JOURNAL of COMPUTER SCIENCES INSTITUTE

JCSI 12 (2019) 206-213
Wysłane: 2019-06-03
Przyjęte: 2019-06-11

Metody zwiększające wydajność i bezpieczeństwo aplikacji internetowych

Tomasz Machulski*, Grzegorz Nowakowski*, Maria Skublewska-Paszkowska
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Artykuł prezentuje metody zwiększające wydajność i bezpieczeństwo aplikacji internetowych oraz ocenia ich wartość i zasadność użycia. Każda z przedstawionych metod została zaimplementowana w aplikacji testowej. W pracy zostały przedstawione wyniki badań porównujących stan aplikacji przed i po implementacji danej metody. Na podstawie rezultatów badań sformułowano wnioski dotyczące wpływu metod wydajności i bezpieczeństwa na zachowanie aplikacji testowej.

Słowa kluczowe: wydajność aplikacji internetowych; bezpieczeństwo; aplikacje internetowe

*Autor do korespondencji.

Adresy e-mail: tomaszekem2@gmail.com, grz55@vp.pl

Methods of enhancing the performance and security of web applications

Tomasz Machulski*, Grzegorz Nowakowski*, Maria Skublewska-Paszkowska
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. This article presents the methods of enhancing the performance and security of web applications. It also evaluates them and describes how to use them. The methods have been implemented in a test application. The article presents results of the research comparing state of the application before and after implementation of every listed method. Conclusions about impact of the methods of enhancing performance and security of web applications are based on the results of the research.

Keywords: performance of web applications; security; web-applications

*Corresponding author.

E-mail addresses: tomaszekem2@gmail.com, grz55@vp.pl

1. Wstęp

Rozwój Internetu oraz rosnąca prędkość łącz spowodował ogromny wzrost zainteresowania siecią Web jako obszarem funkcjonowania i utrzymania aplikacji. Jeszcze niedawno programy komputerowe instalowane były w większości na dyskach lokalnych, całość pracy programu wykonywana była przez komputer użytkownika, a aplikacje rzadko kiedy łączyły się z siecią. Wprowadzanie zmian i poprawek w takich programach, było utrudnione. Producentowi łatwiej zarządzać, modyfikować i uaktualniać program znajdujący się na jego serwerze, niż wprowadzać zmiany w nierzadko milionach komputerów.

Przeniesienie funkcjonowania aplikacji na serwer jest zaletą nie tylko dla właściciela oprogramowania, ale także użytkownika. Centralne utrzymania aplikacji przez twórcę oraz możliwość zdalnej pracy przez klienta to tylko jedne z przykładów, które zaważyły o sukcesie, które odniosły aplikacje internetowe.

Internet zawiera bardzo wiele ważnych danych. Użytkownik nierzadko zostawia tam swoją tożsamość lub wykonuje operacje finansowe na porządku dziennym. Ogromna liczba urządzeń użytkowników podłączonych do Internetu implikuje sytuację, że bazy danych przechowują coraz większe zbiory danych. Biorąc pod uwagę oba fakty, naturalnym jest stwierdzenie, że wydajność i bezpieczeństwo aplikacji internetowych jest bardzo znaczące. Rozwój technologii powoduje, że powstają coraz nowsze mechanizmy tworzenia aplikacji internetowych, jak i zwiększania ich wydajności. Mnogość informacji zawartych w bazach danych wymaga ścisłego kontrolowania wydajności systemu, mimo znacząco rosnącej prędkości dysków, procesorów i łączy sieciowych. Ważność danych znajdujących się na serwerach przyciąga hakerów, przez co na utrzymujących oprogramowanie internetowe ciąży duża odpowiedzialność zapewnienia bezpieczeństwa danych.

Ukazane przypadki wskazują, jak ważnym zagadnieniem jest utrzymanie wysokiej wydajności i bezpieczeństwa aplikacji internetowych. Istnieje wiele metod wspomagających oba zagadnienia. Dynamiczny rozwój Internetu sprawia jednak, że wymagają one ciągłych badań i rozwoju. Tematem pracy jest przeprowadzenie badań i zweryfikowanie postawionych tez: „Metody wydajnościowe frameworków Hibernate i Spring Boot w bardzo znaczący sposób poprawiają szybkość działania aplikacji internetowych", oraz „Metody zwiększające bezpieczeństwo są skutecznym sposobem ochrony aplikacji internetowych". Do zawartych w pracy metod zwiększających bezpieczeństwo należą: haszowanie haseł w bazie danych, użycie JSON Web Token, ustanowienie uprawnień użytkowników, konfiguracja nagłówków http oraz ustanowienie uprawnień użytkowników w bazie danych. Rosnący zbiór danych będących w sieci powoduje, że

zagadnienia ujęte w tezach mają obecnie bardzo duże znaczenie.

2. Przegląd literatury

Informatyka rozwija się bardzo szybko, co jest skutkiem prowadzenia badań tej dziedziny na szeroką skalę. Nie inaczej jest z obszarem aplikacji internetowych. Istnieje wiele sieciowych katalogów prac naukowych, takich jak IEEE Xplore, Scopus lub Web of Science, które publikują tematy w zbliżonej dziedzinie poprawy wydajności i bezpieczeństwa aplikacji internetowych.

Wydajność i optymalizacje aplikacji internetowych opartych o platformę Java EE były analizowane przez autorów publikacji [1]. Badano czasy odpowiedzi serwera przez zasymulowanie jego obciążenia za pomocą aplikacji JMeter. Badaną aplikacją była platforma dla studentów i wykładowców umożliwiająca udostępnianie ocen i obecności. Analiza wyników wskazuje, że baza danych jest głównym czynnikiem mającym wpływ na wydłużenie czasu odpowiedzi serwera przy wzroście obciążenia.

Analizę narzędzi używanych w wykrywaniu podatności na ataki na bazy danych aplikacji internetowych prezentuje praca [2]. Wysiłek autorów skupiony został na testowaniu zachowania aplikacji podczas ataku. Nie zostały tu poruszone zagadnienia metod zabezpieczających.

Przedstawienie sposobu projektowania i budowy systemów zabezpieczających aplikacje internetowe zawiera artykuł [3]. Praca przedstawia ogólny, teoretyczny proces modelowania zabezpieczeń w aplikacjach internetowych. Autorzy wskazują podstawowe metody zabezpieczające, lecz nie przeprowadzają ich głębszej analizy.

Metoda puli połączeń jako wysoce usprawniająca wydajność aplikacji internetowej wskazana zostaje w pracy [4]. Prawdziwość tezy o skuteczności tej metody jest udowodniona badaniami z użyciem narzędzia JMeter. Jest to wartościowe badanie, zawierające przykłady implementacji metody oraz jasno sformułowane wyniki, jednak nie wyczerpuje całkowicie tematu usprawniania wydajności aplikacji webowych.

Ukazane badania oraz wiele innych znajdujących się w licencjonowanych bazach danych przedstawiają głównie pojedyncze metody usprawniające i nie przedstawiają skali różnic przed oraz po ich zastosowaniu. Brakuje zgrupowania najlepszych metod i jasnego określenia stopnia w jaki zwiększają wydajność oraz bezpieczeństwo aplikacji webowych, dlatego uzasadnionym jest przeprowadzanie dalszych badań i analiz w tym kierunku.

W pracy zostały przedstawione i zanalizowane metody zwiększające wydajność należące do frameworków Hibernate oraz Spring. Metody zapewniające bezpieczeństwo dotyczą back-endu aplikacji oraz bazy danych.

3. Metody zwiększające wydajność

Do budowy i rozwijania aplikacji internetowych stosuje się frameworki, które stanowią szkielet aplikacji. Frameworki Hibernate i Spring Boot posiadają wiele możliwości usprawnienia wydajności. Użytkownicy wybierają odpowiednie metody w zależności od obszaru aplikacji, w której wydajność powinna być zwiększona.

3.1. Metody bazodanowe

W bazach danych globalnych systemów komercyjnych znajduje się ogromna liczba rekordów. Wraz ze wzrostem ilości danych utrudnione staje się przeszukiwanie danych w realnym czasie. Odpowiedzią na ten problem jest użycie indeksów bazodanowych.

Indeksowanie polega na utworzeniu odrębnej struktury danych, która przechowuje wartość danej kolumny oraz wskaźnik do rekordu z którym jest powiązany [5]. Struktura jest automatycznie sortowana binarnie po każdej manipulacji na danych tabeli. Indeksowanie należy wprowadzać rozważnie, gdyż indeksy zwiększają rozmiar danych. Są one automatycznie tworzone przez system bazodanowy na kolumnach zawierających wartości klucza głównego lub klucza unikalnego.

Wykorzystane polecenia języka SQL dodające indeksy w tabelach testowej bazy danych przedstawia przykład 1.

Przykład 1. Ustawienie indeksów na kolumnach w tabelach bazy danych

```
CREATE INDEX idx_contr_branch_id ON contract(branch_id);
CREATE INDEX idx_branch_name ON branch(name);
```

3.2. Metody we frameworku Hibernate

3.2.1. Pamięć podręczna II poziomu

Częste odwołania do bazy danych przez ich wysoki koszt, są niepożądane. Należy przez to minimalizować liczbę odniesień do bazy danych. Pamięć podręczna (ang. cache) jest usługą aplikacji pozwalającą na tymczasowe składowanie danych pochodzących ze źródła o wolniejszym dostępie [6]. Odczyt z danych zapisanych w cache jest dużo szybszy od dostępu bazodanowego.

Framework Hibernate w podstawowej konfiguracji wykorzystuje cache I poziomu. Podczas każdej wymiany informacji między aplikacją, a bazą danych tworzona jest sesja. W trakcie jej działania wszystkie wyniki zapytań do bazy danych są zapisywane do cache i poziomu.

W programie może istnieć wiele sesji równocześnie, a każda z nich ma wydzielony indywidualnie cache dla własnego użytku. Ponowne zapytanie o te same dane w ciągu tej samej sesji nie wymaga komunikacji z bazą danych.Po zakończeniu sesji dane w cache I poziomu zostają tracone.

W przypadku użycia pamięci podręcznej II poziomu, w aplikacji powstaje cache globalny, współdzielony przez wszystkie sesje. Sposób działania wszystkich poziomów cache jest identyczny, różnią się one tylko zasięgiem [7].

Objęcie danych pochodzących z tabeli bazodanowej mechanizmem pamięci podręcznej drugiego poziomu uzyskano dzięki dodaniu adnotacji *@Cache* nad nagłówkiem klasy mapującej tą tabelę. Przedstawiono to na przykładzie 2.

Przykład 2. Objęcie klasy reprezentującej tabelę bazodanową mechanizmem pamięci cache II poziomu

```
@Entity
@Table(name = "branch")
@Cache
public class Branch {
    @Id
    private Long id;
}
```

3.2.2. Cachowanie danych słownikowych

W bazie danych istnieją rekordy niezmienne. Pomimo częstego dostępu do nich, ich wartości nigdy nie są nadpisywane, a tylko odczytywane. Duży koszt dostępu do bazy danych sprawia, że zasadnym staje się przeniesienie tych danych do pamięci podręcznej na cały czas działania aplikacji [8].

Do zarejestrowania encji bazodanowej na stałe w cache przez Hibernate, użyto adnotacji *@Cache* z opcją *READ_ONLY* nad klasą reprezentującą tabelę. Zostało to przedstawione na przykładzie 3.

Przykład 3. Objęcie klasy reprezentującej tabelę bazodanową mechanizmem cache'owania danych słownikowych

```
@Entity
@Table(name = "authority")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class Authority implements Serializable {
    @Id
    private String name;
}
```

3.2.3. Query cache

W bazie danych istnieją tabele, których rekordy rzadko ulegają zmianom. Częsty dostęp do nich może być kosztowny. Rzadka zmiana rekordów implikuje małą częstotliwość zmiany wyników zapytań, dlatego framework Hibernate umożliwia przechowywanie tych wyników przez określony czas. Podczas wykonywania zapytania zarejestrowanego przez Query cache, jego wynik ładowany jest z pamięci podręcznej, co ogranicza liczbę odniesień do bazy danych [9]. Istnieje przez to ryzyko uzyskania nieaktualnych danych, dlatego zapytania objęte Query cache nie powinny dotyczyć danych krytycznych do funkcjonowania systemu.

Na przykładzie 4 za pomocą adnotacji *@Cacheable* objęto metodę wykonującą zapytanie do bazy danych. Hibernate rejestruje wyniki tego zapytania w pamięci podręcznej.

Przykład 4. Objęcie metody pobierającej dane z bazy danych mechanizmem Query cache

```
@Repository
interface NewsArticleRepository extends
JpaRepository<NewsArticle, Long> {
    String ARTICLES_BY_BRANCH_ID = "articlesByBranchId";
    @Cacheable(cacheNames=ARTICLES_BY_BRANCH_ID)
     Page<NewsArticle> findAllByBranchId(Long branchId,
    Pageable pageable);
}
```

3.2.4. Ładowanie danych

Podczas korzystania z frameworka Hibernate ładowanie danych ze źródeł może odbywać się według dwóch wzorców. Wyróżnia się zachłanny (ang. eager) oraz leniwy (ang. lazy) tryb dostępu do danych [10]. Pierwszy z nich inicjalizuje oraz przygotowuje wszystkie dane natychmiastowo i zwraca gotowy do wykorzystania obiekt reprezentujący daną encję. Zaleca się stosować go w przypadku pewności, że dane zostaną wykorzystane natychmiastowo.

Z kolei leniwy dostęp do danych przygotowuje obiekt z podstawowymi danymi, lecz odracza jego inicjalizację. Odraczane dane są asocjacjami pomiędzy encjami (relacje jeden-do-wielu, wiele-do-wielu). Ich inicjalizacja następuje dopiero podczas bezpośredniego do nich odniesienia.

Każdy z trybów dostępu posiada zalety i wady, dlatego należy stosować je zgodnie z przeznaczeniem. Ładowanie zachłanne może skrócić czas dostępu do danych, lecz odbywa się to nierzadko kosztem ładowania zbyt dużej ich ilości, co może powodować spadki wydajności. Ładowanie leniwe konsumuje mniej pamięci, gdyż większość ładowanych danych jest niezainicjalizowana. Dostęp do nich wymaga dodatkowego czasu, co może potencjalnie wpływać na spadek wydajności odczytu.

Na przykładzie 5 znajduje się klasa *Client* reprezentująca rekord tabeli przechowującej dane o użytkownikach. Pola tej klasy które mapują relacje wiele-do-wielu lub wiele-do-jednego reprezentowane są jako kolekcje w języku Java. Oznaczono tryb odczytu tych kolekcji jako leniwy (*LAZY*), tak aby ich odczyt został odroczony do momentu pierwszego odniesienia do nich.

Przykład 5. Objęcie klasy reprezentującej tabelę bazodanową mechanizmem cache'owania danych słownikowych

```
@Entity
public class Client extends User {
    @ManyToMany(
        cascade = CascadeType.ALL,
        fetch = FetchType.LAZY)
    private Set<GymClass> classes = Sets.newHashSet();
```

3.3. Metody we frameworku Spring Boot

3.3.1. Mechanizm Spring Cache

Framework Spring podobnie jak Hibernate wspiera mechanizm wykorzystania pamięci podręcznej. Spring Cache pozwala przechowywać wyniki metod klas wykonujących

kosztowne obliczenia [11,12]. Rzadsze wykonywanie obliczeń redukuje wykorzystanie mocy obliczeniowej serwera. Czas przechowywania wyniku w pamięci podręcznej ustala się w konfiguracji mechanizmu Spring Cache.

Na przykładzie 6 przedstawiono funkcję umieszczającą do pamięci podręcznej instancję obiektu będącego rezultatem obliczeń. Obiekt *cacheManager* przechowuje wartości obliczeń jako para klucz-wartość.

Przykład 6. Ładowanie obiektu do pamięci podręcznej za pomocą mechanizmu Spring Cache

```
public void setAsCurrent(Branch branch) {
    var cache= cacheManager.getCache(BRANCH_CACHE);
    cache.clear();
    cache.put(CURRENT_BRANCH_KEY, branch);
}
```

3.3.2. Paginacja danych

Paginacja jest mechanizmem polegającym na ładowaniu ograniczonej ilości danych pochodzących z bazy danych. Użytkownik aplikacji internetowej nie potrzebuje zwykle wizualizacji wszystkich danych jednocześnie. Dzięki temu aplikacja może pobierać dane z bazy danych porcjami i eliminuje problem wczytywania nadmiaru danych [13]. Wpływa to również pozytywnie na szybkość wyświetlania interfejsu użytkownika oraz redukuje obciążenie bazy danych.

Mechanizm paginacji został zaimplementowany we frameworku Spring. Polega na przekazaniu argumentu typu *Pageable* poczynając od metody klasy kontrolera aż po metodę klasy wykonującej zapytanie do bazy danych. Instancja tego argumentu jest tworzona automatycznie przez framework Spring na podstawie parametrów żądania http. Metodę klasy kontrolera obsługującą paginację zaprezentowano na przykładzie 7.

Przykład 7. Metoda klasy kontrolera obsługująca paginację danych za pomocą parametru klasy *Pageable*

```
@GetMapping("/users")
List<UserDTO>getUsers(Pageable pageable) {
    var page = userService.getAllManagedUsers(pageable);
    return page.getContent();
}
```

4. Metody zwiększające bezpieczeństwo

4.1. Haszowanie haseł w bazie danych

Dostęp do większości aplikacji internetowych zabezpieczany jest mechanizmem logowania. Sposobem na utrudnienie odczytania hasła, nawet po jego wykradzeniu jest haszowanie. Polega ono na użyciu tzw. funkcji skrótu, która generuje unikalny ciąg znaków dla dowolnego ciągu wejściowego [14]. Funkcja ta jest nieodwracalna. Na podstawie uzyskanego wyjściowego ciągu znaków nie można określić wejściowego ciągu znaków. Wszelkie próby operacji odwrotnych wymagają ogromnej mocy obliczeniowej i przez to stają się niedostępne dla zwykłych użytkowników.

W aplikacji testowej wykorzystano mechanizm haszowania haseł zaimplementowany we frameworku Spring. Na przykładzie 8 znajduje się fragment kodu odpowiadający za hashowanie podanego hasła podczas rejestracji użytkownika. Metoda *encode* obiektu *passwordEncoder* wykorzystuje funkcję hashującą *bcrypt*.

Przykład 8. Kod odpowiadający za haszowanie hasła użytkownika

```
var passwordHash = passwordEncoder.encode(password);
```

4.2. JSON Web Token

Każda aplikacja internetowa powinna chronić swoje dane przed niepowołanym dostępem. Stosuje się do tego autentykację polegającą na potwierdzaniu tożsamości użytkownika oraz autoryzację, która nadaje dostęp do systemu.

JSON Web Token (JWT) jest standardem definiującym bezpieczną wymianę informacji pomiędzy klientem, a serwerem [15]. Nośnikiem tej informacji jest obiekt JSON. Po poprawnym zalogowaniu się, klient otrzymuje token będący zaszyfrowanym obiektem JSON. Token przechowuje dane zaszyfrowane kluczem szyfrującym, który znajduje się na serwerze. Tylko on potrafi zweryfikować poprawność tokenu przychodzącego wraz z żądaniami http.

JWT dzięki użyciu klucza szyfrującego nie jest możliwy do spreparowania. Odszyfrowanie danych przez serwer stanowi wystarczające zweryfikowanie tożsamości użytkownika bez konieczności ponownych zapytań do bazy [16]. Oprócz zapewnienia bezpieczeństwa, daje to pozytywny wpływ na wydajność zapytań http.

Obiekt JSON składa się z nagłówka, danych oraz podpisu cyfrowego serwera, co przedstawiono na rysunku 2.

Rys. 1. Przykładowy obiekt JSON

Na przykładzie 9 przedstawiono fragment kodu generującego token po pomyślnym weryfikacji poświadczeń użytkownika.

Przykład 9. Generowanie tokena po weryfikacji tożsamości użytkownika

```
return Jwts.builder()
    .setSubject(authentication.getName())
    .claim(AUTHORITIES_KEY, authorities)
    .signWith(key, SignatureAlgorithm.HS512)
    .setExpiration(validity)
    .compact();
```

4.3. Uprawnienia użytkowników

Aplikacje internetowe udostępniają funkcje wielu grupom użytkowników. Użytkownicy nie powinni posiadać dostępu do funkcji oraz zasobów im nieprzeznaczonych. Aby uporządkować ten dostęp definiuje się role, które przydziela się użytkownikom. Najczęściej tworzony jest podział na administratorów oraz użytkowników. Administrator ma kontrolę nad funkcjami zarządzającymi systemem oraz nadaje lub ogranicza dostęp do zasobów systemowych innym użytkownikom [17]. Użytkownicy najczęściej posiadają uprawnienia niezbędne do korzystania z funkcji oferowanych przez aplikację.

Istnieje wiele narzędzi zawierających gotowe implementacje mechanizmu uprawnień. Jedną z nich jest ta oferowana przez framework Spring. Opiera się ona na wykorzystaniu adnotacji języka Java, które wskazują rolę użytkownika niezbędną do uruchomienia funkcji aplikacji.

Na przykładzie 10 przedstawiono użycie adnotacji *@PreAuthorize* nad metodą klasy kontrolera, która wykonuje funkcję przeznaczoną tylko dla administratora.

Przykład 10. Objęcie funkcji przeznaczonej tylko dla administratora adnotacją *@PreAuthorize*

```
@PostMapping("/branches")
@PreAuthorize(IS_ADMIN)
public BranchDTO create(BranchDTO branchDTO) {
    return super.create(branchDTO);
}
```

4.4. Konfiguracja nagłówków http

Klient aplikacji internetowej komunikuje się z serwerem poprzez protokół http. Działa on na podstawie architektury klient-serwer. Komunikat http składa się z nagłówków oraz ciała wiadomości. Odpowiedzi żądań http zawierają wiele informacji niezbędnych do prawidłowego zachowania przeglądarki internetowej [18]. Znajdują się one w nagłówkach. Domyślne zachowanie przeglądarki podatne jest na wiele ataków m.in. Cross Site Scripting (XSS), który polega dołączeniu do wyświetlanej treści strony internetowej złośliwego skryptu [19]. Na przykładzie 11 przedstawiono fragment kodu konfiguracji serwera, który ustawia nagłówek *X-XSS-Protection* dla każdej odpowiedzi.

Przykład 11. Konfiguracja serwera dla generacji nagłówków zabezpieczających

```
void configure(HttpSecurity http){
   http
     .headers()
          .xssProtection()

     .block(true)
     .apply();}
```

Odpowiedź serwera zawierająca nagłówek zabezpieczający przed atakiem XSS znajduje się na rys. 3.

```
HTTP/1.1 200 OK
Server: keycdn-engine
Date: Wed, 09 Mar 2016 20:06:01 GMT
Content-Type: text/html; charset=UTF-8
X-XSS-Protection: 1; mode=block
```

Rys. 2. Odpowiedź serwera wraz z nagłówkami

4.5. Uprawnienia użytkowników w bazie danych

Architektury aplikacji internetowych składają się z wielu warstw. Zabezpieczenie każdej z nich zmniejsza ryzyko niepowołanego dostępu do systemu z zewnątrz. Baza danych stanowi część systemu, do której dostęp powinien być najściślej zabezpieczony. Wyciek danych wiąże się poważną utratą zaufania użytkowników i zaburzeniem działania całego systemu.

Systemy bazodanowe oferują mechanizmy nadawania uprawnień użytkownikom [20]. Uprawnienia określają dostęp do zbioru działań możliwych do zrealizowania na schemacie bazy danych. Użytkownicy bazy danych przynależą do ról [21]. Każda z nich ma zdefiniowany zbiór przywilejów, które administrator może swobodnie przydzielać lub odbierać.

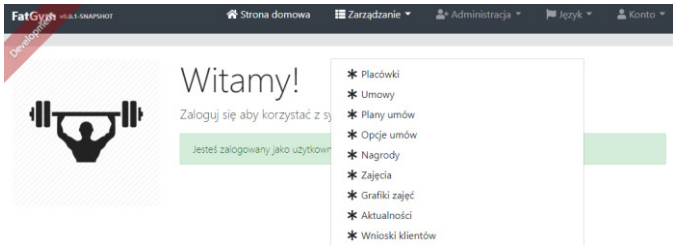
Przydział przywilejów do zdefiniowanej roli przedstawiony został na rysunku 4.

```
GRANT create table, create view
TO manager;
Grant succeeded.
```

Rys. 3. Przydział przywilejów do zdefiniowanej roli

5. Metoda badań

Aplikacja o tytule *FatGym*, wykorzystana do testowania oraz implementacji metod usprawniających jest systemem zarządzania międzynarodową siecią siłowni. Do jej implementacji wykorzystano język Java oraz frameworki Spring oraz Hibernate. Główne menu programu zostało przedstawione na rysunku 5.



Rys. 4. Menu główne aplikacji FatGym

Do przeprowadzania badań metod zwiększających wydajność postępowano według schematu, który obejmował:

- jednorazowe wypełnienie biorących udział w badaniu tabel bazy danych generowanymi danymi symulującymi globalny system obsługi klientów
- pomiar wydajności przed implementacją metody usprawniającej
- implementacja wybranej metody
- pomiar wydajności po implementacji metody usprawniającej
- powrót do stanu aplikacji sprzed implementacji metody usprawniającej.

Pomiar czasów odbywał się za pośrednictwem programu JMeter. Program generował i rozpoczynał wykonywanie 100 wątków równocześnie. Powtarza tą akcję co sekundę przez 10 sekund. W przypadku niedokończenia wykonywania wątków przed generacją nowych - ich liczba kumuluje się.

Badanie metod zwiększania bezpieczeństwa aplikacji obejmowało implementację każdej z metod oraz sprawdzanie poprawności jej działania.

Otrzymane wyniki analizowano w kontekście przydatności metody i zgodności z jej przeznaczeniem, a później zostały wyciągnięte wnioski. Efekt badań pozwolił zweryfikować postawione na początku prac tezy.

6. Wyniki

6.1. Metody zwiększające wydajność aplikacji

Wyniki badań związanych z metodami zwiększającymi wydajność aplikacji koncentrują się głównie wokół czasów wykonania zapytań bazodanowych oraz czasów odpowiedzi na żądania HTTP.

Podsumowanie czasów przed oraz po implementacji dla każdej metody zwiększającej wydajność przedstawione zostało w tabeli 1.

Różnice pomiędzy średnimi czasami odpowiedzi serwera przed i po implementacji metod zwiększających wydajność obrazuje rysunek 6.

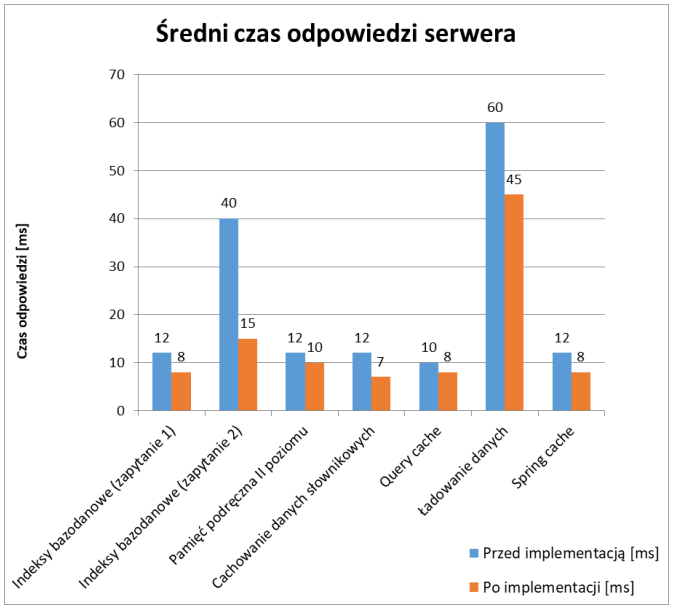
6.2. Metody zwiększające bezpieczeństwo aplikacji

Wyniki badań związanych z metodami zapewniającymi bezpieczeństwo aplikacji koncentrują się wokół sprawdzania poprawności ich działania.

Przeprowadzone testy rejestracji użytkownika wykazały poprawność działania funkcji hashującej BCrypt na podawanych przez użytkowników hasłach. Przetestowano również mechanizm logowania, w którym hash podanego hasła zostaje porównany ze wzorcem znajdującym się w bazie danych. Działanie funkcji przebiega zgodnie z oczekiwaniem.

Tabela 1. Czasy metod zwiększających wydajność aplikacji internetowej

Metoda zwiększająca wydajność	Najwyższy czas odpowiedzi przed implementacją metody [ms]	Średni czas odpowiedzi przed implementacją metody [ms]	Najwyższy czas odpowiedzi po implementacji metody [ms]	Średni czas odpowiedzi po implementacji metody [ms]	Różnica pomiędzy średnimi czasami po i przed implementacją [%]
Indeksy bazodanowe (indeks na kolumnie tekstowej)	55	12	15	8	33.33%
Indeksy bazodanowe (indeks na kluczu obcym)	48	40	22	15	62,50%
Pamięć podręczna II poziomu	19	12	14	10	16,67%
Cachowanie danych słownikowych	28	12	10	7	41,67%
Query cache	160	10	110	8	20.00%
Ładowanie danych	800	60	450	45	25,00%
Spring cache	270	12	110	8	33,33%
Paginacja danych	-	17000	-	9000	47,06%



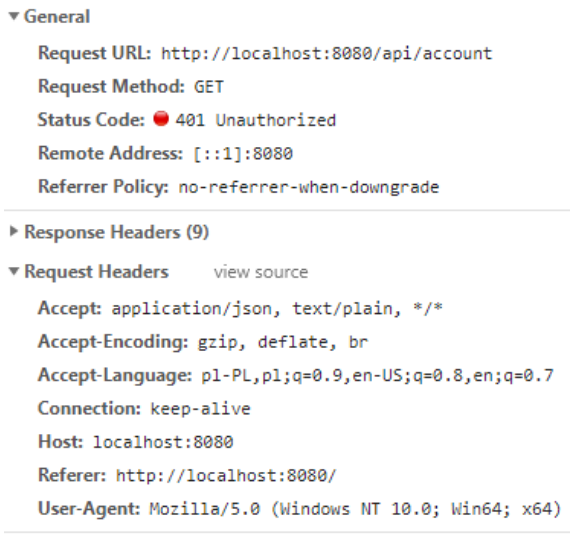
Rys. 5. Porównanie czasów odpowiedzi serwera przed i po implementacji danej metody wydajnościowej

Na rysunku 7 znajduje się fragment tabeli *Users,* która w poprawny sposób przechowuje hashe haseł.

id	login	password_hash
2	anonymoususer	$2a$10$j8S5d7Sr7.8VTOYNviDPOeWX8KcYILUVJBsYV83Y…
4	user	$2a$10$VEjxo0jq2YG9Rbk2HmX9S.k1uZBGYUHdUcid3g/v…
1101	jkowalski	$2a$10$P2ZqM8cad35yfYkWpmfjVu4bXO.ggBduPTKUVU3w…
3	admin	$2a$10$gSAhZrxMllrbgj/kkK9UceBPpChGWJA7SYIb1Mqo…
1	system	$2a$10$mE.qmcV0mFU5NcKh73TZx.z4ueI/.bDWbj0T1BYy…
1401	mpudzianowski	$2a$10$CoNk3VBeump2GUpGs7l0ue7uwsD4XX9DcP8oWzWx…

Rys. 6. Fragment tabeli Users, w której przechowywane są hashe haseł

Testy działania mechanizmu JSON Web Token wykazały odporność aplikacji na nieautoryzowany do niej dostęp. Wysłanie żądania bez tokena kończy się odmową dostępu do systemu, , co prezentuje rysunek 8.



Rys. 7. Odpowiedź na żądanie niezawierające tokena

Testy aplikacji po implementacji uprawnień wykazały poprawne działanie blokady dostępu użytkowników do funkcji przeznaczonych tylko dla administratorów. Próba otwarcia przez użytkownika strony dostępnej tylko dla administratora kończy się odmową dostępu. Odpowiedź na żądanie prezentuje rysunek 9.

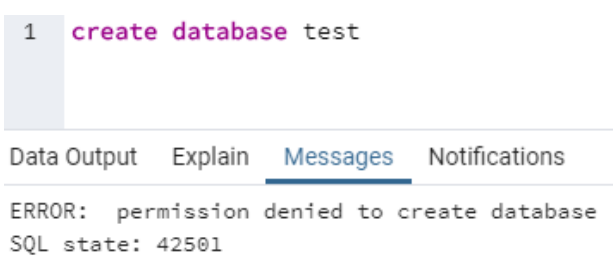
Serwer zwraca odpowiedzi z nagłówkami X-XSS-Protection oraz Strict-Transport-Security. To potwierdza zabezpieczenie aplikacji przed wstrzykiwaniem złośliwego kodu oraz wymusza komunikację z użyciem protokołu https.

Baza danych skutecznie blokuje dostęp aplikacji do funkcji ingerujących w schemat bazy danych. Próba utworzenia nowej bazy danych kończy się odmową dostępu, co prezentuje rysunek 10.

7. Wnioski

Aplikacja *FatGym* okazała się odpowiednią do implementacji metod zwiększających wydajność i bezpieczeństwo. Implementacja wszystkich metod przebiegła pomyślnie.

Rys. 8. Odpowiedź na próbę otwarcia strony dostępnej tylko dla administratora przez użytkownika



Rys. 9. Odmowa dostępu do akcji niedostępnych dla zwykłego użytkownika

Na podstawie badań przeprowadzonych można sformułować następujące wnioski:

- wraz ze wzrostem liczby użytkowników systemu rośnie potrzeba zapewnienia wysokiej wydajności systemu
- metody zwiększające wydajność, będące integralną częścią frameworków Spring i Hibernate oferują zauważalną poprawę wydajności aplikacji (uzyskano poprawę wydajności średnio o 25%)
- stosowanie metod poprawy wydajności na wielu poziomach (back-end i baza danych), potęguje wzrost szybkości działania aplikacji (uzyskano poprawę wydajności średnio o 30%)
- metody poprawy wydajności powinny być implementowane zgodnie z ich przeznaczeniem;
- użycie metody poprawy wydajności niezgodnie z jej przypadkiem użycia może nawet spowolnić działanie aplikacji
- przedstawione w pracy metody zapewniania bezpieczeństwa aplikacji internetowej powinny stanowić podstawę jej działania
- prezentowane metody dotyczące bezpieczeństwa skutecznie zabezpieczają aplikację przed większością zagrożeń

- aplikacje internetowe powinny być zabezpieczane na wszystkich poziomach: od front-endu, przez back-end do bazy danych.

Podsumowując wszystkie badania i wnioski, potwierdzone zostają następujące tezy: „Metody wydajnościowe frameworków Hibernate i Spring Boot w bardzo znaczący sposób poprawiają szybkość działania aplikacji internetowych" oraz „Metody zwiększające bezpieczeństwo są skutecznym sposobem ochrony aplikacji internetowych". Do metod zwiększających bezpieczeństwo należą: haszowanie haseł w bazie danych, użycie JSON Web Token, ustanowienie uprawnień użytkowników, konfiguracja nagłówków http oraz ustanowienie uprawnień użytkowników w bazie danych. W zależności od użytego mechanizmu udało się skrócić czasy wykonywania żądań od 15% do 60%.

Literatura

[1] Qinglin Wu, Yan Wang; Performance Testing and Optimization of J2EE-based Web Applications; 2010 Second International Workshop on Education Technology and Computer Science; 2010.

[2] Abdulrahman Alzahrani, Ali Alqazzaz, Ye Zhu, Huirong Fu, Nabil Almashfi; Web Application Security Tools Analysis; 2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids); 2017.

[3] Mahesh Bang, Himanshu Saraswat; Building an effective and efficient continuous web application security program; 2016 International Conference On Cyber Situational Awareness, Data Analytics And Assessment (CyberSA), 2016.

[4] Kirti Gupta, Manish Mathuria; Improving Performance of Web Application approaches using Connection Pooling; 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA); 2017.

[5] Smith G., Wysoko wydajny PostgreSQL, Helion, 2011.

[6] Bauer C., Java Persistence: programowanie aplikacji bazodanowych w Hibernate, Helion, 2017.

[7] Hibernate Second-Level Cache | Baeldung, https://www.baeldung.com/hibernate-second-level-cache, [15.05.2019].

[8] Hibernate Cache Strategy Work, https://vladmihalcea.com/how-does-hibernate-read_only-cacheconcurrencystrategy-work/, [12.05.2019].

[9] Hibernate. Caching, https://docs.jboss.org/hibernate/orm/4.0/devguide/en-US/html/ch06.html, [13.05.2019].

[10] Eager/Lazy Loading in Hibernate | Baeldung, https://www.baeldung.com/hibernate-lazy-eager-loading, [14.05.2019].

[11] Mak G., Rubio D., Long J., Spring. Receptury, Helion, 2014

[12] Spring. Caching, https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-caching.html, [14.05.2019].

[13] Spring. Paging and Sorting, https://docs.spring.io/spring-data/rest/docs/2.0.0.M1/reference/html/paging-chapter.html, [15.05.2019].

[14] Lis M., Tworzenie bezpiecznych aplikacji internetowych, Helion, 2014.

[15] JSON Web Token Introduction, https://jwt.io/introduction/, [15.05.2019].

[16] Barnett R., Web Application Defender's Cookbook: Battling Hackers and Protecting Users, Wiley, 2012.

[17] Introducing to Spring Method Security | Baeldung, https://www.baeldung.com/spring-security-method-security, [15.05.2019].

[18] The 8 HTTP Security Headers Best Practices, https://www.globaldots.com/8-http-security-headers-best-practices/, [15.05.2019]

[19] Hope P., Walther B., Testowanie bezpieczeństwa aplikacji internetowych. Receptury, Helion, 2017

[20] Elsmari R., Navathe S., Wprowadzenie do systemów baz danych, Helion, 2019.

[21] Understanding Users, Privileges, and Roles, https://www.vertica.com/blog/understanding-users-privileges-roles/, [14.05.2019].