Politechnika Łódzka

Instytut Informatyki

PRACA DYPLOMOWA INŻYNIERSKA

ZAAWANSOWANY MODEL BEZPIECZEŃSTWA ZASTOSOWANY W PRZYKŁADOWYM WIELODOSTĘPNYM SYSTEMIE INFORMATYCZNYM

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Promotor: dr inż. Mateusz Smoliński
Dyplomant: Przemysław Komuda
Nr albumu: 216802
Kierunek: Informatyka
Specjalność: Infrastruktura i aplikacje sieciowe

Łódź 01.11.2020

Instytut Informatyki
90-924 Łódź, ul. Wólczańska 215, budynek B9
tel. 042 631 27 97, 042 632 97 57, fax 042 630 34 14 email: office@ics.p.lodz.pl

Spis treści

1. Wstęp

1.1. Przedstawienie rozwiązywanego problemu

W dzisiejszych czasach wiele osób korzysta z różnego rodzaju aplikacji sieciowych w życiu zawodowym oraz prywatnym – dla usprawnienia pewnych procesów lub rozrywki. Zakładając konto w jakimkolwiek serwisie należy ustawić silne hasło (np. zgodne z wytycznymi NIST[8]), jednak nie zawsze jest to gwarancją bezpieczeństwa przed włamaniem na nasze konto, które z kolei może mieć katastrofalne skutki. Biznes, dla którego przeznaczona jest aplikacja oraz jej deweloperzy powinni zadbać o ochronę danych użytkowników implementując odpowiednie mechanizmy bezpieczeństwa. W tej dziedzinie swoistym wzorem do naśladowania mogą być systemy bankowe, zwłaszcza po wprowadzeniu unijnej dyrektywy PSD2, wymagającej np. „silnego uwierzytelniania klienta”[6].

Kwestią, którą należy rozważyć jest również komfort korzystania z aplikacji bądź systemu informatycznego. Choć jest to bardzo subiektywna opinia każdego użytkownika, zazwyczaj wygoda nie idzie w parze z bezpieczeństwem. Dla przykładu, gdy podczas zakładania konta na stronie internetowej jest wymagane podanie skomplikowanego hasła, 62% użytkowników uzna to za irytujące, zgodnie z badaniem[5]. Stopień zabezpieczeń najczęściej jest dostosowany do rodzaju systemu informatycznego – zazwyczaj nie znajdziemy biometrii na forum internetowym.

Obecnie aplikacje sieciowe są narażone na wiele ataków[7], do najczęstszych z nich należą między innymi:

- SQL injection – wprowadzenie w pole tekstowe na stronie internetowej kwerend SQL, które zostanę wykonane w bazie danych ze względu na nieodpowiednie filtrowanie danych wejściowych
- Cross site scripting (XSS) – umieszczenie skryptu na stronie internetowej, która zostanie wyświetlona przez innego użytkownika, a ze względu na nieodpowiednie filtrowanie danych wejściowych niechciany skrypt zostanie wykonany
- Cross site request forgery (CSRF) – wykonanie niepożądanej akcji na stronie internetowej, na której użytkownik jest stale uwierzytelniony np. poprzez kliknięcie w odpowiednio spreparowane hiperłącze.

Podczas implementacji zabezpieczeń w systemach informatycznych deweloperzy stają przed dużymi wyzwaniami. Pierwszym i najważniejszym z nich jest zapewnienie ochrony przed atakami. Z drugiej strony biznes, dla którego wytwarzane jest oprogramowanie oczekuje, aby interakcja z systemem była przystępna dla jego użytkowników.

1.2. Alternatywne rozwiązania

Za najbezpieczniejsze systemy często uważane są te, w których dochodzi do obrotu pieniędzmi, takie jak:

- https://www.aliorbank.pl – bankowość internetowa
- https://www.xtb.com – platforma maklerska

Główną inspiracją do stworzenia opisywanego systemu informatycznego był mechanizm uwierzytelniania wykorzystany w Alior Banku, Pekao S.A. oraz kilku innych systemach polskich banków. Wymaga on podania unikalnego identyfikatora użytkownika, wybranych znaków hasła zamiast jego pełnej formy (nazywanych hasłem maskowanym) oraz jednorazowego kodu SMS. Taka metoda znacząco utrudnia niepożądany dostęp do czyjegoś konta, przez co klienci czują się dużo bardziej bezpiecznie.

2. Cel i zakres pracy

2.1. Cel pracy

Celem pracy było stworzenie oraz opisanie wielodostępnego systemu informatycznego, który zapewnia spójność danych, wysoki poziom zabezpieczeń oraz utrudnia włamanie się na konto któregoś z użytkowników. Spójność danych została zapewniona poprzez fakt, iż system jest typu OLTP (online transaction processing). Oznacza to, że stan odwzorowywanych obiektów jest taki sam w rzeczywistości, bazie danych oraz warstwie prezentacji. Ponadto został zastosowany mechanizm blokad optymistycznych, który nie dopuszcza do równoległej edycji tego samego obiektu przez wielu użytkowników. Zamysłem podczas implementacji było stworzenie niektórych mechanizmów zabezpieczeń na wzór tych, które są wykorzystywane np. w bankowości internetowej oraz przeniesienie ich do systemu zajmującego się zupełnie inną branżą.

Oprócz tego zostały zastosowane standardowe metody, które mają na celu zadbanie o bezpieczeństwo w systemie, należą do nich między innymi:

- Zasada 3A (uwierzytelnianie, autoryzacja, księgowość)
- Kontrola dostępu RBAC (bazująca na rolach użytkowników).

2.2. Zakres pracy

Praca dzieli się na dwie główne części: implementacyjną i opisową. W części implementacyjnej można wyróżnić następujące etapy:

- Wybór technologii i narzędzi deweloperskich
- Zaprojektowanie systemu informatycznego
- Stworzenie struktur bazy danych
- Napisanie kodu aplikacji
- Konfiguracja docelowego środowiska uruchomieniowego
- Osadzenie systemu w docelowym środowisku.

W części opisowej udokumentowano proces wytwarzania oprogramowania i sposób, w jaki zostały przygotowane zastosowane mechanizmy oraz model biznesowy systemu informatycznego.

3. Założenia

3.1. Opis architektury

System informatyczny stworzony w ramach pracy inżynierskiej opiera się o architekturę trójwarstwową. Składają się na niego:

1. Warstwa prezentacji zrealizowana jako aplikacja internetowa typu SPA[3], poprzez którą użytkownicy prowadzą interakcję z systemem. Wykonywane w niej akcje prowadzą do wywołania metod punktów dostępowych (tzw. „endpointów") REST udostępnionych przez warstwę logiki binzesowej z wykorzystaniem protokołu HTTPS.
2. Warstwa logiki biznesowej zrealizowana jako bezstanowa, monolityczna aplikacja sieciowa, w której odbywa się transakcyjne przetwarzanie danych. Sama w sobie jest podzielona na kolejne trzy warstwy:
   a) Warstwa kontrolerów (punktów dostępowych REST) – umożliwia ona komunikację dalszych warstw ze światem zewnętrznym (w tym przypadku z aplikacją SPA), wywołuje metody warstwy serwisów. Nie są w niej przeprowadzane transakcje aplikacyjne.
   b) Warstwa serwisów – odpowiada za przetwarzanie danych oraz spełnienie reguł biznesowych systemu. Rozpoczynają się w niej transakcje aplikacyjne.
   c) Warstwa repozytoriów – jest pośrednikiem, który zapewnia połączenie warstwy logiki biznesowej z warstwą danych. Przeprowadza operacje zapisu oraz odczytu danych. Kontynuuje transakcję aplikacyjne rozpoczęte przez warstwę serwisów.
3. Warstwa składowania danych – system zarządzania relacyjną bazą danych, w której przechowywane są dane wykorzystywane w aplikacji. Operacje zapisu i odczytu zainicjowane przez warstwę logiki biznesowej, które są wykonywane w ramach transakcji aplikacyjnych powodują rozpoczęcie transakcji bazodanowych, co zapewnia spójność danych w systemie.

3.2. Stos technologiczny

Przed przystąpieniem do implementacji należało wybrać język programowania oraz szkielet budowy aplikacji, który zostanie wykorzystany do stworzenia warstwy logiki biznesowej. Wybór padł na język Java (wersja 11.0.8) oraz Spring Framework[2]. Jego głównym komponentem jest kontener IoC, który zarządza Jako narzędzie do zarządzania zależnościami oraz budowania projektu został użyty Maven (wersja 3.6.3). Ponadto użyte zostało narzędzie Spring Boot (wersja 2.3.5.RELEASE), które jest rozwiązaniem faworyzującym konwencję ponad konfigurację. Dzięki niemu w projekcie jest zawarty wbudowany serwer aplikacyjny Tomcat, który można uruchomić wraz z osadzoną aplikacją przy pomocy jednego polecenia w terminalu. Ponadto zarządza ono

wersjami zależności. W skład Springa wchodzi wiele modułów, a najważniejsze z nich, które zostały wykorzystane w systemie to:

- Spring Data – dostarcza warstwę abstrakcji do komunikacji ze źródłem danych
- Spring Security – biblioteka, przy pomocy której zostały zrealizowane mechanizmy uwierzytelnienia i autoryzacji; wykorzystuje bezpieczeństwo deklaratywne
- Spring AOP – umożliwia stosowanie programowania aspektowego oraz tworzenia własnych adnotacji
- Spring Mail – dostarcza abstrakcję pomocną przy wysyłaniu wiadomości email do użytkowników systemu z poziomu aplikacji

Warto również wspomnieć o bibliotece Hibernate (wersja 5.3.23.Final)[1] będącej implementacją Java Persistence API. Jest to mechanizm do mapowania obiektowo-relacyjnego domyślnie wykorzystywany przez Spring Data, który również został wykorzystany w projekcie.

Do implementacji warstwy składowania danych został wykorzystany system zarządzania relacyjną bazą danych PostgreSQL (wersja 11.6)[4]. Jest to zaawansowany, otwartoźródłowy system, który umożliwia przetwarzanie transakcyjne oraz dobrze integruje się z biblioteką Hibernate.

Ostatnim krokiem był wybór technologii, która zostanie wykorzystania do stworzenia warstwy prezentacji. W tym przypadku postawiono na język programowania JavaScript (wersja ES6) oraz bibliotekę do budowania interfejsów użytkownika React.js (wersja 16.14.0). Do zainicjowania projektu zostało użyte narzędzie Create React App, które w pewnym sensie jest odpowiednikiem Spring Boota. Dzięki niemu do projektu zostaje dodana wstępna konfiguracja narzędzi Webpack do podziału aplikacji na mniejsze pakiety oraz Babel do konwersji kodu JavaScript w wersji ES6 lub wyższej na ES5, czyli zgodną ze starszymi przeglądarkami. Do zarządzania zależnościami zostało wykorzystane narzędzie NPM (wersja 6.14.8), czyli domyślny menedżer pakietów JavaScript. Najważniejsze pakiety wykorzystane w aplikacji warstwy prezentacji to:

- Axios (wersja 0.19.2) – klient HTTP ułatwiający wysyłanie żądań do serwera
- i18next (wersja 19.8.3) – popularna biblioteka dodająca funkcjonalność internacjonalizacji
- yup (wersja 0.29.3) – biblioteka umożliwiająca definiowanie schematów, zgodnie z którymi ma być przeprowadzana walidacja obiektów wysyłanych
- React Bootstrap (wersja 1.4.0) – zapewnia integrację Reacta z biblioteką Bootstrap dostarczającą arkusze stylów, przy pomocy których zbudowanych jest wiele responsywnych stron internetowych.

3.3. Opis środowisk

3.3.1. Środowisko deweloperskie

Jako zintegrowane środowisko programistyczne posłużył program IntelliJ IDEA (wersja 2020.2.3). Zaraz obok Eclipse jest to najpopularniejsze narzędzie na rynku, które zapewnia świetną integrację z Mavenem oraz Springiem. Ze względu na wykorzystanie Spring Boota instalacja samodzielnego serwera aplikacyjnego nie była wymagana. Jeżeli chodzi o bazę danych, to dewelopersko była wykorzystywana lokalna instancja PostgreSQL. Przy implementacji warstwy prezentacji bardzo użytecznym narzędziem okazał się Webpack, które udostępnia serwer deweloperski działający w środowisku Node.js (wersja 12.19.0) umożliwiający przeładowywanie zmian w projekcie podczas działania aplikacji.

3.3.2. Docelowe środowisko uruchomieniowe

Warstwa logiki biznesowej oraz warstwa prezentacji stworzone w ramach pracy inżynierskiej zostały osadzone jako dwie niezależne aplikacje na platformie chmurowej Heroku. Ich kod źródłowy znajduje się w zdalnym repozytorium Git w serwisie GitHub. Do automatycznego osadzania aplikacji został wykorzystany mechanizm GitHub Actions. Repozytorium posiada następującą strukturę katalogów:

- .github/workflows – katalog z konfiguracją GitHub Actions
- backend – katalog z kodem źródłowym aplikacji warstwy logiki biznesowej
- frontend – katalog z kodem źródłowym aplikacji warstwy prezentacji

Mechanizm GitHub Actions został skonfigurowany w taki sposób, aby jakakolwiek zmiana w katalogu frontend lub backend powodowała zbudowanie projektu znajdującego się w tym katalogu oraz osadzenie go na platformie Heroku. Do osadzania aplikacji warstwy logiki biznesowej została wykonana następująca konfiguracja:

```
name: backend

on:
  push:
    branches:
      - master
    paths:
      - 'backend/**'
      - '.github/workflows/backend.yml'

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v1
      with:
        java-version: 11
    - name: Build with Maven
```

```
      run: mvn -B package --file backend/pom.xml
    - name: Add backend remote origin
      run: git remote add heroku-backend https://heroku:$
{{secrets.HEROKU_API_KEY}}@git.heroku.com/$
{{secrets.HEROKU_BACKEND_APP_NAME}}.git
    - name: Deploy backend to Heroku
      run: git push heroku-backend `git subtree split --prefix backend
master`:refs/heads/master –force
```

Listing 1: Konfiguracja GitHub Actions

Platforma chmurowa Heroku oferuje również możliwość korzystania z systemu zarządzania relacyjną bazą danych PostgreSQL w modelu Database as a Service. W wersji darmowej posiada on jednak pewne ograniczenia, takie jak brak możliwości utworzenia wielu użytkowników bazodanowych oraz nadania im uprawnień do tabel. Z tego powodu docelowym środowiskiem, w jakim został uruchomiony RDBMS jest platforma Microsoft Azure. Zawarta w niej usługa Azure Database for PostgreSQL ułatwia konfigurację oraz uruchomienie własnej instancji bazy danych w chmurze Microsoftu. Jest to szybsze rozwiązanie w porównaniu do np. usług oferujących hosting kontenerów Dockera. Ponadto daje pełną kontrolę nad użytkownikami bazodanowymi oraz ich uprawnieniami.

3.4. Reguły biznesowe

Tematyką stworzonego projektu jest wypożyczalnia pojazdów – użytkownicy mogą dokonać rezerwacji na określony pojazd w wybranych przez siebie przedziałach czasowych. System jest typu OLTP (online transaction processing), gdyż zapewnia spójność rzeczywistych obiektów z danymi prezentowanymi użytkownikom oraz danymi w bazie danych. System został podzielony na trzy moduły funkcjonalne:

- MOK – moduł obsługi kont
- MOP – moduł obsługi pojazdów
- MOR – moduł obsługi rezerwacji.

Interakcja z systemem może być prowadzona przez czterech aktorów:

Aktor	Nazwa stosowana w kodzie źródłowym	Opis
Gość	GUEST	Użytkownik nieuwierzytelniony. Nie posiada własnego konta.
Klient	CLIENT	Użytkownik uwierzytelniony. Posiada konto stworzone przez administratora lub w procesie rejestracji. Może wynajmować dostępne pojazdy oraz zarządza własnymi rezerwacjami.
Pracownik	EMPLOYEE	Użytkownik uwierzytelniony. Posiada konto stworzone przez administratora. Zarządza pojazdami oraz rezerwacjami klientów.

Administrator	ADMIN	Użytkownik uwierzytelniony. Posiada konto stworzone przez innego administratora. Zarządza kontami użytkowników.

Tabela 1: Aktorzy

Aktorzy Klient, Pracownik, Administrator odpowiadają dostępnym w aplikacji poziomom dostępu. Jeden użytkownik może posiadać wiele poziomów dostępu. Wszystkie pojazdy mają określoną cenę wynajmu, jest ona naliczana za każdy dzień rezerwacji. Ponadto posiadają one flagę „active” będącą wartością logiczną, która określa czy dany pojazd może zostać wynajęty. Nie ma możliwości usunięcia pojazdu – w przypadku, gdy staje się on niezdatny do użytku wartość flagi „active” zostaje ustawiona na fałsz. Rezerwacja posiada przypisany do niej status, natomiast statusom są przypisane następujące liczby:

0. Złożony
1. Anulowany
2. Zakończony.

3.5. Oferowana funkcjonalność

System oferuje użytkownikom następujące przypadki użycia:

P.U.	Nazwa	Użytkownik nieuwierzytelniony	Klient	Pracownik	Administrator
MOK.1	Zarejestruj	✓			
MOK.2	Zaloguj	✓			
MOK.3	Wyloguj		✓	✓	✓
MOK.4	Zresetuj hasło	✓			
MOK.5	Przełącz poziom dostępu		✓	✓	✓
MOK.6	Utwórz konto				✓
MOK.7	Zablokuj/odblokuj konto				✓
MOK.8	Zmień poziomy dostępu konta				✓
MOK.9	Wyświetl wszystkie konta				✓
MOK.10	Filtruj listę kont				✓
MOK.11	Wyświetl szczegóły własnego konta		✓	✓	✓
MOK.12	Wyświetl szczegóły konta innego użytkownika				✓
MOK.13	Edytuj dane własnego konta		✓	✓	✓

MOK.14	Edytuj dane konta innego użytkownika				✓
MOK.15	Zmień własne hasło		✓	✓	✓
MOK.16	Zmień hasło innego użytkownika				✓
MOK.17	Wyślij maila potwierdzającego				✓
MOP.1	Dodaj pojazd			✓	
MOP.2	Edytuj pojazd			✓	
MOP.3	Aktywuj/dezaktywuj pojazd			✓	
MOP.4	Wyświetl wszystkie pojazdy	✓	✓	✓	✓
MOP.5	Filtruj listę pojazdów	✓	✓	✓	✓
MOP.6	Wyświetl szczegóły pojazdu	✓	✓	✓	✓
MOR.1	Zarezerwuj pojazd		✓		
MOR.2	Zmień status rezerwacji			✓	
MOR.3	Anuluj własną rezerwację		✓		
MOR.4	Wyświetl wszystkie rezerwacje			✓	
MOR.5	Filtruj listę rezerwacji			✓	
MOR.6	Wyświetl własne rezerwacje		✓		
MOR.7	Wyświetl szczegóły rezerwacji		✓	✓	
MOR.8	Edytuj rezerwację		✓		

Tabela 2: Przypadki użycia

3.6. Zastosowany model

Pierwszym aspektem modelu, za sprawą którego można mówić o podniesionym poziomie zabezpieczeń w systemie jest proces rejestracji konta, przez który muszą przejść klienci. W pierwszym kroku wymagane jest podanie podstawowych informacji, takich jak nazwa użytkownika, imię, nazwisko, adres email. Wartości tych pól, tak jak wszystkich innych w systemie są poddawane walidacji zarówno w warstwie prezentacji oraz logiki biznesowej. Kolejnym krokiem jest podanie hasła (wprowadzane znaki są wyświetlane jako kropki) o długości określonej przez parametr uruchomieniowy systemu oraz przejście testu reCAPTCHA. Po zatwierdzeniu użytkownikowi prezentowany jest komunikat mówiący, że do końcówki jego hasła został dopisany losowy ciąg znaków, który jest widoczny na ekranie. Jego długość również jest zdefiniowana

w parametrze uruchomieniowym. Ma to na celu zapobiegnięcie wpisania całego hasła przez użytkownika, co pomaga uchronić go przed atakami hakerów w przypadku, gdy jego komputer jest zainfekowany złośliwym oprogramowaniem typu keylogger. Następnie zostaje wysłane żądanie HTTP typu POST do serwera zawierające wszystkie dane konta użytkownika. Na serwerze nie dochodzi do wygenerowania skrótu całego hasła, lecz wielu skrótów wybranych kombinacji jego znaków. Dla przykładu hasła „Abcdefg!1234” przy następujących wartościach parametrów uruchomieniowych:

MASKED_PASSWORD_MIN_LENGTH=4

MASKED_PASSWORD_MAX_LENGTH=6

FULL_PASSWORD_LENGTH=12,

z hasła zostają wydzielone wszystkie kombinacje 4-, 5- i 6-znakowe oraz przypisane im skonkatenowane indeksy tych znaków w formie listy obiektów typu MaskedPassword. Dla pierwszej 4-znakowej kombinacji obiekt ten będzie miał postać {combination=”0123”, characters=”Abcd”}. Następnie wartości obu atrybutów każdego obiektu z listy są zastępowane ich skrótami wygenerowanymi przy pomocy algorytmu SHA-512 z losowym ciągiem zaburzającym (powszechnie nazywanym solą). Po wykonaniu tych kroków dane użytkownika oraz lista jego haseł maskowanych są zapisywane w bazie danych. W przypadku pomyślnej rejestracji prezentowany jest komunikat z informacją o wysłaniu wiadomości email z hiperłączem potwierdzającym konto oraz kodem QR wygenerowanym przez warstwę logiki biznesowej, który należy zeskanować aplikacją do dwufazowego uwierzytelniania na smartfona, taką jak Google Authenticator czy Authy. Po wykonaniu tych czynności możliwe jest uwierzytelnienie w systemie.

Podczas uwierzytelniania użytkownik najpierw wpisuje swoją nazwę użytkownika, po czym jest wysyłane żądanie HTTP typu POST do serwera, które powoduje wylosowanie kombinacji indeksów odpowiadających znakom hasła, jakie użytkownik powinien wpisać w następnym kroku – jest ona zwracana jako odpowiedź HTTP. Gdy zostanie podana nazwa nieistniejącego użytkownika, również zwracana jest losowa kombinacja indeksów, aby nie można było stwierdzić, czy użytkownik o danej nazwie istnieje. Długość kombinacji również jest liczbą losową z przedziału określonego przez parametry uruchomieniowe [MASKED_PASSWORD_MIN_LENGTH, MASKED_PASSWORD_MAX_LENGTH]. Kombinacja jest zapisywana w bazie danych, aby później zweryfikować, czy użytkownik nie spreparował żądania i nie podał innej kombinacji. Wpisanie jedynie części znaków hasła chroni użytkownika przez złośliwym oprogramowaniem typu keylogger, podobnie jak w przypadku rejestracji. Po wpisaniu znaków hasła następuje przejście do ostatniego kroku, jakim jest podanie jednorazowego kodu o ograniczonym czasie ważności wygenerowanym przez sparowaną wcześniej aplikację do dwufazowego uwierzytelniania na smartfonie. Gdy użytkownik zatwierdzi wpisanie jednorazowego kodu, do serwera wysyłane jest żądanie HTTP typu POST z nazwą użytkownika, kombinacją indeksów znaków hasła, wybranymi znakami hasła oraz jednorazowym kodem w ciele żądania. Serwer będący aplikacją warstwy logiki biznesowej pobiera z bazy obiekt reprezentujący użytkownika na podstawie jego nazwy, wyszukuje skrót kombinacji znaków hasła odpowiadający otrzymanej kombinacji oraz porównuje skrót tych znaków z otrzymanymi znakami. Jeśli te elementy się zgadzają, zostaje przeprowadzona

weryfikacja jednorazowego kodu z aplikacji oraz tego, czy konto użytkownika jest aktywne i potwierdzone. Wszystkie te mechanizmy zostały zintegrowane z biblioteką Spring Security. Dopiero po spełnieniu tych wymagań użytkownik jest poprawnie uwierzytelniony. Oprócz procesu uwierzytelniania, jednorazowe kody z aplikacji na smartfonie zostały również wykorzystane do potwierdzania krytycznych operacji, takich jak złożenie rezerwacji przez klienta.

Ze względu na bezstanową naturę aplikacji warstwy logiki biznesowej, do uwierzytelniania oraz autoryzacji wykorzystano żetony JWT (JSON Web Token) o krótkim czasie ważności (możliwym do skonfigurowania poprzez parametry uruchomieniowe aplikacji warstwy logiki biznesowej). W odróżnieniu od tradycyjnej sesji użytkownika, to rozwiązanie oferuje większą wydajność oraz lepsze możliwości skalowania aplikacji. Działa ono w następujący sposób:

- Klient wysyła żądanie HTTP typu POST z danymi uwierzytelniającymi do odpowiedniego punktu dostępowego serwera.
- Po zweryfikowaniu poprawności danych uwierzytelniających generowany jest żeton będący ciągiem znaków w formacie JSON zawierający nazwę użytkownika, jego poziomy dostępu oraz datę wydania żetonu.
- Żeton zostaje zakodowany w formacie Base64 oraz podpisany przy pomocy algorytmu HMAC-SHA-256.
- Żeton zostaje przesłany klientowi jako odpowiedź HTTP oraz zostaje zapisany w ciasteczku HttpOnly.

Otrzymany żeton jest wysyłany jako nagłówek HTTP w żądaniach do serwera wykonywanych przez użytkownika. Nad każdą metodą punktu dostępowego REST została umieszczona adnotacja bezpieczeństwa deklaratywnego pochodząca z biblioteki Spring Security, która określa jaką rolę musi posiadać użytkownik, aby móc wywołać tę metodę. W żetonie JWT są przechowywane role używane w aplikacji warstwy prezentacji, czyli ADMIN, EMPLOYEE, CLIENT, natomiast w warstwie logiki biznesowej została zastosowana zasada, która określa nową rolę dla każdej metody punktu dostępowego. Mapowanie jednych ról na drugie zostało zdefiniowane w pliku konfiguracyjnym roles.properties. Podczas przeprowadzania autoryzacji następuje odczyt tego pliku i weryfikacja, czy dany użytkownik posiada wymaganą rolę. Spring Security w standardzie nie oferuje takiej funkcjonalności, również jest to autorski mechanizm. Oprócz roli sprawdzane jest, czy żeton nie wygasł oraz czy nie został spreparowany poprzez weryfikację podpisu cyfrowego HMAC-SHA-256.

Kolejnym elementem zastosowanego modelu jest sposób, w jaki została przygotowana warstwa danych systemu. W bazie danych przechowywane są informacje audytowe dotyczące każdego obiektu. Należą do nich:

- tożsamość użytkownika, który stworzył dany obiekt oraz data utworzenia
- tożsamość użytkownika, który jako ostatni modyfikował dany obiekt oraz data tej modyfikacji.

Oprócz tego w bazie danych przechowywane są anonimizowane logi wywołań metod biznesowych zawierające tożsamość użytkownika, który zainicjował to wywołanie wraz z datą wywołania oraz strefą czasową, która została przechwycona z przeglądarki tego użytkownika. W systemie zarządzania relacyjną bazą danych zostały utworzone trzy konta bazodanowe o minimalnych wymaganych uprawnieniach. Każde z nich odpowiada za dostęp do danych przypisanych jednemu z modułów funkcjonalnych systemu.

4. Projekt systemu

4.1. Struktury bazy danych

4.2. Użytkownicy bazodanowi

4.3. Identyfikacja obiektów encji

4.4. Diagram klas encji

4.5. Identyfikacja transakcji bazodanowych

5. Opis implementacji i uruchomienia

5.1. Diagram klas ziaren Springa

5.2. Konfiguracja uwierzytelniania

5.2.1. Implementacja interfejsu AuthenticationProvider

5.2.2. Obsługa jednorazowych kodów

5.3. Konfiguracja autoryzacji

5.3.1. Mapowanie grup na role

5.3.2. Schemat bezpieczeństwa ziaren Springa

5.4. Konfiguracja księgowości

5.4.1. Dane audytowe

5.4.2. Logi i interceptory

5.5. Identyfikacja transakcji aplikacyjnych i ich powtarzanie

5.6. Obsługa błędów

5.7. Interfejs użytkownika

5.8. Walidacja danych

5.9. Internacjonalizacja

5.10. Osadzenie systemu w środowisku uruchomieniowym

6. Podsumowanie

7. Spis tabel

8. Spis listingów

Bibliografia

[1] Christian Bauer, Gavin King, Gary Gregory, *Java Persistence. Programowanie aplikacji bazodanowych w Hibernate.*, Helion, 2016, wydanie 2
[2] Madhusudhan Konda, *Just Spring*, O'Reilly Media, Inc., 2011, wydanie 1
[3] Michael Mikowski, Josh Powell, *Single Page Web Applications: JavaScript End-to-end*, Manning Publications, 2013, wydanie 1
[4] Salahaldin Juba, Andrey Volkov, *Learning PostgreSQL 11*, Packt Publishing, 2019, wydanie 3
[5] Badanie laboratorium CUPS z Carnegie Mellon University dotyczące zachowań użytkowników, https://cups.cs.cmu.edu/soups/2010/proceedings/a2_shay.pdf (dostęp 07.11.2020)
[6] Dyrektywa Parlamentu Europejskiego i Rady (UE) 2015/2366, artykuł 4, punkt 30, https://eur-lex.europa.eu/legal-content/PL/TXT/HTML/?uri=CELEX:32015L2366&from=EN#d1e1384-35-1 (dostęp 02.11.2020)
[7] Lista podatności aplikacji sieciowych OWASP Top Ten, https://owasp.org/www-project-top-ten/ (dostęp 08.11.2020)
[8] Publikacja NIST 800-63B dotycząca wytycznych bezpieczeństwa w sieci, artykuł 5.1, https://pages.nist.gov/800-63-3/sp800-63b.html (dostęp 30.10.2020)