obszerny zbiór sprawdzonych i powszechnie stosowanych rozwiązań technologicznych. W ramach niniejszej sekcji, opisane zostanie każde z nich.

C#

C# jest wieloparadygmatowym, a także nowoczesnym językiem programowania ogólnego przeznaczenia, charakteryzującym się bezpieczeństwem i niezawodnością w aspekcie typowania struktur danych. Pierwsza z wersji tego języka, stworzona została przez Andersa Hejlsberga w roku 1998. Od tamtej chwili, do momentu napisania niniejszej pracy, upublicznionych zostało 9 kolejnych, stabilnych wydań projektu C#. Każda z następnych wersji omawianego języka programowania wprowadzała zarówno usprawnienia w kontekście ekosystemu budowy i kompilacji programów źródłowych, jak i wzbogacała interfejs bibliotek funkcyjnych o kluczowe z punktu widzenia doświadczonego programisty rozwiązania. Do rozwiązań tych, zaliczyć należy między innymi: mechanizmy programowania współbieżnego, typy anonimowe, operatory zmiennych typów niezdefiniowanych, obsługę referencji, typy generyczne, czy też wyrażenia lambda.

W ramach niniejszej pracy, język C# wykorzystany został do implementacji jednego z dwóch zbiorów interfejsów programowania aplikacji. Ze względu na zastosowanie rozwiązań z zakresu przetwarzania współbieżnego (tj. operacji asynchronicznych oraz wielowątkowych) udostępnianych przez omawiany język programowania, interfejsy API realizowane w tej technologii mogą obsługiwać w sposób równoległy żądania pochodzące od wielu klientów, a także utrzymywać sekwencyjny charakter przetwarzanych procedur niezależnie od czasu ich wykonywania. Należy także zwrócić uwagę na mechanizm wewnętrznych usprawnień wydajnościowych implementowany w ramach kompilatora i uruchamiany w momencie tłumaczenia kodu języka do tzw. języka pośredniego *(ang. Intermediate Language)*. Dzięki zastosowaniu przedstawionego mechanizmu, operacje zdefiniowane przez programistę mogą być modyfikowane w procesie kompilacji, tak aby nie wpłynąć na zaimplementowaną funkcjonalność, zwiększając jednocześnie wydajność generowanego programu [12].

.NET Core

.NET Core postrzegać należy jako środowisko budowy, kompilacji oraz wykonywania rozwiązań implementowanych w języku C#. Przedstawiana technologia stanowi podzbiór bibliotek, dzięki którym programista jest w stanie budować systemy różnorodnego przeznaczenia, a także uruchamiać je w wielu wspieranych środowiskach programowych. W przeciwieństwie do technologii .NET Framework będącej poprzednikiem .NET Core, aplikacje tworzone na bazie omawianej biblioteki mogą być wydawane nie tylko na system operacyjny Windows, ale także na systemy Linux oraz MacOS.

W ramach omawianego środowiska wykorzystywany zostaje komponent języka C# zwany biblioteką standardową *(ang. .NET Standard Library)*. Biblioteka ta jest wspólna dla wielu środowisk uruchomieniowych, a zawarte w niej funkcjonalności, traktować należy jako metody ogólnego przeznaczenia.

Ponadto, środowisko .NET Core, w ramach procesu budowy i kompilacji rozwiązania nawiązuje komunikację z komponentem wspólnej infrastruktury *(ang. Common Infrastructure)*. Komponent ten, podobnie jak biblioteka standardowa, współdzielony jest przez wiele środowisk wykonawczych. W kontekście wspólnej infrastruktury, wspomnieć należy o wspólnej specyfikacji języka *(ang. CLS - Common Language Specification)*, wspólnym systemie typów *(ang. CTS - Common Type System)*, a także środowisku uruchomieniowym wspólnego języka *(ang. CLR - Common Language Runtime)*. Wykorzystanie między innymi tych trzech elementów, pozwala na budowę systemu dostępnego na wielu platformach [35].

W kontekście realizowanej pracy, technologia .NET Core użyta została jako środowisko uruchomieniowe dla interfejsów programowania aplikacji tworzonych w języku C#. W obrębie technologii tej, poza przedstawionymi powyżej komponentami, wyróżnić możemy natywną bibliotekę ASP.NET Core, stanowiącą zbiór metod przydatnych w procesie definiowania internetowych usług sieciowych oraz aplikacji webowych. Dzięki zastosowaniu ASP.NET Core operacje takie jak, między innymi: obsługa definicji kontrolerów API, zarządzanie stanem ciała żądania oraz jego rzutowaniem na określony typ danych, czy też implementacja mechanizmów uwierzytelniania i autoryzacji klienta, wykonane mogą zostać na wysokim poziomie abstrakcji z jednoczesnym zapewnieniem należytego poziomu ich wydajności.

Entity Framework Core

Entity Framework Core stanowi narzędzie stworzone przez firmę Microsoft, którego zastosowaniem jest mapowanie obiektowo-relacyjne realizowane w kontekście usług internetowych tworzonych z wykorzystaniem języka C# oraz uruchamianych na platformie .NET Core. Przedstawiana biblioteka zapewnia programiście zorientowany obiektowo interfejs, za pomocą którego może on uzyskać dostęp do danych, a także je definiować oraz przetwarzać. Zbiory obiektów mogą być składowane zarówno w relacyjnych jak i niereleacyjnych bazach danych. Niniejsza biblioteka, podobnie do środowiska uruchomieniowego .NET Core, jest rozwiązaniem wieloplatformowym i może być wykorzystywana przy budowie systemów internetowych wdrażanych na systemach Windows, Linux oraz MacOS.

Zastosowanie biblioteki mapera obiektowo-relacyjnego jaką jest Entity Framework Core umożliwia zastosowanie podejścia zorientowanego na kod źródłowy w kontekście aplikacji komunikujących się i wykorzystujących zewnętrzne zbiory danych *(ang. Code-First Approach)*. Podejście to, polega na definiowaniu w ramach kodu źródłowego zbioru klas modelu danych, które będą następnie przekształcane do postaci tabel określonego systemu bazodanowego. Przedstawiona operacja przekształcenia wykonywana jest bezpośrednio za pomocą mechanizmów mapera obiektowo-relacyjnego.

Niezaprzeczalną zaletę wykorzystania biblioteki ORM jaką jest Entity Framework Core stanowi możliwość operowania na jednolitym interfejsie realizacji operacji na danych, niezależnie od obsługiwanego systemu bazodanowego. Oznacza to, że w momencie zmiany dostawcy zewnętrznego źródła danych, zawartość kodu źródłowego programu nie musi podlegać modyfikacji [31].

MediatR

MediatR to otwartoźródłowa biblioteka języka C#, z wykorzystaniem której zaimplementowany może zostać wzorzec projektowy, dotyczący separacji odpowiedzialności za obsługę zapytań

oraz komend przetwarzanych przez usługę sieciową. Kluczowym elementem biblioteki MediatR jest para generycznych interfejsów, za pomocą których implementowana jest obsługa zarówno żądań jak i zapytań dotyczących danych. Interfejsami tymi są kolejno: *IRequest* - struktura programistyczna implementowana przez klasy definiujące zawartość ciała żądania lub komendy, a także powiązany z nią *IRequestHandler*, który jest konkretyzowany przez klasę definicji metody obsługi żądania bądź operacji na danych.

Ponadto, należy również podkreślić znaczenie metody *Send* dostępnej w ramach głównego API pakietu MediatR. Dzięki niej, wywołana może zostać procedura obsługi określonego żądania lub operacji, z dowolnego miejsca kodu źródłowego interfejsu programowania aplikacji [4].

JavaScript

JavaScript to wielofunkcyjny oraz wieloplatformowy skryptowy język programowania cechujący się wysokim poziomem abstrakcji. Najbardziej popularnym przeznaczeniem omawianego języka jest budowa systemów internetowych, a także mobilnych. Historyczną rolą technologii JavaScript było udostępnianie programiście funkcjonalności umożliwiających określanie różnorodnych sposobów interakcji pomiędzy użytkownikiem serwisu internetowego, a jego statycznymi elementami. Podstawowym środowiskiem wykonania oraz interpretacji omawianego języka była uwcześnie przeglądarka internetowa. Wraz z pojawieniem się serwerowego środowiska uruchomieniowego NodeJS, przeznaczonego dla języka JavaScript, popularność omawianej technologii wzrosła w gwałtownym tempie. Zmianie uległo również główne przeznaczenie technologii, która od tej pory stała się pełnoprawnym językiem programowania, stosowanym w kontekście budowy zarówno systemów internetowych, rozwiązań mobilnych, jak i programów desktopowych.

Język JavaScript uznać należy za technologię charakteryzującą się typowaniem słabym oraz dynamicznym. W związku z zastosowaniem przez twórców rozwiązania takiego właśnie podejścia, tworzone kody programów narażone są na występowanie zjawisk niezgodności typów, a także niejawnej koercji. Ponadto, w kontekście mechanizmów omawianego języka, realizacja operacji przetwarzania współbieżnego oraz wykonania metod asynchronicznych, zależna jest w całkowitym stopniu od rozwiązań implementacyjnych poczynionych w ramach środowiska uruchomieniowego. Oznacza to, że przetwarzanie i wykonywanie operacji wielowątkowych może cechować się zróżnicowaną wydajnością, w zależności od konkretnego interpretera języka.

Niewątpliwymi zaletami technologii JavaScript są: składnia cechująca się niskim poziomem złożoności poleceń, możliwość dowolnego wykorzystywania wielu spośród wspieranych paradygmatów programowania, modułowość i skalowalność implementowanych rozwiązań, a także elastyczność w kontekście operowania na wykorzystywanych strukturach danych [17].

W ramach niniejszej pracy, język JavaScript zastosowany został w celu implementacji jednego z dwóch zbiorów badanych interfejsów programowania aplikacji. Tworzone w omawianym języku API, wykonywane będą w środowisku uruchomieniowym NodeJS.

TypeScript

TypeScript stanowi statycznie typowany nadzbiór języka JavaScript. Określenie to, oznacza że omawiana technologia nie jest stricte językiem programowania, a tylko określoną grupą instrukcji oraz procedur, które włączyć można do języka JavaScript, po to, aby zapewnić w jego kontekście statyczny sposób typowania danych. Technologia TypeScript nie może być wykorzystywana samodzielnie, a środowisko wykonawcze JavaScript jest wymagane w celu uruchomienia skompilowanego modułu, definiowanego zgodnie ze składnią omawianego języka.

Kluczowym elementem przedstawianej technologii jest transpilator języka TypeScript o nazwie tsc *(ang. TypeScript Compiler)*. Program ten, uruchamiany jest tuż przed rozpoczęciem procedury interepretacji kodu JavaScript i przekształca on metody odpowiedzialne za obsługę typów danych, do struktur dostępnych w ramach standardowej implementacji języka. Dlatego też, z punktu widzenia środowiska uruchomieniowego, dostępne programistom mechanizmy definicji typów czy interfejsów, nie są znane.

Celem zastosowania omawianego nadzbioru językowego jest możliwość kontroli zgodności definiowanych obiektów programistycznych pod kątem ich wewnętrznej struktury. Ponadto, wykorzystanie TypeScript umożliwia weryfikację faktu nieumyślnego odwołania się do struktury typu nieokreślonego, jeszcze przed rozpoczęciem procesu interpretacji kodu [3].

NodeJS

NodeJS jest środowiskiem uruchomieniowym języka JavaScript zbudowanym w oparciu o otwartoźródłowy silnik interpretacji kodu Chrome V8. Dzięki zastosowaniu omawianego środowiska uruchomieniowego, kod źródłowy języka JavaScript może być wykonywany poza ekosystemem przeglądarki internetowej. Rozwój niniejszej technologii, doprowadził do diametralnej zmiany w obszarze zastosowania języka JavaScript, a także gwałtownego wzrostu jego popularności w kontekście budowy systemów internetowych.

Podobnie do rozwiązania Microsoft .NET Core, platforma NodeJS składa się nie tylko ze środowiska uruchomieniowego, ale także ze zbioru bibliotek oraz narzędzi linii komend. Aplikacje budowane na bazie omawianej technologii cechują się zastosowaniem architektury sterowanej zdarzeniami *(ang. Event-driven architecture)*, która ponadto wzbogacana jest (dzięki wykorzystaniu mechanizmu pętli) o możliwość obsługi operacji asynchronicznych. Co więcej, rozwiązania definiowane na podstawie środowiska NodeJS posiadają budowę modułową, co przyczynia się do zwiększenia ich zdolności w kontekście skalowania systemów.

Należy również uwypuklić generyczność charakterystyki środowiska NodeJS. Nie jest ono przeznaczone stricte do definiowania i wdrażania usług sieciowych, a wykorzystywane jest do uruchamiania dowolnego kodu języka JavaScript, jaki może zostać stworzony za pomocą jego składni. Dlatego też, aby dostarczać mechanizmy dotyczące specyficznych funkcjonalności, ekosystem NodeJS może być rozbudowywany poprzez otwartoźródłowe moduły. Liczność modułów tych, a także popularność ich wykorzystania stanowi niewątpliwie o sile omawianej technologii [6].

ExpressJS

ExpressJS stanowi bibliotekę środowiska NodeJS, dostarczającą zbiór metod pozwalających na budowę webowych interfejsów programowania aplikacji w tym właśnie środowisku. Pakiet ExpressJS cechuje się minimalizmem w kontekście złożoności udostępnianych programiście operacji, elastycznością dotyczącą współpracy z zewnętrznymi pakietami, a także wysoką wydajnością działania tworzonych aplikacji, poprzez eliminację złożonych funkcjonalności przetwarzania zasobów w obrębie żądania.

Koncepcja przetwarzania zasobu uzyskanego od klienta w ramach ExpressJS sprowadza się do potokowej obsługi dostarczonego wejścia, przez kolejne funkcje pośredniczące *(ang. Middleware functions)*. Ostatnia z funkcji ma za zadanie zwrócić odpowiedź wygenerowaną na podstawie operacji wykonywanych przez wszystkie poprzednie metody. Ciałem funkcji pośredniczącej może być zarówno kod źródłowy zdefiniowany przez programistę, jak i ten dostarczony poprzez referencję do zewnętrznego pakietu. Do każdej z metod przekazywane są parametry żądania, odpowiedzi, a także referencji do następnego middleware. Dzięki uruchomieniu napi-

sanej w taki sposób aplikacji w środowisku NodeJS, poszczególne funkcje pośredniczące mogą być realizowane asynchronicznie [21].

Prisma

Prisma ORM to narzędzie pełniące rolę mapera obiektowo-relacyjnego wykorzystywanego w kontekście implementowania interfejsów programowania aplikacji w języku JavaScript. Dystynktywną cechą omawianego narzędzia jest prostota definiowania rzutowanego modelu danych. W przypadku pakietu Prisma, cała struktura modelu danych opisywana jest w ramach jednego pliku zwanego plikiem schematu. W pliku tym, określane zostają zarówno właściwości poszczególnych encji modelu, jak i relacje które między nimi występują.

Analogicznie do narzędzia Entity Framework Core, Prisma ORM wspiera zarówno relacyjne jak i nierelacyjne systemy bazodanowe. Ponadto, zauważyć należy kompatybilność omawianego narzędzia z omawianą wyżej technologią TypeScript [27].

Mongoose

Biblioteka Mongoose stanowi narzędzie rzutowania obiektowego modelu danych definiowanego w ramach kodu programu, do postaci obiektowej nierelacyjnej bazy danych MongoDB. Technologii tej nie należy nazywać maperem obiektowo-relacyjnym, gdyż wykonywane przez nią operacje synchronizują struktury danych, które zarówno po stronie kodu, jak i po stronie źródła danych cechują się obiektową naturą.

Połączenie interfejsu programowania aplikacji z nierelacyjnym źródłem danych obsługiwanym poprzez bibliotekę Mongoose, prowadzić może do zwiększenia efektywności oraz zmniejszenia czasu wykonania operacji na danych. Jednakże, w związku z naturą przechowywanych informacji, a także brakiem uwzględniania w ich strukturze metadefinicji, zastosowanie omawianego mechanizmu może również prowadzić do braku spójności źródła danych, a także niemożliwości zastosowania usprawnień wydajnościowych w kontekście bazy danych [14].

Apache JMeter

Narzędzie Apache JMeter to otwartoźródłowe oprogramowanie stworzone w języku Java. Program ten, wykorzystywany jest przeprowadzania ewaluacji wydajności oprogramowania sieciowego opierającego swoje działanie o protokoły HTTP oraz FTP. Testy efektywności działania usług mogą zostać przeprowadzane w trybie lokalnym (tj. z wykorzystaniem jednego hosta wysyłającego żądania do określonej usługi sieciowej), jak i w trybie rozproszonym (tj. budowana zostaje hierarchia hostów będących generatorami żądań). W ramach niniejszej pracy, zastosowany został drugi z przedstawionych trybów ewaluacji.

Działanie oprogramowania Apache JMeter sprowadza się do wykonywania pętli testowej, w ramach określonych grup wątków. Pętla testowa symuluje sekwencyjne generowanie żądań w kierunku serwera, z uwzględnieniem stałej wartości opóźnienia pomiędzy wysyłanymi pakietami. Grupa wątków natomiast, określa współbieżny charakter testów obciążenia usługi sieciowej i może być utożsamiana zarówno z konkretnymi wątkami procesora lokalnego hosta, jak i z oddzielnie pracującymi generatorami żądań w trybie rozproszonym.

W celu zdefiniowania testu wydajności w ramach Apache JMeter, zbudowany powinien zostać plan testowy. Jest to podstawowa jednostka wyróżniana w ramach niniejszego oprogramowania i skupia ona w sobie między innymi komponenty grup wątków, próbników *(ang. Samplers)*, a także elementów nasłuchujących na odpowiedź usługi *(ang. Listeners)*. Zarówno próbniki, jak i elementy nasłuchujące mogą być powielane w ramach planu testowego, a także indywidualnie konfigurowane, w zależności od specyfiki usługi sieciowej.

Oprogramowanie Apache JMeter obsługiwać można zarówno z poziomu narzędzia linii komend, jak i udostępnionego graficznego interfejsu użytkownika [11].

2.5. Przegląd literatury

W niniejszym rozdziale przedstawione zostaną pozycje literaturowe, do których odnosić się będzie opisywana praca dyplomowa. Pozycje te, podzielone zostały na oddzielne grupy, związane z określoną tematyką.

Na początku, przedstawiona zostanie literatura powiązana z aspektem budowy interfejsów programowania aplikacji oraz będąca wprowadzeniem do wykorzystywanych technologii. Następnie, opisane zostaną pozycje traktujące o wydajności interfejsów API, a także o analizie działania powszechnie dostępnych serwisów internetowych opartych o metodologię REST. Kolejne prace, skupiać się będą na tematyce testowania usług sieciowych, teorii testowania, a także konfiguracji narzędzi dla testów rozproszonych. W następnej kolejności, wspomniane zostaną prace naukowe oraz dokumenty standaryzacyjne dotyczące sposobu działania protokołu przesyłania danych hipertekstowych. Ostatnią grupą pozycji literaturowych będą prace referencyjne dotyczące badań wydajności systemów internetowych.

Pozycja [35] stanowi wprowadzenie do zaawansowanych konceptów języka C#, a także dostarcza informacji związanych z wykorzystaniem tego języka w środowiskach uruchomieniowych .NET oraz .NET Core. W początkowych rozdziałach przedstawiono sposób budowy, kompilacji oraz wykonywania programu w środowisku .NET. Kolejno opisana została struktura bazowych aplikacji uruchamianych w tym właśnie środowisku i tworzonych za pomocą języka C#. Ostatnim elementem wprowadzenia do opisywanej technologii było przedstawienie struktur języka w kontekście obiektowego paradygmatu programowania. W następnych sekcjach literatury, w sposób wyczerpujący poruszono tematykę bardziej zaawansowanych aspektów programowania w języku C#, którymi są między innymi: kolekcje i typy generyczne, delegaty i wyrażenia lambda, czy też cykl życia obiektu w pamięci programu. Ważnym tematem, poruszonym w ramach tej książki jest struktura oraz zasada działania środowiska .net core, będącego podstawowym elementem interfejsów programowania aplikacji tworzonych w języku C#.

Analogiczną do przedstawionej powyżej pozycji literaturowej, dotyczącą jednak technologii NodeJS oraz języka JavaScript jest [6]. W ramach tej pracy zawarto obszerne wprowadzenie do platformy NodeJS uwzględniające ponadto kwestie obsługi operacji wejścia/wyjścia, wykonywania natywnego kodu JS, czy też przetwarzania operacji przez silnik NodeJS oraz bibliotekę libuv. Znaczna część pracy, obejmuje przedstawienie zaawansowanych wzorców projektowych, których głównym przeznaczeniem jest obsługa zdarzeń oraz operacji asynchronicznych. Wspomniane zostały także rozwiązania dotyczące skalowalności aplikacji z wykorzystaniem mechanizmów kolejkowania wiadomości.

Niezależnie od wykorzystywanej technologii, interfejsy programowania aplikacji, które zostały zbudowane na potrzeby tej pracy dyplomowej, oparte są o styl architektoniczny RESTful. Styl ten, jest pewnym zbiorem zasad projektowania usług sieciowych, określającym zarówno aspekty sposobu komunikacji klienta z usługą sieciową, jak i techniczne wymagania dotyczące przetwarzania żądań. Dobre praktyki, które uwzględnia metodologia REST, zawarte zostały w pozycji literaturowej [33]. Autorzy tego dokumentu, na wstępie dokonują porównania architektury zorientowanej na zasoby, będącej podstawą konwencji REST, z popularną uprzednio architekturą zorientowaną na usługi. Następnie, przedstawiane są najlepsze praktyki, cele oraz reguły REST dotyczące projektowania interfejsu programowania aplikacji. Co więcej, w omawianej książce zawarte zostały także podstawowe oraz zaawansowane wzorce projektowania API, uwzględniające aspekty bezstanowości, paginacji, osiągalności, a także identyfikacji zasobów interfejsu. Końcowe rozdziały książki, wprowadzają w kwestie testowania oraz bez-

pieczeństwa REST API, omawiają technikę kompozycji usług RESTful, a także przedstawiają rozwiązania (biblioteki oraz języki programowania) pozwalające na tworzenie interfejsów API zgodnych z metodologią REST.

Podstawowym celem działania interfejsu programowania aplikacji jest dostarczenie danych do konsumenta, bądź też ich manipulacja zgodnie z jego żądaniem. Aby operować na danych, interfejs API musi komunikować się ze źródłem danych, którym najczęściej jest serwer bazodanowy. W celu dostarczenia metod komunikacji pomiędzy API a źródłem danych, które jednocześnie są niezależne od wykorzystywanego źródła, a także pozwalają na zarządzanie danymi z poziomu struktur języka, stworzone zostały biblioteki zwane maperami obiektowo-relacyjnymi (ang. Object-Relational Mappers). Dla API napisanego w języku C# podstawowym rozwiązaniem ORM jest biblioteka Entity Framework Core, która przedstawiona została w pozycji [31]. Pozycja ta, uwzględnia zarówno opis działania najczęściej wykorzystywanych metod służących do manipulacji danymi, jak i rolę klasy kontekstu bazodanowego w procesie tłumaczenia operacji programistycznych na polecenia bazodanowe. Ponadto, dowiedzieć możemy się jak przetwarzać zaawansowane typy danych (takie jak np. DateTime), czy też w jaki sposób wykorzystywać zapytania LINQ do budowania kwerend.

Dla interfejsu programowania aplikacji napisanego w języku JavaScript i uruchamianego w środowisku NodeJS, w przeciwieństwie do platformy .NET, zastosować możemy zdecydowanie większą liczbę bibliotek pełniących rolę maperów obiektowo-relacyjnych. Biblioteki te, zostały opisane w pozycjach [5] i [20]. Pozycja [5] pełni rolę całościowego wprowadzenia do tematyki tworzenia interfejsów API, korzystając z platformy NodeJS, frameworka ExpressJS oraz nierelacyjnej bazy danych MongoDB. Rodział piąty tej pracy, traktujący o wykorzystaniu baz danych NoSQL, przybliża tematykę jednego z najczęściej wykorzystywanych maperów obiektowo-relacyjnych dla Node czyli mongoose. Przedstawiono tutaj sposób zestawienia połączenia z serwerem bazodanowym, tworzenia encji modelu, przekształcanego następnie na struktury bazy danych, a także wykonywania operacji dostępu do danych i ich modyfikacji. W pracy [20] natomiast, porównano nierelacyjne podejście do składowania danych typu geograficznego z podejściem relacyjnym, wykorzystując w tym przypadku biblioteki mongoose i sequelize. Oba mapery obiektowo relacyjne zostały użyte w ramach interfejsu API wykorzystującego technologie NodeJS/ExpressJS. Celem opisywanej pracy było przedstawienie różnic w czasach odpowiedzi API na uzyskane żądanie, dla różnej liczby danych geolokalizacyjnych, uwzględniając zastosowanie relacyjnych i nierelacyjnych baz danych.

Następne pozycje literaturowe, związane są z analizą usług REST oraz wydajnością webowych interfejsów programowania aplikacji.

Pozycja [24] stanowi analizę 500 serwisów internetowych z listy alexa.com4000 najpopularniejszych dostępnych publicznie usług sieciowych. Twórcy każdego z 500 serwisów deklarują zgodność swoich produktów z konwencją REST. Przeprowadzona analiza dotyczyła kluczowych aspektów technicznych związanych z funkcjonowaniem API, stopnia zgodności API z regułami dotyczącymi metodologii REST, a także przestrzegania najlepszych praktyk projektowania interfejsów programowania aplikacji, takich jak m.in. zastosowanie mechanizmu wersjonowania. W trakcie analizy, zaobserwowano określone trendy dla aplikacji REST API, takie jak m.in. rozpowszechnione wsparcie notacji JSON, czy wykorzystywanie narzędzi do dokumentacji generowanej programowo. Ponadto, zauważono, że tylko ok. 0.8% analizowanych serwisów webowych przestrzega w sposób ścisły reguł zawartych w ramach konwencji REST.

Wydajność interfejsów programowania aplikacji, jako jeden z elementów miary jakości API została przedstawiona w pozycji [2]. Na początku pracy, jej autorzy wskazują na interakcję interfejsu programowania aplikacji z systemami klienckimi. Opisany został tutaj zestaw protokołów sieciowych wykorzystywanych podczas formułowania i transmisji żądania, system zunifikowanych lokacji zasobów, a także semantyka interakcji w zależności od wykorzystywanych typów żądań protokołu hipertekstowego. Ponadto, wskazano najczęstsze przyczyny błędów przepływu

danych dla http, uwzględniając działanie usługi DNS, błędy połączenia, błędy leżące po stronie klienta, a także błędy wynikające z działania serwera. Kolejna część pracy, związana jest ze składowymi metryki jakości, do których według autorów, poza wydajnością, zaliczyć możemy: dostępność, procent żądań dla których uzyskano pozytywną odpowiedź, osiągalność, a także możliwość sprawdzenia stanu usługi w dowolnym momencie jej działania. Dodatkowo, w niniejszej pracy zaproponowano podejście oraz zestaw narzędzi pozwalających na dokonanie ewaluacji jakości interfejsu programowania aplikacji, zgodnie z przyjętą normą jakości.

Kolejnym etapem następującym po zdefiniowaniu metryki wydajności, jest ustalenie wartości tejże metryki w kontekście testowanych usług sieciowych. Przytoczone poniżej pozycje literaturowe, związane są z wykonywaniem pomiarów wydajności API, czyli testowaniem.

Pozycja [32] stanowi obszerne wprowadzenie do teorii testowania oprogramowania. W pierwszych rozdziałach tego dokumentu, wyjaśniono czym jest testowanie, dlaczego jest ono niezbędne podczas tworzenia oprogramowania, a także jak wygląda podstawowy proces wykonywania testów. Następnie przedstawiono proces testowania w kontekście tworzenia oprogramowania. Uwzględniono tu zarówno modele cyklu życia rozwoju systemów w powiązaniu z testowaniem, poziomy realizowanych testów, ich typy, jak i sposoby zarządzania testami. Kolejne rozdziały tyczą się testowania statycznego (tj. testowania funkcjonalności lub modułu na poziomie jego specyfikacji lub implementacji bez wykonywania kodu testowanego oprogramowania), dostarczają teorii związanej z poszczególnymi technikami testowania rozwiązań oraz przedstawiają aspekt organizacji, planowania, monitorowania oraz uwzględniania ryzyka w czasie dokonywania ewaluacji systemów. W ostatnim z rozdziałów dokumentu, autorzy przedstawiają narzędzia przydatne w procesie testowania, a także sposób ich efektywnego wykorzystania w codziennej pracy.

Pozycja [28] zawiera wiele analogicznych treści do pracy opisanej powyżej, jednakże rozwija ona w sposób wyczerpujący, wspomniane tylko w poprzedniej pracy aspekty. W części drugiej dokumentu zawarto dogłębną analizę zagadnienia testowania statycznego, uwzględniając m.in. testowanie zgodności ze standardami oprogramowania, symboliczne wykonywanie kodu, a nawet wprowadzając aparat matematyczny do formalnego dowodzenia poprawności fragmentów oprogramowania. W ramach tej książki, przedstawiono także dynamiczną analizę systemu (tj. testowanie funkcjonalności lub modułu na poziomie wykonywanego kodu) uwzględniając często występujące błędy związane m.in. z nieumiejętnym zarządzaniem strukturami pamięci programu. Ponadto, uwzględniono zagadnienie priorytetyzacji przypadków testowych, wprowadzając pojęcie miary średniego procenta wykrytych usterek. Autor dokumentu przedstawia także testowanie charakterystyk jakościowych zgodnie z normą ISO 9126 oraz ISO 25010, tworzenie dokumentacji w ramach zarządzania testowaniem, czy chociażby zarządzanie incydentami występującymi w ramach procesu ewaluacji oprogramowania.

W ramach pozycji [23], dowiedzieć możemy się ponadto o testowaniu usług internetowych. Przedstawiono tutaj podstawową strukturę standardowej usługi sieciowej (w tym przypadku – usługi e-commerce) cechującej się architekturą trójwarstwową. Ponadto, wyjaśniono rolę każdej z warstw systemu, a także przedstawiono aspekty testowania oprogramowania w kontekście każdej z nich. Dodatkowo, zawarte zostały przykładowe przypadki testowe, dotyczące zarówno prezentacji danych w systemie, jak i dostępu do danych poprzez serwer webowy. Dla zaprezentowanych przypadków testowych, przedstawione zostały także scenariusze realizacji testów w postaci listy czynności jakie należy podjąć, aby dokonać ewaluacji systemu.

Aspekty technologii testowania oprogramowania ujęte zostały także w pozycji [22]. Artykuł ten, stanowi sekcję wprowadzającą do książki pt. Tutorial: Software Testing and Valdation Techniques, tego samego autora. Pozycja ta, przedstawia przekrój technik oraz technologii testowania oprogramowania wykorzystywanych na przestrzeni ostatnich ok. 30 lat. Opisane zostały tutaj zarówno teoretyczne podstawy testowania, narzędzia i techniki analizy statycznej i dynamicznej, oceny efektywności przeprowadzanych testów, a także badania przeprowadzane

w dziedzinie testowania i walidacji oprogramowania. Omawiany artykuł, wyszczególnia pozytywne oraz negatywne aspekty poszczególnych technik oraz wskazuje przydatność określonych rozwiązań do testowania oprogramowania różnego typu.

Ostatnią przytoczoną w ramach tego przeglądu literaturowego pozycją, dotyczącą teorii ewaluacji oprogramowania jest [16]. Pozycja ta, stanowi normę międzynarodowej organizacji normalizacyjnej (ang. International Organization for Standardization) dotyczącą weryfikacji jakości oprogramowania. Uwzględniono tu przede wszystkim znaczenie pojęć stosowanych w dziedzinie testowania oprogramowania, wprowadzono definicje dla określonych terminów oraz zjawisk występujących w ramach ewaluacji systemów, a także określono zgodność wprowadzanych przez standard konceptów, z konceptami zawartymi w standardach pochodnych. Główną część dokumentu, stanowi wprowadzenie szkieletu modelu jakości, uwzględniającego określone modele jakościowe, modele jakości w użyciu, a także modele jakości produktu. Dodatkowo, przedstawiono cel oraz sposób wykorzystania modeli jakościowych, wyjaśniono różnicę w postrzeganiu modeli jakościowych z punktu widzenia różnych interesariuszy, a także zdefiniowano relacje pomiędzy określonymi modelami. Dokument ten, wraz z normą ISO 9126, stanowią definicję pojęcia jakości w kontekście testowania oprogramowania.

Pozycja [26] stanowi przegląd narzędzi wykorzystywanych do testowania działania systemów komputerowych. Na początku książki, wprowadzany jest termin zapewnienia jakości (ang. Quality Assurance), który w dzisiejszych czasach definiuje zakres odpowiedzialności osoby testującej oprogramowanie. Kolejno, przedstawiane są kryteria sukcesu dotyczące tworzonego systemu, a także fazy poszczególnych modeli rozwoju oprogramowania zorientowanych na procesy. Analogicznie do pozycji literaturowych przedstawionych uprzednio, w ramach tej pozycji określone zostały metryki i definicje jakości oprogramowania oraz omówiony został proces realizacji testów. Główna część omawianego dokumentu skupiona jest wokół narzędzi stosowanych do realizacji ewaluacji oprogramowania. Wyszczególniono tutaj narzędzie WinRunner, przedstawiając między innymi wykorzystywany w tym programie skryptowy język testów (ang. Test Script Language). Ponadto, przedstawiono architekturę oraz najważniejsze funkcjonalności narzędzi SilkTest, SQA Robot, LoadRunner, TestDirector, QuickTest Professional a także Apache JMeter. Ostatni z wymienionych programów, wykorzystywany zostanie w ramach niniejszej pracy dyplomowej, dlatego też dalszy przegląd tej pozycji literaturowej skupiony będzie na rozdziale dotyczącym właśnie tego narzędzia. Opis funkcjonalności aplikacji JMeter został w niniejszej pozycji podzielony na sekcje związane z testowaniem rozwiązań bazodanowych wykorzystujących interfejs JDBC (ang. Java DataBase Connectivity), a także sekcję dotyczącą testowania aplikacji bazujących w swoim działaniu na protokole hipertekstowym. Przedstawiono tutaj sposób tworzenia grup wątków reprezentujących użytkowników aplikacji, generowania żądania protokołu hipertekstowego, uruchomienia mechanizmu nasłuchiwania na odpowiedź serwisu, dodawania licznika czasu, a także zapisywania i przeglądania rezultatów przeprowadzonego testu.

W ramach dokumentów [25] oraz [11] przedstawiono pełen zakres funkcjonalności dostępnych w ramach narzędzia Apache JMeter. Pierwsza z prac (tj. [25]), skupia się na wykorzystaniu narzędzia w celu wykonywania testów wydajności usług sieciowych, natomiast druga z pozycji (tj. [11]), przedstawia aplikację JMeter dla różnych kontekstów jej potencjalnego użycia. W obu pracach wyszczególnione zostają podstawowe elementy, na które składa się środowisko testowe. Elementami tymi są: grupy wątków, komponenty próbkujące, kontrolery, komponenty nasłuchujące, liczniki czasu oraz asercje. Ponadto, omówiono elementy graficznego interfejsu użytkownika dla aplikacji, przedstawiono proces instalacji oraz uruchamiania narzędzia JMeter, a także zdefiniowano pojęcie planu testów. W kontekście pracy [25], poza wymienionymi uprzednio kwestiami, zobrazowany został także proces wykonywania testu przeciążeniowego dla usługi zorientowanej na serwisy (ang. Service-Oriented Application). Proces ten uwzględniał: tworzenie grupy wątków, konfigurację struktury żądania wysyłanego do usługi, uruchomienie

testu, a także pozyskanie wyniku. W pracy [11] natomiast, analogiczny proces, możemy zaobserwować dla monolitycznej aplikacji internetowej oraz interfejsu programowania aplikacji. Ponadto, przedstawione zostały zaawansowane opcje konfiguracji elementów nasłuchujących oraz liczników czasu, a także pokazany został proces wykorzystania pośredniczącego serwera http, w celu dokumentowania realizowanych żądań.

Następne pozycje literaturowe omówione w ramach tej pracy, dotyczą budowy oraz zasady działania internetowego protokołu hipertekstowego (ang. Hypertext Transfer Protocol), a także implementacji mechanizmu zarządzania stanem. Mechanizm ten, w związku z naturą protokołu http, nie jest w nim domyślnie realizowany.

Pozycja [8] stanowi techniczny dokument dotyczący semantyki oraz budowy internetowego protokołu hipertekstowego w wersji 1.1. Zdefiniowano w nim pojęcie zasobu żądania oraz omówiono cykl życia jego przetwarzania. Wskazano także moment, w którym zasób rekonstruowany jest przez serwer na podstawie jego efektywnego identyfikatora URI (ang. Uniform Resource Identifier). Ponadto, nakreślono pojęcie reprezentacji danych przesyłanych za pomocą protokołu http, definiując określone pola nagłówkowe dotyczące: typu danych, sposobu kodowania, języka danych, a także lokalizacji zasobu. Kolejne rozdziały dokumentu zawierają informacje dotyczące definicji dozwolonych metod protokołu http oraz znaczenia jakie te metody wprowadzają w kontekście operacji na zasobie. W dokumencie przedstawiono także kody statusu odpowiedzi na żądnie, grupując je w sposób semantyczny. Dla każdego z przedstawionych kodów statusu nakreślono kontekst, w jakim odpowiedź, oznaczona tym właśnie kodem, powinna być zwracana klientowi. Na końcu pracy, omówiono kwestie związane z bezpieczeństwem protokołu takie jak: ataki bazujące na wstrzykiwaniu kodu czy ochrona przed ujawnianiem informacji wrażliwych w identyfikatorach zasobów.

Pozycja [10] pozwala na poszerzenie wiedzy dotyczącej protokołu hipertekstowego w bardziej praktycznym kontekście. Podobnie jak w dokumencie [8], przedstawiono tutaj informacje teoretyczne dotyczące architektury protokołu, definicji zasobów czy też ujednoliconego formatu ich adresowania. Ponadto, wskazano i scharakteryzowano określone typy połączeń realizowanych z wykorzystaniem protokołu hipertekstowego. Co więcej, dla każdego z nich rozważono kwestie związane z wydajnością połączenia pomiędzy klientem a serwerem. Kolejne rozdziały pracy [10] traktują o identyfikacji klienta w ramach serwera, jego uwierzytelniania przed serwerem, a także szyfrowania danych przesyłanych pomiędzy tymi dwiema jednostkami. W niniejszej pracy wspomniano także o internacjonalizacji żądań w kontekście zastosowania nagłówka ‘Accept-Language’. Ostatnie rozdziały dokumentu dotyczą kwestii publikowania i dystrybucji zawartości. Wyszczególnione zostały tu takie elementy jak: web hosting, systemy publikacji treści, czy też mechanizm przekierowań oraz równoważenia obciążeń.

Zgodnie z charakterystyką protokołu http, realizuje on komunikację w sposób bezstanowy. Oznacza to, że domyślnie, pomiędzy klientem a serwerem nie jest utrzymywana sesja połączeniowa, a każde żądanie generowane przez klienta w kierunku serwera rozpatrywane jest indywidualnie. Rozwiązanie takie, pozwala na znaczące przyspieszenie działania protokołu hipertekstowego, a także uproszczenie jego konstrukcji. Jednakże, szczególnie w przypadku aplikacji internetowych komunikujących się z serwerem http, bezstanowy charakter protokołu bywa problematyczny w aspekcie kontekstu wysyłanych sekwencyjnie żądań. Dlatego też, do protokołu http wprowadzono mechanizm zarządzania stanem opisany w dokumencie [1]. Dokument ten, definiuje pola nagłówkowe o nazwach ‘HTTP Cookie’ oraz ‘Set-Cookie’. Pola te, mogą być używane przez serwery http w celu przechowywania stanu w ramach aplikacji klienckich, dając serwerom tym możliwość zarządzania, zawierającą stan sesją, przy wykorzystaniu protokołu bezstanowego. W niniejszym dokumencie, dla obu przedstawionych pól wyszczególniono atrybuty składowe pola, a także określono znaczenie każdego z nich. Ponadto, dokument definiuje wymagania dla klienta http, dotyczące możliwości wykorzystania mechanizmu zarządzania stanem. Pod uwagę wzięte zostały także kwestie bezpieczeństwa takie jak identyfikatory sesji, słaba

poufność danych, czy też zaufanie do usługi nazw domenowych w celu prawidłowego działania mechanizmu zarządzania stanem.

Ostatnia grupa pozycji literaturowych, zawartych w ramach niniejszego przeglądu literaturowego dotyczy badań związanych z testowaniem wydajności aplikacji internetowych w środowisku rozproszonym. Pozycje przedstawione poniżej, będą stanowić prace referencyjne względem niniejszej pracy dyplomowej.

Artykuł [15] dotyczy porównania wydajności działania interfejsów programowania aplikacji tworzonych z wykorzystaniem platform .NET Core 3.1 oraz .NET 5. Celem powstania tego dokumentu była weryfikacja zjawiska wzrostu wydajności działania programów, tworzonych i uruchamianych z wykorzystaniem nowszej z platform firmy Microsoft. Praca ta, ma także na celu pomóc pozwolić odpowiedzieć na pytanie, czy kod źródłowy interfejsu programowania aplikacji o określonych funkcjonalnościach, a także korzystający z określonych narzędzi, powinien zostać zaktualizowany w taki sposób, aby wspierać najnowszą, stabilną wersję środowiska .NET. W ramach dokumentu, w celu realizowania pomiarów wydajności wykorzystano opisane w poprzednich akapitach narzędzie Apache JMeter, a także dedykowaną środowisku .NET, bibliotekę BenchmarkDotNet. Kolejne rozdziały artykułu przedstawiają przygotowane środowisko testowe, plan wykonywanych testów, a także uzyskane rezultaty wraz z ich analizą. Autor pracy, zobrazował wyniki sześciu testów wydajnościowych, biorących pod uwagę proces serializacji oraz deserializacji obiektów typu JSON za pomocą bibliotek NewtonsoftJson, a także System.Text.Json. Ponadto, przygotowany został test wyszukiwania wzorca z obszernym ciągu tekstowym oraz test wykorzystania punktu końcowego jako klienta zewnętrznego API. Na podstawie otrzymanych rezultatów, wnioskować możemy o około 24 procentowym średnim wzroście wydajności wykonywania operacji realizowanych w ramach testów. Ponadto, wykazano także dość znaczący (około 35 procentowy) średni spadek wydajności nowego rozwiązania względem poprzednika, w kontekście testów obciążeniowych.

Analogiczne badania przeprowadzono w ramach pracy [38]. W tym przypadku jednak, nie skupiały się one na aspekcie porównania technologii, a na sposobie wykonywania pomiarów, a także definiowaniu kryteriów oceny jakości. W pracy tej, interfejs programowania aplikacji zbudowany w oparciu o metodologię REST poddawany był zmiennym obciążeniom (tj. testy linii bazowej, testy obciążeniowe oraz testy przeciążeniowe). W czasie dokonywania ewaluacji monitorowano średni czas odpowiedzi serwera, zgodność kodów statusu zawartych w ramach uzyskiwanych odpowiedzi, informacje o zużyciu zasobów sprzętowych serwera, czy też wartość wskaźnika satysfakcji klienta. Rezultaty przeprowadzonych badań wykazały kluczowe znaczenie optymalizacji kodu źródłowego aplikacji, w kontekście realizacji rozbudowanych i skalowalnych usług sieciowych.

Rozdział 3

Opis problemu badawczego

W ramach niniejszego rozdziału omówiony został podjmowany problem badawczy. W związku z jego złożonością, autor pracy zdecydował się na podział tego zagadnienia na określone aspekty, będące różnorodnymi względem funkcjonalności internetowych interfejsów programowania aplikacji. Na koniec tej części pracy, na podstawie sformułowanych konktekstów badawczych, zdefiniowano listę scenariuszy realizacji badań.

3.1. Przedstawienie aspektów problemu badawczego

Podejmowany w ramach niniejszej pracy problem badawczy, tyczy się wydajności usług sieciowych, których charakterystyka wyróżnia zastosowanie wielu odrębnych komponentów programistycznych. Komponentami tymi, mogą być zarówno: biblioteki obsługujące proces mapowania obiektowo-relacyjnego, zastosowany wzorzec projektowy w kontekście wewnętrznej architektury API, rodzaj wykorzystywanego zewnętrznego źródła danych, czy też wybrana chmurowa platforma wdrożeniowa. W związku z mnogością zagadnień występujących w ramach niniejszego problemu badawczego, zdecydowano się na dokonanie podziału jego opisu w taki sposób, aby na podstawie każdego z aspektów, możliwe było zdefiniowanie wyspecyfikowanych scenariuszy realizacji badań.

Każdy z aspektów problemu badawczego wymieniony poniżej, rozpatrywany będzie względem dwóch odrębnych zestawów technologii programistycznych (tj. C#/.NET oraz JavaScript/NodeJS). Scenariusze badawcze, opracowane na podstawie, każdego z opisanych w tym podrozdziale aspektów, uwzględniać będą porównanie uzyskanych wyników badań dla obu wymienionych rozwiązań informatycznych.

Wydajność interfejsu API względem liczby żądań generowanych przez klientów

Pierwszym z omawianych aspektów rozważanego problemu badawczego jest wpływ wydajności działania interfejsu programowania aplikacji, względem liczby klientów, którzy w sposób równoległy generują żądania w kierunku API.

Wydajność, w kontekście tego właśnie aspektu, interpretowana jest poprzez metryki czasu odpowiedzi na żądanie, a także procentowe wartości wykorzystania zasobów sprzętowych interfejsu programowania aplikacji, takich jak centralna jednostka przetwarzania, czy też pamięć operacyjna o dostępie swobodnym.

Zgodnie z obowiązującymi praktykami realizacji pomiarów wydajności usług sieciowych, charakterystyka ta, powinna być wyliczana w oparciu o technikę testowania rozproszonego *(ang. Distributed Testing)*. Przedstawiana technika, zakłada wykorzystanie wielu odrębnych syste-

mów informatycznych, wykonujących równolegle ewaluację obciążeniową. Wartości metryk, uzyskane dla każdej z maszyn przeprowadzających testy, powinny zostać zgrupowane, a także analizowane jako pochodzące z jednego źródła.

Zmiana wydajności, obserwowana powinna być wraz ze stałym zwiększeniem natężenia liczby klientów, a dla każdego z kolejnych przedziałów liczbowych, uwzględniających kolejne przyrosty wysyłanych pakietów, metryki wydajnościowe powinny być porównywane względem ustalonego wskaźnika referencyjnego. Kalkulacja tego wskaźnika natomiast, wykonywana jest w oparciu o ewaluację wydajności dla standardowych warunków pracy interfejsu programowania aplikacji. Przykładem takich warunków, może być realizacja testu obciążeniowego dla pojedynczej maszyny testującej.

Jednym z podstawowych, otwartych standardów, które posłużyć mogą do budowy wskaźnika referencyjnego jest APDEX *(ang. Application Performance Index)*. Indeks ten, pozwala na zdefiniowanie trzech przedziałów liczbowych, określających odczucia klienta testującego oprogramowanie. Przedziały te, przedstawiane są jako progi satysfakcji, tolerancji oraz frustracji. Po wykonaniu testów odbywających się w standardowych warunkach pracy usługi sieciowej, ustalenie wartości metryk wydajnościowych dla każdego z trzech progów jest możliwe, a co za tym idzie, możliwe jest także porównanie wyników uzyskiwanych przy dowolnej liczbie klientów API, względem ustalonych progów.

Na podstawie weryfikacji wydajności interfejsu API względem liczby generowanych równolegle żądań, należy także ustalić graniczną wartość sumy maszyn klienckich, dla których interfejs programowania aplikacji jest w stanie obsługiwać zapytania, w czasie zawierającym się w każdym z przedziałów referencyjnych.

Korelacja charakterystyk wydajnościowych względem określonego zewnętrznego źródła danych

Kolejny z kontekstów problemu badawczego dotyczy wpływu zastosowania odmiennych zewnętrznych źródeł danych, na efektywność pracy usługi sieciowej jaką jest interfejs API.

Koncepcja wydajności w przypadku tego aspektu problemu, rozumiana jest w sposób analogiczny do pojęcia, wprowadzonego w ramach poprzedniej sekcji.

W związku z istotnością uwzgladnienia zewnętrznych źródeł danych, jako elementów z którymi nieustannie komunikują się nowoczesne usługi sieciowe, w ramach niniejszego aspektu omawianego problemu badawczego, weryfikowany jest wpływ sposobu obsługi najpopularniejszych spośród dostępnych nieodpłatnie systemów bazodanowych, na efektywność operacji realizowanych przez poszczególne interfejsy programowania aplikacji. Problem badawczy, uwzględnia zastosowanie zarówno czterech relacyjnych systemów baz danych, jak i jednego nierelacyjnego.

Sposób obserwacji zmiany wydajności usługi sieciowej, również jest analogiczny, do tego, który został przedstawiony w ramach poprzedniego aspektu, jednakże należy zwrócić uwagę, na konieczność ponownego zdefiniowania przedziałów satysfakcji, tolerancji oraz frustracji dla wskaźnika referencyjnego. Referencja do wartości tego wskaźnika, obliczonego bez uwzględnienia połączenia z systemem bazodanowym, prowadziła by do zaniżenia wartości ogólnej wydajności testowanego oprogramowania.

Kluczowym czynnikiem omawianego aspektu problemu badawczego jest zapewnienie deterministycznego charakteru stanu łącza sieciowego, występującego pomiędzy serwerem bazodanowym a usługą interfejsu programowania aplikacji. Dlatego też, w przypadku testów realizowanych w środowisku lokalnym, oba komponenty programowo-sprzętowe powinny znajdować się w tej samej lokalizacji. Należy podkreślić jednak, że ważnym jest, aby obie usługi informatyczne nie były uruchomione w ramach tego samego środowiska sprzętowego. Dzięki temu, pozyskane wartości metryk wykorzystania zasobów sprzętowych nie będą obarczone nie-

dokładnością. W kontekście realizacji badań w środowisku chmurowym natomiast, ważnym jest wdrożenie zarówno API, jak i serwera bazy danych, w ramach tego samego centrum obliczeniowego, a także rozdzielenie obu usług, pomiędzy odmienne fizyczne urządzenia. Ponadto, modyfikacji powinien ulec jeden z elementów kryterium wydajności, który definiowany jest jako czas odpowiedzi interfejsu na żądanie klienta. W związku z dyspersją geograficzną obu stron komunikacji, niemożliwym jest zachowanie przewidywalnego charakteru łącza sieciowego wykorzystywanego do transmisji danych. Twierdzenie to, implikuje konieczność realizacji pomiaru czasu działania usługi sieciowej w sposób odmienny. Kryterium czasu odpowiedzi na żądanie, rozumiane w tym przypadku jest jako przedział czasowy od momentu pozyskania żądania, do momentu zakończenia wszystkich operacji, realizowanych w kontekście tego żądania.

Efektywność realizacji złożonych obliczeń oraz wsparcia dla programowania współbieżnego i metod asynchronicznych

Niniejszy aspekt problemu badawczego dotyczy wpływu wykorzystania, dostępnych w ramach określonego języka mechanizmów programowania współbieżnego, a także sposobu realizacji operacji asynchronicznych, na efektywność przeprowadzania kalkulacji w obrębie warstwy logiki biznesowej interfejsu programowania aplikacji.

Pojęcie efektywności dokonywanych kalkulacji postrzegane jest poprzez liczbę wykonanych iteracji głównej pętli zaimplementowanego algorytmu metaheurystycznego, rozwiązującego określony problem z rodziny NP-trudnych.

W celu zachowania rzetelności badań, omawiany fragment problemu badawczego uwzględnia zastosowanie analogicznego algorytmu, realizującego operacje w ten sam sposób, a także rozwiązującego ten sam problem obliczeniowy. W tym przypadku, zaobserwować będzie można fakt przystosowania technologii poddawanej ewaluacji, do dokonywania procesu zrównoleglania obliczeń, a także przeprowadzania wewnętrznej optymalizacji określonych linii zdefiniowanego kodu źródłowego.

Zaimplementowany algorytm metaheurystyczny, dostępny będzie bezpośrednio z poziomu punktów końcowych badanych interfejsów programowania aplikacji, a liczba iteracji głównej pętli algorytmu, mierzona będzie dla ustalonego, stałego czasu wykonania programu.

Ponadto, omawiany aspekt badawczy dotyczy także weryfikacji wydajności w kontekście zastosowania metod asynchronicznych. W związku ze znacząco odmienną strukturą badanych środowisk uruchomieniowych oraz języków programowania, mechanizmy obsługi operacji asynchronicznych zaimplementowane są w tych technologiach, na różnych poziomach obsługi programu. W jedym przypadku, obsługa operacji tych, jest wykonywana bezpośrednio w ramach języka programowania, natomiast w kontekście drugiej z technologii, metody których wynik nie jest dostarczany natychmiastowo, muszą zostać obsłużone wewnątrz środowiska uruchomieniowego.

Badanie weryfikacji wydajności dla funkcji asynchronicznych, oparte jest o odwołanie się interfejsu API, do współpracującej z nim hipertekstowej usługi sieciowej, pełniącej rolę pośrednika w dostępie do zdefiniowanych wewnątrz niej informacji. Przedstawiona usługa sieciowa, zostanie zaimplementowana jako odrębne oprogramowanie i będzie niezależna od obu porównywanych technologii.

W kontekście drugiej z części aspektu problemu badawczego, metryką wydajności będzie czas odpowiedzi interfejsu na żądanie.

Wpływ zastosowania wzorca projektowego podziału odpowiedzialności na efektywność realizacji operacji bazodanowych

Rozważany aspekt problemu badawczego tyczy się wpływu implementacji optymalizacji wydajnościowych w kontekście komunikacji interfejsu programowania aplikacji z zewnętrznym źródłem danych.

Wykorzystując konwencjonalną trójwarstwową architekturę interfejsu API, stosowany zostaje ten sam model danych, zarówno do operacji odczytu jak i zapisu. Powoduje to brak możliwości dostosowania modelu, względem specyfiki konkretnego rodzaju operacji. Wprowadzenie wzorca projektowego separacji zapytań oraz komend ma na celu umożliwienie dokonania optymalizacji wydajnościowych wyizolowanych fragmentów modelu danych, a także wykorzystywanie ich tylko i wyłącznie w kontekście jednego typu operacji.

Co więcej, optymalizacja może być wykonana nie tylko na poziomie modelu danych, ale także w ramach fizyczynych struktur zawartych wewnątrz obsługiwanego systemu bazodanowego. Dlatego też, przedstawiany aspekt problemu badawczego dotyczy zastosowania zarówno odrębnych modeli danych wewnątrz API, odrębnych struktur programistycznych obsługujących dane modelu, jak i odseparowanych zewnętrznych źródeł danych.

Oba zastosowane źródła danych, powinny cechować się taką samą strukturą, jednakże każde z nich powinno wprowadzać charakterystyczne dla typu wykonywanych operacji, usprawnienia wydajnościowe. Ponadto, aby zachować spójność zawartości dostępnej dla klienta w ramach API, po odwołaniu się do systemu bazodanowego w celu zapisania rekordu, musi on zostać następnie zreplikowany do źródła danych obsługującego operację odczytu.

Wydajność, rozumiana poprzez czas odpowiedzi interfejsu API na żądanie klienta, powinna zostać porównana z tą, wykazywaną przez usługę sieciową opierającą się na architekturze 3-warstwowej i wykorzystującą pojedyncze źródło danych.

Wpływ zastosowania mechanizmów pamięci podręcznej na wydajność interfejsów API

Problem badawczy w kontekście wykorzystania mechanizmów pamięci podręcznej, dotyczy porównania efektywności działania interfejsów programowania aplikacji implementujących standardowy oraz autorski mechanizm przechowywania rezultatów wykonanych uprzednio żądań.

Standardowy mechanizm przechowywania żądań w ramach pamięci podręcznej uwzględniać powinien stały czas ważności pojedynczego wpisu, a także jego unieważnienie w przypadku wykonania operacji modyfikującej dane. W takim przypadku, czas odpowiedzi na żądanie powinien być zwiększony w momencie konieczności odwołania się API do zewnętrznego źródła danych, a następnie zredukowany w przedziale czasowym, w ramach którego wpis pamięci podręcznej jest aktywny.

Zaimplementowany autorski mechanizm pamięci podręcznej wyróżniać będzie się zmiennym czasem ważności poszczególnych wpisów, który zależny będzie od prawdopodobieństwa wywołania określonego punktu końcowego, na podstawie informacji o liczbie historycznych wywołań. Czym większe istnieje prawdopodobieństwo ponownego wywołania punktu końcowego, tym czas ważności rezultatu przechowywanego w pamięci podręcznej będzie większy. Analogicznie do standardowego mechanizmu pamięci podręcznej, operacja modyfikacji danych unieważnia wszystkie spośród wpisów, które odwołują się do przekształconych informacji.

Celem niniejszego aspektu badawczego w ramach rozważanego problemu jest porównanie zmiany wydajności działania API, postrzeganej jako średni czas odpowiedzi na żądanie w ustalonym, stałym przedziale czasowym. Porównywane zostaną mechanizmy standardowy oraz autorski, odrębnie dla każdej z technologii programistycznych.

Wpływ wdrożenia interfejsu API na dedykowanej platformie chmurowej na jego efektywność działania

Ostatni z przedstawianych aspektów problemu badawczego odnosi się do wpływu wydajności pracy interfejsu API, w zależności od rodzaju zatosowanej platformy chmurowej, na jakiej zostanie on wdrożony.

Interfejs programowania aplikacji jest usługą, której funkcjonowanie jest nieodłącznie powiązane z serwerem internetowym. Serwer sieci Web pełni rolę warstwy opakowującej, wewnątrz której działać może interfejs API. Wdrożenie rozważanej usługi sieciowej w ramach sieci Internet, wiąże się w związku z tym z uruchomieniem serwera sieci web w ramach komputera eksponowanego w sieci rozległej. Należy również zauważyć konieczność zastosowania hipertekstowego serwera pośredniczącego, po to, aby klient usługi, mógł się z nią komunikować z wykorzystaniem protokołu HTTP.

System informatyczny, składający się z przedstawionych powyżej komponentów może zostać uruchomiony w ramach wirtualnego serwera prywatnego, udostępnianego przez określonego dostawcę infrastruktury serwerowej. Model taki, definiowany jest jako infrastruktura w postaci usługi klienckiej *(ang. Infrastructure as a Service)*. Ponadto, przygotowane oprogramowanie może zostać wdrożone na dedykowanej określonej technologii, platformie chmurowej. W ramach platformy tej, użytkownik, za pomocą dostarczonego interfejsu komunikacji może wdrażać oraz konfigurować działanie swojego oprogramowania. Taki model dostarczania zasobu z kolei, nazywany jest platformą w postaci usługi klienckiej *(ang. Platform as a Service)*.

Niniejszy aspekt problemu badawczego, dotyczy porównania wydajności API, w zależności od jego wdrożenia na generycznym wirtualnym serwerze prywatnym, a także dedykowanej platformie chmurowej, dostosowanej pod kątem określonego środowiska uruchomieniowego oraz języka programowania.

Wskaźnik ewaluacji efektywności działania interfejsu programowania aplikacji, obejmuje te same metryki, które przedstawione zostały w pierwszym spośród omawianych aspektów problemu badawczego. Kryterium czasu odpowiedzi na żądanie, musi zostać jednakże uniezależnione od niedeterministycznego charakteru łącza internetowego, dlatego też, ten właśnie parametr, będzie dotyczył czasu od momentu otrzymania żądania przez API, do chwili wygenerowania odpowiedzi na żądanie.

3.2. Sformułowanie scenariuszy badawczych

Na podstawie przedstawionych w poprzednim podrozdziale aspektów problemu badawczego, sformułowane zostały konkretne scenariusze badawcze. W każdym ze scenariuszy, zdefiniowano zbiór czynności wykonywanych w ramach określonego badania, wyszczególniono kryteria porównawcze dla danej obserwacji, wymieniono dostosowywalne parametry badania, a także skonkretyzowano czynności, które muszą zostać podjęte, jako warunki konieczne przed wykonaniem badania. Każdy ze scenariuszy badawczych odzwierciedlony został w formie tabeli.

Rozdział 4

Redakcja pracy

4.1. Układ pracy

Standardowo praca powinna być zredagowana w następującym układzie:

Strona tytułowa
Strona z dedykacją (opcjonalna)
Spis treści
Spis rysunków (opcjonalny)
Spis tabel (opcjonalny)
Skróty (wykaz opcjonalny)
1. Wstęp
   1.1 Cel i zakres pracy
   1.2 Układ pracy
2. Kolejny rozdział
   2.1 Sekcja
      2.1.1 Podsekcja
         Nienumerowana podpodsekcja
            Paragraf
...
#. Podsumownie i wnioski
Literatura
A. Dodatek
   A.1 Sekcja w dodatku
...
$. Zawartość płyty CD/DVD
Indeks rzeczowy (opcjonalny)

Spis treści – powinien być generowany automatycznie, z podaniem tytułów i numerów stron. Typ czcionki oraz wielkość liter spisu treści powinny być takie same jak w niniejszym wzorcu.

Spis rysunków, Spis tabel – powinny być generowane automatycznie (podobnie jak Spis treści). Elementy te są opcjonalne (robienie osobnego spisu, w którym na przykład są tylko dwie pozycje specjalnie nie ma sensu).

Wstęp – pierwszy rozdział, w którym powinien znaleźć się opis dziedziny, w jakiej osadzona jest praca, oraz wyjaśnienie motywacji do podjęcia tematu. W sekcji „Cel i zakres" powinien znaleźć się opis celu oraz zadań do wykonania, zaś w sekcji „Układ pracy" – opis zawartości kolejnych rozdziałów.

Podsumowanie – w rozdziale tym powinny być zamieszczone: podsumowanie uzyskanych efektów oraz wnioski końcowe wynikające z realizacji celu pracy dyplomowej.

Literatura – wykaz źródeł wykorzystanych w pracy (do każdego źródła musi istnieć odpowiednie cytowanie w tekście). Wykaz ten powinien być generowany automatycznie.

Dodatki – miejsce na zamieszczanie informacji dodatkowych, jak: Instrukcja wdrożeniowa, Instrukcja uruchomieniowa, Podręcznik użytkownika itp. Osobny dodatek powinien być przeznaczony na opis zawartości dołączonej płyty CD/DVD. Założono, że będzie to zawsze ostatni dodatek.

Indeks rzeczowy – miejsce na zamieszczenie kluczowych wyrazów, do których czytelnik będzie chciał sięgnąć. Indeks powinien być generowany automatycznie. Jego załączanie jest opcjonalne.

4.2. Styl

Zasady pisania pracy (przy okazji można tu zaobserwować efekt wyrównania wpisów występujących na liście wyliczeniowej uzależnione od długości etykiety):

1. Praca dyplomowa powinna być napisana w formie bezosobowej („w pracy pokazano ...”). Taki styl przyjęto na uczelniach w naszym kraju, choć w krajach anglosaskich preferuje się redagowanie treści w pierwszej osobie.
2. W tekście pracy można odwołać się do myśli autora, ale nie w pierwszej osobie, tylko poprzez wyrażenia typu: „autor wykazał, że ...”.
3. Odwołując się do rysunków i tabel należy używać zwrotów typu: „na rysunku pokazano ...”, „w tabeli zamieszczono ...” (tabela i rysunek to twory nieżywotne, więc „rysunek pokazuje” jest niepoprawnym zwrotem).
4. Praca powinna być napisana językiem formalnym, bez wyrażeń żargonowych („sejwowanie” i „downloadowanie”), nieformalnych czy zbyt ozdobnych („najznamienitszym przykładem tego niebywałego postępu ...”)
5. Pisząc pracę należy dbać o poprawność stylistyczną wypowiedzi
   - trzeba pamiętać, do czego stosuje się „liczba”, a do czego „ilość”,
   - nie „szereg funkcji” tylko „wiele funkcji”,
   - redagowane zdania nie powinny być zbyt długie (lepiej podzielić zdanie wielokrotnie złożone na pojedyncze zdania),
   - itp.
6. Zawartość rozdziałów powinna być dobrze wyważona. Nie wolno więc generować sekcji i podsekcji, które mają zbyt mało tekstu lub znacząco różnią się objętością. Zbyt krótkie podrozdziały można zaobserwować w przykładowym rozdziale 6.
7. Niedopuszczalne jest pozostawienie w pracy błędów ortograficznych czy tzw. literówek – można je przecież znaleźć i skorygować automatycznie.

10005. Niedopuszczalne jest pozostawienie w pracy błędów ortograficznych czy tzw. literówek – można je przecież znaleźć i skorygować automatycznie.

Rozdział 5

Uwagi techniczne

5.1. Rysunki

W niniejszym szablonie numeracja rysunkw odbywa si automatycznie wedug nastpujcych regu: rysunki powinny mie numeracj cig w obrbie danego rozdziau, sam za numer powinien skada si z dwch liczb rozdzielonych kropk. Pierwsza liczb ma by numer rozdziau, drug – kolejny numer rysunku w rozdziale. Przykadowo: pierwszy rysunek w rozdziale 1 powinien mie numer 1.1, drugi – numer 1.2 itd., pierwszy rysunek w rozdziale 2 powinien mie numer 2.1, drugi – numer 1.2 itd.

Rysunki powinny by wyrodkowane na stronie wraz z podpisem umieszczonym na dole. Podpisy nie powinny koczy si kropk. Czcionka podpisu powinna by mniejsza od czcionki tekstu wiodcego o 1 lub 2 pkt (w szablonie jest to czcionka rozmiaru `small`). Ponadto naley zachowywa odpowiedni odstp midzy rysunkiem, podpisem rysunku a tekstem rozdziau. W przypadku korzystania z szablon odstpy te regulowane s automatycznie. Podpis i grafika musz stanowi jeden obiekt. Chodzi o to, e w edytorach tekstu typu Office podpis nie scala si z grafik i czasem trafia na nastpn stron, osieracajc grafik. Korzystajcym z niniejszego szablonu i otoczenia `\figure` takie osierocenie nigdy si nie zdarzy.

Do kadego rysunku musi istnie odwoanie w tekcie (inaczej mwic: niedopuszczalne jest wstawienie do pracy rysunku bez opisu). Odwoania do rysunkw powinny mie posta: „Na rysunku 3.3 przedstawiono...” lub „... co ujto na odpowiednim schemacie (rys. 1.7)”. Jeli odwoanie stanowi cz zdania, to wtedy wyraz „rysunek” powinien pojawi si w caoci. Jeli za odwoanie jest ujte w nawias (jak w przykadzie), wtedy naley zastosowa skrt „rys.”. Jeli do stworzenia obrazka wykorzystano jakie rda, to powinny one by zacytowane w podpisie tego rysunku.

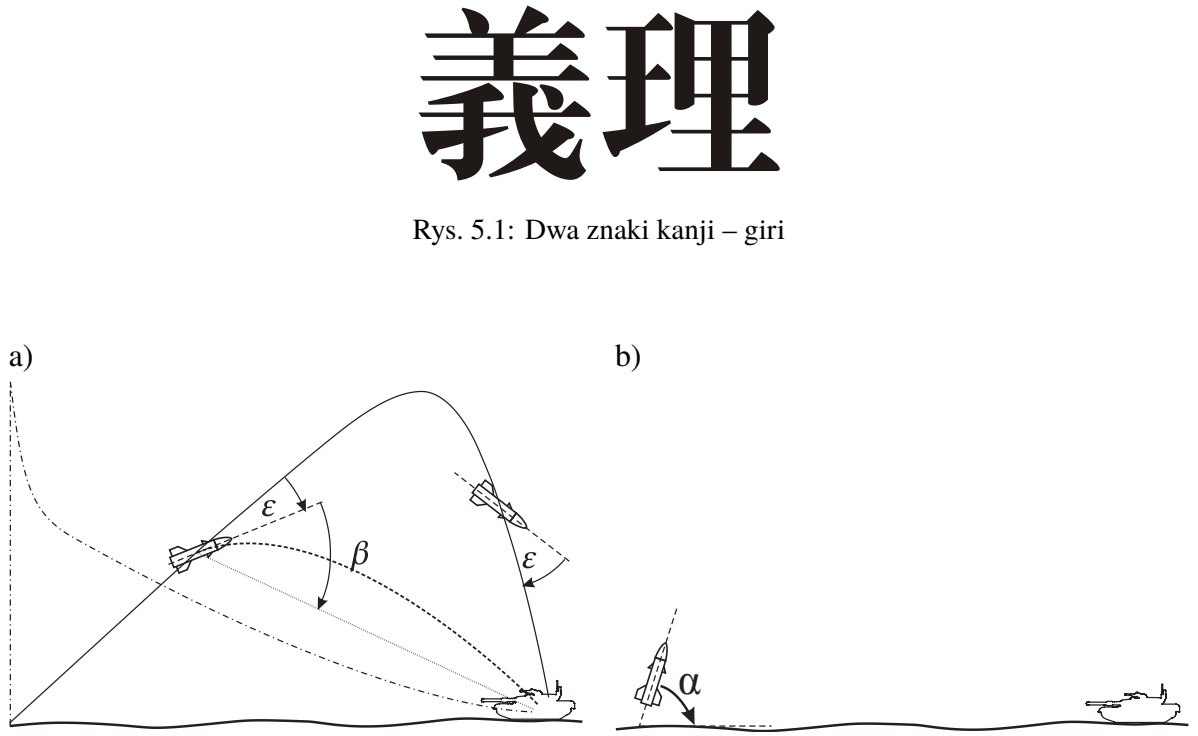
Naley pamita o tym, e „rysunki” to twory nieywotne. W zwizku z tym nie mog ”pokazywa”. Dlatego „rysunek 1.1 pokazuje ...” jest stylistycznie niepoprawne. Zamiast tego zwrotu trzeba uy „ na rysunku 1.1 pokazano ...”.

Rysunki mona wstawia do pracy uywajc polecenia `\includegraphics`. Zalecane jest, aby pliki z grafikami byy umieszczane w katalogach odpowiadajcych numerom rozdziaw czy literom dodatkw: `rys01`, `rysA` itd. Sposb wstawiania rysunkw do pracy zademonstrowano na przykadze rysunkw 5.1 i 5.2.

Listing 5.1: Kod rdowy przykadw wstawiania rysunkw do pracy

```
\begin{figure}[ht]
 \centering
  \includegraphics[width=0.3\linewidth]{rys05/kanji-giri}
 \caption{Dwa znaki kanji - giri}
 \label{fig:kanji-giri}
\end{figure}
```

```
\begin{figure}[htb]
 \centering
  \begin{tabular}{@{}ll@{}}
  a) & b) \\
  \includegraphics[width=0.475\textwidth]{rys05/alfa1} &
  \includegraphics[width=0.475\textwidth]{rys05/beta1}
        % jeli obraki s rnej wysokoci, mona je wyrwna do gry stosujc vtop
            ↪ jak niej
        % \vtop{\vskip-2ex\hbox{{\includegraphics[width=0.475\textwidth]{
            ↪ rys05/beta1}}}} &
        % \vtop{\vskip-2ex\hbox{{\includegraphics[width=0.475\textwidth]{
            ↪ rys05/alfa1}}}}
  \end{tabular}
 \caption{Wyznaczanie trajektorii lotu rakiety:
 a) trzy podejcia, b) podejcie praktyczne}
 \label{fig:alfabeta}
\end{figure}
```



Rys. 5.1: Dwa znaki kanji – giri

Rys. 5.2: Wyznaczanie trajektorii lotu rakiety: a) trzy podejcia, b) podejcie praktyczne

Grafiki wektorowe powinny by dostarczone w plikach o formacie pdf. Rozmiar strony w pliku pdf powinien by troszeczk wikszy ni zamieszczona na nim grafika (prosz spojrze na przykady grafik wykorzystanych w niniejszym szablonie). Chodzi o to, aby na rysunku nie pojawiaa si niepotrzebna biaa przestrze. Grafiki rastrowe (gwnie zrzuty z ekranu bd zdjcia) powinny by dostarczane w plikach o formacie png z kompresj bezstratn. Zastosowanie kompresji stratnej, jak jpg, wprowadza niepotrzebne artefakty. Podobnie jak w przypadku grafik wektorowych, grafiki rastrowe nie powinny mie biaych marginesw.

Na rysunkach nie powinno stosowa si 100% czarnego wypenienia, bo robi si plamy przebijajce si przez kartk. Zamiast tego wypenienie powinno by ok. 90% czerni.

Czcionka na rysunkach nie moe by wiksza od czcionki wiodcej tekstu (jedyny wyjtek to np. jakie nagwki). Naley stosowa czcionk kroju Arial, Helvetica bd tego samego kroju co czcionka dokumentu (`texgyre-termes`).

Jeli na jednym rysunku pojawi si ma kilka grafik, to zamiast stosowa `subfigure` lub inne otoczenia naley wstawi grafiki w tabel, opisa j indeksami a) i b), a potem odnie si do tego w podpisie (rys. 5.2). Czasem pomaga w pozycjonowaniu rysunkw uycie komendy: `\vtop{\vskip3ex\hbox{\includegraphics[width=0.475\textwidth]{nazwa}}}`

Na rysunkach nie wolno naduywa kolorw oraz ozdobnikw (wiele narzdzi do tworzenia diagramw dostarcza grafik z cieniowaniem, gradacj kolorw itp. co niekoniecznie przekada si na czytelno rysunku).

Podczas rozbienia zrzutw z ekranu naley zadba o to, by taki zrzut by czytelny po wydrukowaniu. Czyli aby pojawiajce si literki byy wystarczajco due, a przestrzenie bez treci – relatywnie mae. Przystpujc do robienia zrzutu trzeba odpowiednio wyskalowa elementy na ekranie. Na przykad robic zrzut z przegldarki FF najpierw naley wcisn CTR–0 (domylne skalowanie), potem CTR–- (zmniejszenie skali o stopie). Potem dobrze jest zawzi okno przegldarki tak, by interesujca tre wypenia je w caoci. Jeli na obserwowanej stronie jest zbyt duo pustych obszarw, to naley je jako zawzi (sterujc wielkoci okna przegldarki lub aktywnymi elementami interfejsu uytkownika). Zrzut bowiem wcale nie musi by odzwierciedleniem 1:1 domylnego ukadu obserwowanych elementw. Wane jest, by na zrzucie z ekranu pokaza interesujcy, opisywany fragment i eby ten fragment by czytelny.

Czasem problemem jest tworzenie zrzutw z ekranu, gdy wystpuj na nim dane wraliwe. Istniej dwa sposoby na radzenie sobie z tym problemem. Pierwszy polega na zastpieniu w systemie danych danych rzeczywistych danymi testowymi – wygenerowanymi tylko do celw prezentacji. Zrzut robi si wtedy na bazie danych testowych. Drugi polega na wykonaniu zrzutu z ekranu, na ktrym pokazano dane rzeczywiste, i nastpnie zamianie tych danych ju w pliku graficznym za pomoc odpowiedniego edytora (np. `gimp`). Czyli oryginalny zrzut z ekranu naley otworzy w edytorze, a potem nadpisa oryginalny tekst wasnym tekstem. Konieczne jest wtedy dobranie odpowiednich czcionek aby nie byo wida wprowadzonych zmian.

> Uwaga: takie manipulowanie zrzutami jest usprawiedliwione jedynie w przypadku koniecznoci ochrony danych wraliwych czy te lepszego pokazania wybranych elementw. Nie moe to prowadzi generowania faszywych rezultatw!!!

5.2. Wstawianie kodu rdowego

Kod rdowy mona wstawia jako blok tekstu pisany czcionk maszynow. Uywa si do tego otoczenie `\lstlisting`. W atrybutach otoczenia mona zdefiniowa tekst podpisu wstawianego wraz z numerem nad blokiem, etykiet do tworzenia odwoa, sposb formatowania i inne ustawienia. Zaleca si stosowanie w tym otoczeniu nastpujcych parametrw:

```
\begin{lstlisting}[label=list:req1,caption=Initial HTTP Request,
                   basicstyle=\footnotesize\ttfamily]
```

Szczeglnie przydatne podczas wstawiania wikszej iloci kodu rdowego jest zastosowanie parametru `basicstyle=\footnotesize\ttfamily`. Dziki niemu zmniejsza si czcionka, a przez to na stronie mona zmieci dusze linijki kodu. Uycie tak zdefiniowanego parametru nie jest jednak sztywnym zaleceniem. Wielko czcionki mona dobiera do potrzeb.

Listing 5.2: Initial HTTP Request

```
GET /script/Articles/Latest.aspx HTTP/1.1
Host: www.codeproject.com
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml
User-Agent: Mozilla/5.0 ...
Accept-Encoding: gzip,deflate,sdch
```

```
Accept-Language: en-US...
Accept-Charset: windows-1251,utf-8...
```

Mona te sformatowa kod bez stosowania numerowanego podpisu (wtedy nie zamieszcza si caption na licie atrybutw).

```
GET /script/Articles/Latest.aspx HTTP/1.1
Host: www.codeproject.com
Connection: keep-alive
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml
User-Agent: Mozilla/5.0 ...
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US...
Accept-Charset: windows-1251,utf-8...
```

Istnieje moliwo wstawiania kodu rdowego w biecej linijce tekstu. Mona to zrobi na kilka sposobw:

- korzystajc z polecenia \texttt ustawiajcego czcionk maszynow, jak w przykadzie tutaj (efekt zastosowania komendy \texttt{tutaj}). Problemem jednak mog okaza si znaki podkrelenia i inne znaki kontrolne.
- korzystaj z otoczenia \verb zapewniajcego wypisanie kodu czcionk maszynow jak w przykadzie tutaj (efekt zastosowania komendy \verb|tutaj|). Problemem jest to, e polecenie \verb nie potrafi ama duszego tekstu.
- korzystajc z polecenia \lstin umoliwiajcego wypisanie kodu czcionk ustawian w opcjach jak w przykadzie tutaj (efekt komendy \lstset{basicstyle=\ttfamily}\lstinline{tutaj}) lub tutaj (efekt komendy \lstinline[basicstyle=\ttfamily]=tutaj=).

5.3. Wykaz literatury oraz cytowania

Cytowania powinny by zamieszczane w tekcie z uyciem komendy \cite{}. Jej argumentem powinien by klucz cytowanej pozycji (lub lista kluczy rozdzielonych przecinkiem bez spacji, jeli takich pozycji w danym miejscu cytuje si wicej) jaki jest uywany w bazie danych bibliograficznych (plik dokumentacja.bib). Po kompilacji bibtex i pdflatex w tekcie pojawia si waciwy odsyacz do pozycji w wykazie literatury (ujty w kwadratowe nawiasy – zgodnie z tym, co definiuje styl plabbrv.bst), za w samym wykazie (rozdzia Literatura) – zacytowana pozycja. Przykadem cytowania jest: „dobrze to opisano w pracach [**?**, **?**]" (gdzie zastosowano komend \cite{JS07,SQL2}).

Co do zawartoci rekordw bibliograficznych - style bibtexowe potrafi „skraca" imiona (czyli wstawia, jeli taka wola, inicjay zamiast penych imion). Niemniej dobrze jest od razu przyj jak konwencj. Proponuje si, aby w rekordach od razu wstawiane byy inicjay zamiast penych imion.

Niekiedy tytuy prac zawieraj wyrazy z duymi i maymi literami. Takie tytuy naley bra w podwjne nawiasy klamrowe, aby bibtex nie zamieni ich na posta, w ktrej poza pierwsz liter pozostae s mae.

Jeli jaki cytowany zasb pochodzi z Internetu, to jego rekord w pliku bib powinien wyglda jak niej.

```
@INPROCEEDINGS{SQL2,
  title={{A MySQL-based data archiver: preliminary results}},
  author={Bickley, M. and Slominski, Ch.},
  booktitle = {{Proceedings of ICALEPCS07}},
        month = oct,
        day = {15--19},
```

```
        year={2007},
  note={\url{http://www.osti.gov/scitech/servlets/purl/922267}
        [dostp dnia 20 czerwca 2015]}
}
```

A to inny przykad rekordu danych bibliograficznych:

```
@TechReport{JS07,
        author = {Jdrzejczyk, J. and rdka, B.},
        title  ={Segmentacja obrazw metod drzew decyzyjnych},
        year = {2007},
        institution = {Politechnika Wrocawska, Wydzia Elektroniki}
}
```

5.4. Indeks rzeczowy

Generowanie indeksu po trosze wyglda jak generowanie wykazu literatury – wymaga kilku krokw. Podczas pierwszej kompilacji `pdflatex` generowany jest plik z rozszerzeniem `*.idx` (zawierajcy „surowy indeks"). Nastpnie, bazujc na tym pliku, generowany jest plik z rozszerzeniem `*.ind` zawierajcy sformatowane dane. Ten krok wymaga uruchomienia odpowiedniego narzdzia oraz zastosowania plik z definicj stylu `Dyplom.ist`. W kroku ostatnim dokonuje si kolejnej kompilacji `pdflatex` (dziki niej w wynikowym dokumencie pojawi si Indeks rzeczowy). Domylnie Indeks rzeczowy zostanie sformatowany w ukadzie dwukolumnowym.

Oczywicie aby to wszystko zadziaao w kodzie szablonu naley umieci odpowiednie komendy definiujce elementy indeksu rzeczowego (`\index`) oraz wstawiajce sformatowany Indeks rzeczowy do dokumentu wynikowego (`\printindex`). Wicej informacji o tworzeniu indeksu rzeczowego mona znale na stronie `https://en.wikibooks.org/wiki/LaTeX/Indexing`. Poniej przedstawiono przykady komend uytych w szablonie do zdefiniowania elementw indeksu rzeczowego:

- `\index{linia komend}` – pozycji gwna.
- `\index{generowanie!-- indeksu}` – podpozycja.

Generowanie pliku `*.ind` mona inicjowa na kilka sposobw:

- poprzez wydanie odpowiedniego polecenia bezporednio w linii komend
  ```
  makeindex Dyplom.idx -t Dyplom.ilg -o Dyplom.ind -s Dyplom.ist
  ```
- poprzez odpalenie odpowiedniego narzdzia rodowiska. Na przykad w `TeXnicCenter` definiuje si tzw. `output profiles`:
  ```
  makeindex "%tm.idx" -t "%tm.ilg" -o "%tm.ind" -s "%tm.ist"
  ```
  a samo generowanie pliku `*.ind` zapewni wybranie pozycji menu `Build/Makeindex`.
- korzystajc z odpowiednio sparametryzowanych pakietw i komend wewntrz kompilowanego dokumentu (czyli od razu przy okazji jego kompilacji).
  ```
  \DisemulatePackage{imakeidx}
  \usepackage[noautomatic]{imakeidx}
  % jeli chcemy, by indeks by generowany automatycznie programem makeindex:
  %\usepackage[makeindex]{imakeidx}
  % a tak pono mona przekaza opcje do programu generujcego indeks:
  %\makeindex[options=-s podrecznik -L polish -M lang/polish/utf8]
  %\makeindex[options=-s podrecznik]
  \makeindex
  ```
  Niestety, `makeindex` jest narzdziem, ktre umieszcza cz pozycji w grupie `Symbols`, a nie w grupach zwizanych z literkami alfabetu. W zwizku z czym indeksowany element zaczynajcy si od polskiej literki trafia do grupy `Symbols`, jak np. `\index{wiato}`. Jeli chce

si zamieszcza w indeksie symbole matematyczne, to dobrze jest to robi jak w nastpujcym przykadzie: `\index{$asterisk@$\ast$}` czy te `\index{c@$\mathcal{C}$}`, tj. dostarczajc przy okazji klucz do sortowania. Lepiej w tym wzgldzie radz sobie inne narzdzia, jak `texindy` lub `xindy` dostpne pod linuxem. Korzystajc z nich uzyskuje si grupy polskich literek w indeksie rzeczowym (hasa zaczynajce si od polskich literek ju nie trafiaj do grupy Symbols). Przykad polecenia wydanego z linii komend, w ktrym wykorzystano `texindy` zamieszczono poniej (zakadamy kodowanie plikw w UTF8, mona dla niniejszego szablonu zmieni na cp1250):

```
texindy -L polish -M lang/polish/utf8 Dyplom.idx
```

To polecenie wygeneruje `Dyplom.ind` o zawartoci:

```
\begin{theindex}
  \providecommand*\lettergroupDefault[1]{}
  \providecommand*\lettergroup[1]{%
      \par\textbf{#1}\par
      \nopagebreak
  }

  \lettergroup{G}
  \item generowanie
    \subitem -- indeksu, 27
    \subitem -- wykazu literatury, 27

  \indexspace

  \lettergroup{L}
  \item linia komend, 27

  \indexspace

  \lettergroup{}
  \item \'Swiat\IeC {\l }o, 28

\end{theindex}
```

Aby mie wiksz kontrol automatyczne generowanie indeksu zostao w niniejszym szablonie wyczone (indeks trzeba wygenerowa samemu, wydajc polecenie `makeindex` lub zalecane `texindy`).

5.5. Inne uwagi

Dobrym sposobem na kontrol bdw wystpujcych podczas kompilacji jest wstawianie linijki `\end{document}` w wybranym miejscu dokumentu. Jest to szczeglnie przydatne w przypadkach, gdy bdy te s trudne do zidentyfikowania (gdy wygenerowane przez kompilator numery linii z bdami nie s tymi, w ktrych bdy wystpuj). Wystarczy wtedy przestawi wspomnian linijk do kolejnych miejsc, a znajduj to miejsce, gdzie wystpuje problem.

Aby osign apostrofy maszynowe (czyli takie zoone z samych kresek) naley uy polecenia `"{}jak tutaj{}"` (podwjny apostrof i podwjny apostrof z na wszelki wypadek umieszczonymi nawiasami klamrowymi, nawiasy s potrzebne z tej racji, i podwjny apostrof przed niektrymi literkami zamienia je na literki z akcentami). W efekcie otrzymamy "jak tutaj". Jeli natomiast apostrofy maj by drukarskie (czyli zoone z kropek i kresek), to naley uy polecenia `,,jak tutaj''` (dwa pojedyncze przecinki i dwa pojedyncze apostrofy). W efekcie otrzymamy „jak tutaj”. Mona te uy znakw apostrofw odpowiednio zakodowanych jak tutaj, tylko e czasem trudno pisze si takie apostrofy w rodowiskach kompilacji projektw latexowych.

Oto sposoby ustawienia odstpw midzy liniami:

- uywajc komendy `\linespread{...}` (akceptowalne), przy czym atrybutem tej metody jest wspczynnik zaleny od wielkoci czcionki. Dla czcionki wiodcej 12pt odstp ptora linii osignie si komend `\linespread{1.241}`. Dla innych czcionek wiodcych wartoci tego parametru s jak w poniszym zestawieniu.

```
10pt 1.25 dla \onehalfspacing
     1.667 for \doublespacing,
                 poniewa ,,basic ratio'' = 1.2
                (\normalfont posiada \baselineskip rozmiaru 12pt)
11pt 1.213 dla \onehalfspacing oraz 1.618 dla \doublespacing,
     poniewa ,,basic ratio'' = 1.236
                (\normalfont posiada \baselineskip rozmiaru 13.6pt)
12pt 1.241 dla \onehalfspacing oraz 1.655 dla \doublespacing,
     poniewasince ''basic ratio'' is 1.208
                (\normalfont has a \baselineskip of 14.5pt)
```

  Kopot w tym, e raz ustawiony odstp bdzie obowizywa do wszystkich czcionek (nie dziaa tu adem mechanizm zmiany wspczynnika w zalenoci od wielkoci czcionki akapitu).
- uywajc pakietu `setspace` (niezalecane). Poniewa klasa `memoir` emuluje pakiet `setspace`, w preambule dokumentu naleaoby umieci:

```
\DisemulatePackage{setspace}
\usepackage{setspace}
```

  a potem mona ju sterowa odstp komendami:

```
\singlespacing
\onehalfspacing
\doubelspacing
```

  Ten sposb pozwala na korzystanie z mechanizmu automatycznej zmiany odlegoci linii w zalenoci od wielkoci czcionki danego akapitu.
- korzystajc bezporednio z komend dostarczonych w klasie `memoir` (zalecane):

```
\SingleSpacing
\OnehalfSpacing
\DoubleSpacing
```

  Ten sposb rwnie pozwala na korzystanie z mechanizmu automatycznej zmiany odlegoci linii w zalenoci od wielkoci czcionki danego akapitu.

Na koniec jeszcze uwaga o rozmiarze pliku wynikowego. Ot `pdflatex` generuje pliki `pdf`, ktre zazwyczaj mogyby by nieco lepiej skompresowane. Do lepszego skompresowania tych plikw mona uy programu `ghostscript`. Wystarczy w tym celu wyda komend (pod windowsami):

```
gswin64 -sDEVICE=pdfwrite -dCompatibilityLevel=1.4 -dNOPAUSE -dQUIET \
-dSAFER -dBATCH -sOutputFile=Dyplom-compressed.pdf Dyplom.pdf
```

W poleceniu tym mona rwnie wstawi opcj `-dPDFSETTINGS=/prepress` (zapewniajc uzyskanie wysokiej jakoci, zachowanie kolorw, uzyskanie obrazkw w rozdzielczoci 300 dpi). Ze wzgldw licencyjnych ghostscript uywa domylnie algorytmw z kompresj stratn. Przy kompresji moe wic doj do utraty jakoci bitmap.

Rozdział 6

Podsumowanie

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

6.1. Sekcja poziomu 1

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Nam id nulla a adipiscing tortor, dictum ut, lobortis urna. Donec non dui. Cras tempus orci ipsum, molestie quis, lacinia varius nunc, rhoncus purus, consectetuer congue risus.

6.1.1. Sekcja poziomu 2

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Sekcja poziomu 3

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Paragraf 4 Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

6.2. Sekcja poziomu 1

Lorem ipsum dolor sit amet eleifend et, congue arcu. Morbi tellus sit amet, massa. Vivamus est id risus. Sed sit amet, libero. Aenean ac ipsum. Mauris vel lectus.

Literatura

[1] A. Barth. Rfc 6265-http state management mechanism. *Internet Engineering Task Force (IETF)*, strony 2070–1721, 2011.

[2] D. Bermbach, E. Wittern. Benchmarking web api quality. *International Conference on Web Engineering*, strony 188–206. Springer, 2016.

[3] G. Bierman, M. Abadi, M. Torgersen. Understanding typescript. *European Conference on Object-Oriented Programming*, strony 257–281. Springer, 2014.

[4] J. Bogard. Mediatr library documentation. `https://github.com/jbogard/MediatR/wiki`. Dostęp z dnia: 2022-03-27.

[5] V. Bojinov. *RESTful Web API Design with Node. js 10: Learn to create robust RESTful web services with Node. js, MongoDB, and Express. js*. Packt Publishing Ltd, 2018.

[6] M. Casciaro, L. Mammino. *Node. js Design Patterns: Design and implement production-grade Node. js applications using proven patterns and techniques*. Packt Publishing Ltd, 2020.

[7] Cisco. Cisco annual internet report (2018–2023) white paper. *Cisco: San Jose, CA, USA*, 2018.

[8] R. Fielding, J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content, 2014.

[9] R. T. Fielding, J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, Czerw. 2014.

[10] D. Gourley, B. Totty, M. Sayer, A. Aggarwal, S. Reddy. *HTTP: the definitive guide*. Ó'Reilly Media, Inc.", 2002.

[11] E. H. Halili. *Apache JMeter*. Packt Publishing Birmingham, 2008.

[12] A. Hejlsberg, S. Wiltamuth, P. Golde. *C# language specification*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[13] B. Hetzel. *The complete guide to software testing*. QED Information Sciences, Inc., 1988.

[14] S. Holmes. *Mongoose for Application Development*. Packt Publishing Ltd, 2013.

[15] T. Hyttinen. .net core 3.1 &. net 5: Performance benchmarking in web api use. *Bachelor's Thesis*, 2021.

[16] International Organization for Standardization. ISO/IEC 25010:2011, systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models, 2011.

[17] S. H. Jensen, A. Møller, P. Thiemann. Type analysis for javascript. *International Static Analysis Symposium*, strony 238–255. Springer, 2009.

[18] C. Kambalyal. 3-tier architecture. *Retrieved On*, 2(34):2010, 2010.

[19] B. Lakshmiraghavan. *Pro Asp. Net Web API Security: Securing ASP. NET Web API*. Apress, 2013.

[20] D. Laksono. Testing spatial data deliverance in sql and nosql database using nodejs fullstack web app. *2018 4th International Conference on Science and Technology (ICST)*, strony 1–5. IEEE, 2018.

[21] A. Mardan. *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[22] E. Miller. Introduction to software testing technology. *Tutorial: Software Testing & Validation Techniques, Second Edition, IEEE Catalog No. EHO*, strony 180–0, 1981.

[23] G. J. Myers, C. Sandler, T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[24] A. Neumann, N. Laranjeiro, J. Bernardino. An analysis of public rest web service apis. *IEEE Transactions on Services Computing*, 2018.

[25] D. Nevedrov. Using jmeter to performance test web services. *Published on dev2dev*, strony 1–11, 2006.

[26] K. Prasad. *Software Testing Tools: Covering WinRunner, Silk Test, LoadRunner, JMeter and TestDirector with case studies w/CD*. Dreamtech press, 2004.

[27] G. e. a. Rauch. Prisma orm library documentation. `https://www.prisma.io/docs/reference`. Dostęp z dnia: 2022-03-27.

[28] A. Roman. *Testowanie i jakość oprogramowania: modele, techniki, narzędzia*. Warszawa: Wydawnictwo Naukowe PWN, 2015.

[29] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, X. Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[30] A. Singjai, U. Zdun, O. Zimmermann, M. Stocker, C. Pautasso. Patterns on designing api endpoint operations. *28th Conference on Pattern Languages of Programs (PLoP)*, October 2021.

[31] J. P. Smith. *Entity Framework core in action*. Simon and Schuster, 2021.

[32] A. Spillner, T. Linz. *Software Testing Foundations: A Study Guide for the Certified Tester Exam-Foundation Level-ISTQB® Compliant*. dpunkt. verlag, 2021.

[33] H. Subramanian, P. Raj. *Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs*. Packt Publishing Ltd, 2019.

[34] A. Torres, R. Galante, M. S. Pimenta, A. J. B. Martins. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, 82:1–18, 2017.

[35] A. Troelsen, P. Japikse. *Pro C# 7: With. net and. net Core*, wolumen 1328. Springer, 2017.

[36] H. Wang, J. Z. Huang, Y. Qu, J. Xie. Web services: problems and future directions. *Journal of Web Semantics*, 1(3):309–320, 2004.

[37] J. Webber, S. Parastatidis, I. Robinson. *REST in practice: Hypermedia and systems architecture*. Ó'Reilly Media, Inc.", 2010.

[38] M. Zacharuk. Przedstawienie i rozwiązanie problemu testowania wydajności aplikacji internetowej opartej na rest api w środowisku rozproszonym. *Bachelor's Thesis*, 2020.

Dodatek A

Instrukcja wdroeniowa

Jeli praca skoczya si wykonaniem jakiego oprogramowania, to w dodatku powinna pojawi si instrukcja wdroeniowa (o tym jak skompilowa/zainstalowa to oprogramowanie). Przydaoby si rwnie krtkie how to (jak uruchomi system i co w nim zrobi – zademonstrowane na jakim najprostszym przypadku uycia). Mona z tego zrobi osobny dodatek,

Dodatek B

Opis zaczonej pyty CD/DVD

Tutaj jest miejsce na zamieszczenie opisu zawartoci zaczonej pyty. Naley wymieni, co zawiera.