

Politechnika Warszawska

W Y D Z I A Ł E L E K T R Y C Z N Y



Instytut Elektrotechniki Teoretycznej
i Systemów Informacyjno-Pomiarowych
Zakład Elektrotechniki Teoretycznej
i Informatyki Stosowanej

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Inżynieria oprogramowania

Implementacja portalu do grywalizacji z użyciem platformy
SPRING i bazy danych Oracle

Jacek Kozieja
nr albumu 261053

promotor
dr inż. Jacek Korytkowski

WARSZAWA 2017

Implementacja portalu do grywalizacji z użyciem platformy SPRING i bazy danych Oracle

Streszczenie

Praca składa się z krótkiego wstępu opisującego cel oraz strukturę wykonanego projektu, dwóch rozdziałów zawierających opis wykorzystanych technologii (szkielet Spring i baza danych Oracle) i przykłady ich zastosowań w projekcie. Czwarty rozdział to opis zagadnień związanych z bezpieczeństwem współczesnych aplikacji internetowych. Rozdział piąty przedstawia wdrożenie aplikacji do chmury obliczeniowej. Ostatni rozdział to wnioski, dotyczące wydajności i sposobów tworzenia aplikacji.

Słowa kluczowe: Spring, Angular, Oracle

Implementation of gamification web portal based on SPRING framework and Oracle database

Abstract

This thesis consists of a short introduction that describes goal and structure of a project. Next two chapters depict used technologies (Spring framework and Oracle database) and their use in the project. Fourth chapter is a description of security topics related to modern web applications. Fifth chapter shows cloud migration of a system. The last one is a conclusion about overall performance and approach to application development.

Keywords: Spring, Angular, Oracle

WARSZAWA, 16 maja 2017

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa inżynierska pt. Implementacja portalu do grywalizacji z użyciem platformy SPRING i bazy danych Oracle:

- została napisana przeze mnie samodzielnie,
- nie narusza niczyich praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

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

Jacek Kozieja.....

Spis treści

1	Wstęp	1
1.1	Grywalizacja	1
1.2	Projekt	1
1.3	Użyte technologie	5
1.4	Wzorzec MVC	5
2	Angular	8
3	Spring	9
3.1	Metody konfiguracji	9
3.2	Spring Boot	10
3.3	Maven	13
3.4	Przydatne moduły	15
3.5	Implementacja REST API	16
3.5.1	Mapowanie obiektowo - relacyjne	16
3.5.2	Repozytoria	20
3.5.3	Kontroler	21
4	Baza danych Oracle	28
4.1	Integracja z projektem	28
4.2	Optymalizacja	29
5	Bezpieczeństwo	34
5.1	Ataki CSRF	34
5.2	Protokół HTTPS	35
5.3	Logowanie	36
5.4	Reset i zmiana hasła	38
5.5	SQL Injection	39
6	Wdrożenie systemu	40
6.1	Migracja bazy danych	40

6.2 Migracja aplikacji	41
7 Działanie aplikacji	43
8 Wnioski	44
Bibliografia	47

Podziękowania

Dziękuję serdecznie Panu dr. inż. Jackowi Korytkowskiemu za pomoc w przygotowaniu pracy. Dziękuję także moim wspaniałym rodzicom Bożenie i Robertowi Koziejom, dzięki którym miałem możliwość kształcić się i zdobywać cenną wiedzę.

Jacek Kozieja

Rozdział 1

Wstęp

Celem pracy było stworzenie aplikacji internetowej, motywującej grupy ludzi do wspólnej pracy poprzez system wyzwań i nagród. Należy jednak pamiętać że o ile aplikacja pomaga zarządzać zadaniami, prowadzić ranking i kontrolować rozwój użytkowników, o tyle weryfikacja wykonanych zadań czy przyznawanie fizycznych nagród pozostaje obowiązkiem administratora. Stworzone rozwiązanie było także pretekstem do zbadania współczesnych technologii wytwarzania aplikacji internetowych, opisanych w dalszej części pracy.

1.1 Grywalizacja

Grywalizacja, gryfikacja lub gamifikacja, to zgodnie z definicją [6] "Wykorzystanie mechaniki znanej np. z gier fabularnych i komputerowych, do modyfikowania zachowań ludzi w sytuacjach życia codziennego, w celu zwiększenia zaangażowania ludzi." Ta szeroka definicja obejmuje rozwiązania stosowane w marketingu, zarządzaniu projektami jak i edukacji.

1.2 Projekt

Plany stworzenia aplikacji „Gamify” przybliżyć mogą poniższe przykładowe Przypadki Użycia:

UC Użytkownika:

Wykonanie Zadania

1. Użytkownik loguje się do Systemu.
2. Użytkownik wybiera opcję „Wykonaj Zadanie” (Ścieżka alternatywna):
 - (a) Użytkownik skanuje kod QR

- (b) System dodaje zadanie i punkty Użytkownikowi
- 3. System wyświetla formularz wykonania Zadania.
- 4. Użytkownik wpisuje numer Zadania.
- 5. System dodaje zadanie i punkty Użytkownikowi

Przejrzenie tabeli Rankingu

- 1. Użytkownik loguje się do Systemu.
- 2. Użytkownik wybiera opcję „Ranking”.
- 3. System wyświetla tabelę Użytkowników.
- 4. Użytkownik wybiera sortowanie malejąco, po Punktach.
- 5. System wyświetla posortowaną listę.

UC Administratora:

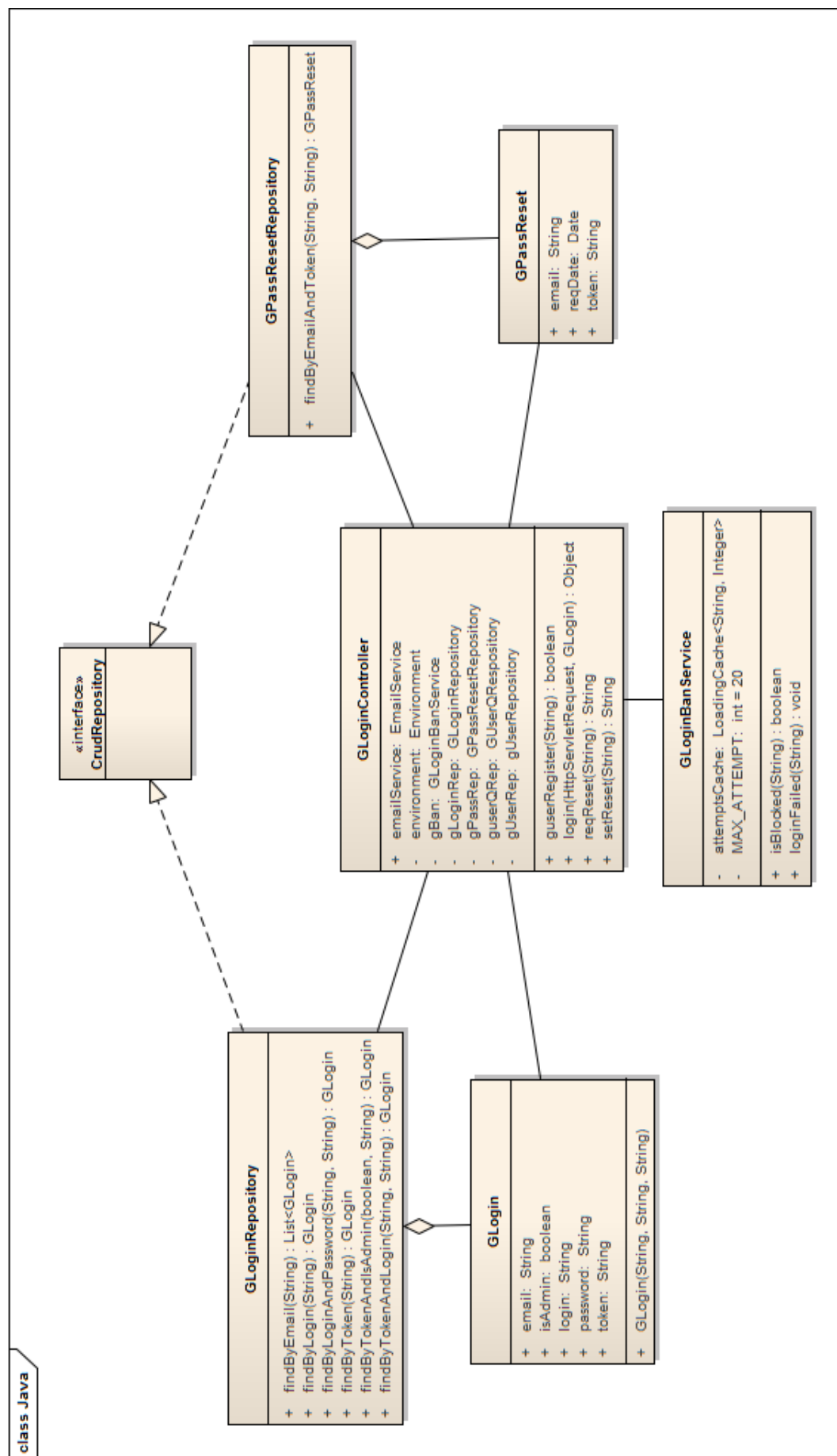
Dodanie nowego Zadania

- 1. Administrator loguje się do systemu
- 2. Administrator wybiera opcję „Dodaj Zadanie”
- 3. System wyświetla formularz dodania Zadania.
- 4. Administrator podaje opis, datę końcową i punkty Zadania.
- 5. System generuje numer Zadania i zapisuje je.

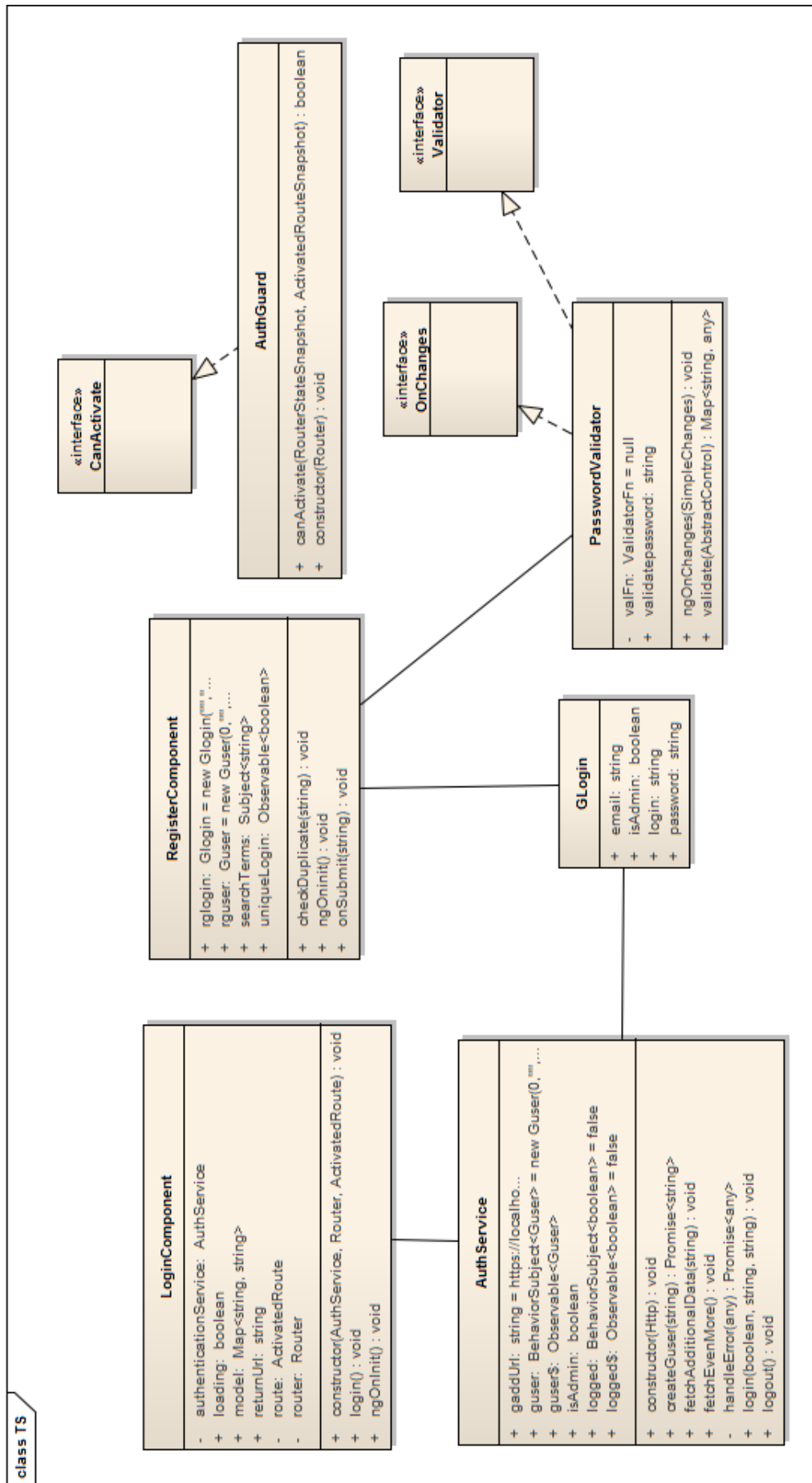
Przysnanie Odznaki Użytkownikom

- 1. Administrator loguje się do Systemu.
- 2. Administrator wybiera opcję „Odznaki”.
- 3. System wyświetla okno edycji Odznak.
- 4. Administrator wybiera opcję „Przysnaj odznakę”
- 5. System wybiera okno wyboru Odznaki.
- 6. Administrator wybiera Odznakę.
- 7. System wyświetla okno wyboru Użytkowników.
- 8. Administrator wybiera Użytkowników.
- 9. System przyznaje odznakę wybranym Użytkownikom.

Diagramy klas na Rysunkach 1.1 i 1.2 opisują strukturę modułu GLogin - oddzielnie dla wnętrza („back-end”) i fasady („front-end”). Są one reprezentatywne dla całości projektu, zaś stworzenie pełnej dokumentacji uważam za odpowiedni temat dla oddzielnej pracy dyplomowej. Prywatne zmienne posiadające publiczne metody dostępne zostały przedstawione jako zmienne publiczne. Redukuje to liczbę metod i zwiększa czytelność diagramów.



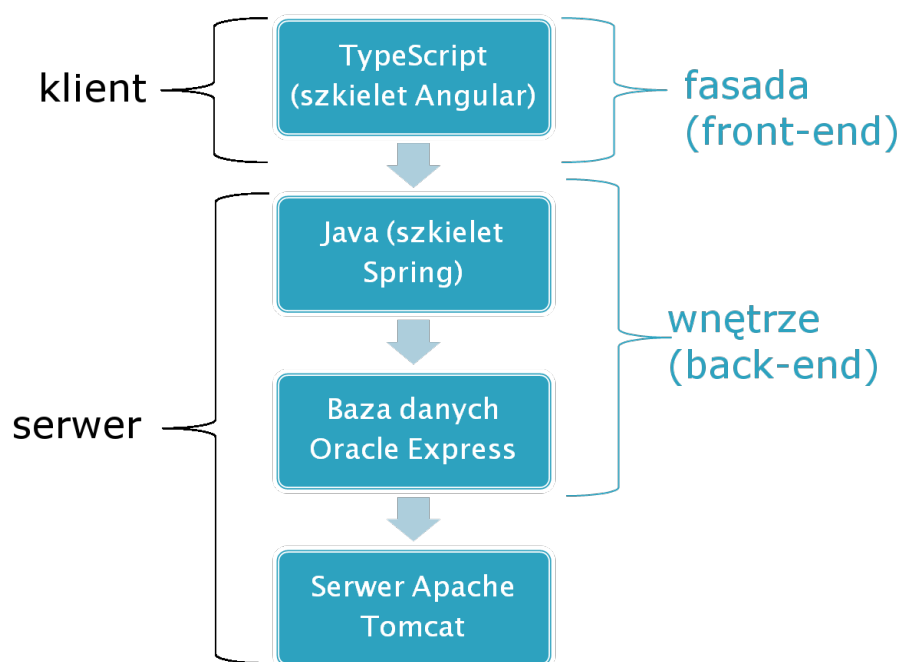
Rysunek 1.1: Diagram klas w języku Java.



Rysunek 1.2: Diagram klas w języku TypeScript.

1.3 Użyte technologie

Do wykonania aplikacji użyto technologii, przedstawionych na Rysunku 1.3. Jak widać, aplikacja została wykonana w architekturze Klient-Serwer,



Rysunek 1.3: Języki, szkielety i systemy będące częścią projektu.

opisanej w [LITERATURA]. Fasada oznacza kod wykonywany przez przeglądarkę internetową po stronie Klienta. Warstwa Klienta obsługuje żądania Użytkownika, wysyła zapytania do Serwera i prezentuje odpowiedzi Użytkownikowi. Wnętrze oznacza kod wykonywany po stronie Serwera. Serwer nasłuchuje żądań Klienta, wykonuje logikę biznesową związaną z danym żądaniem i wysyła odpowiedź do Klienta. Z usług jednego Serwera może korzystać wiele instancji Klientów. Obydwie składowe wykorzystują inne technologie, opisane szerzej w kolejnych rozdziałach.

1.4 Wzorzec MVC

Złożoność współczesnych aplikacji (w tym internetowych), wymusza wykorzystanie odpowiednich wzorców (projektowych, architektonicznych). Narzucają one sprawdzoną formę, która pozwala zapanować nad rozbudowanym kodem. W tworzonej aplikacji, zdecydowano się na wykorzystanie wzorca

MVC. Jest on szeroko opisywany w literaturze, chociażby w [LITERATURA], oraz jest wykorzystywany przez wybrane szkielety aplikacji. MVC to wzorzec architektoniczny służący do organizowania struktury systemów interaktywnych. Składa się on z trzech części:

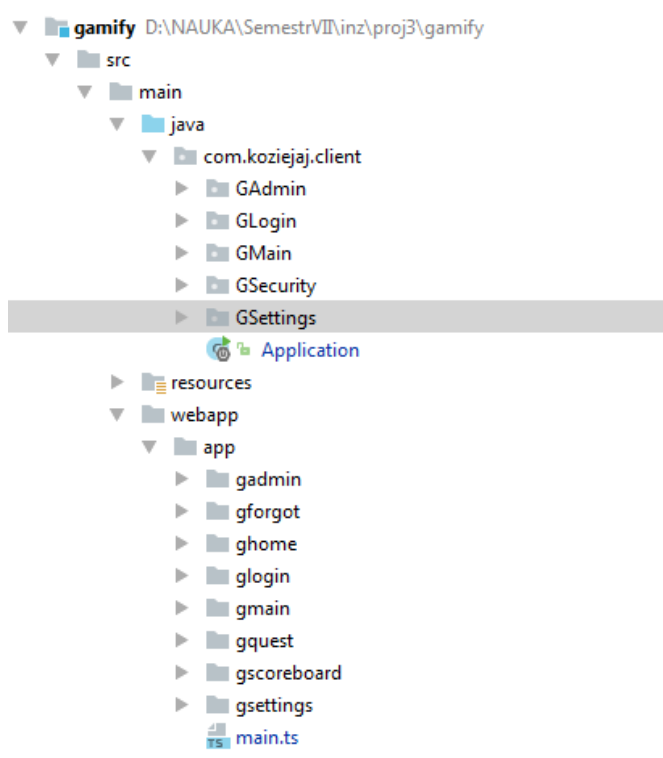
1. Model - reprezentuje dane i wykonuje logikę biznesową.
2. Widok - wyświetla dane pobrane z Modelu.
3. Kontroler - reaguje na dane wejściowe użytkownika. Przesyła żądania wykonania logiki biznesowej do Modelu i zmian Widoku.

Zarówno Angular jak i Spring samodzielnie realizują wzorzec MVC. Jednak gdy połączymy te dwie technologie, sytuacja zmienia się. Warstwa kontrolera przeniesiona jest do Angulara i jest wykonywana przez przeglądarkę. To samo samo dzieje się z warstwą widoku. Po stronie serwerowej Springa pozostaje warstwa modelu. Ten rozdział powoduje że potrzebny jest odpowiedni sposób komunikacji Klient-Serwer. Nazywa się on REST API. Skróty te oznaczają Representational State Transfer i Application Programming Interface. Pierwszy skrót oznacza bezstanową wymianę tekstowych zasobów sieciowych. Przykładem są tu zapytania HTTP GET, POST, PUT, DELETE. Drugi skrót oznacza jednoznacznie zdefiniowany sposób komunikacji między komponentami. W wypadku aplikacji internetowej oznacza to że strona serwerowa odbierając zapytanie HTTP pod danym adresem, powinna zwrócić odpowiedź w postaci XML lub JSON. Przykład zapytania do takiego API i odpowiedzi, znajduje się na Rysunku 1.4. Modułowa struktura projektu jest

```
C:\Users\Jacek>curl -k https://localhost:7000/api/hello
{"id":"ebf90e2d-1ace-4627-aaf1-8f23adba8b22","content":"Hello World"}
C:\Users\Jacek>
```

Rysunek 1.4: Zapytanie do API i odpowiedź. Taka komunikacja pozwala wykorzystać tą samą warstwę modelu w kilku różnych aplikacjach.

przedstawiona na Rysunku 1.5. Przedstawia się ona następująco: wewnątrz projektu Springa rezydują obok siebie 3 foldery - z plikami źródłowymi Javy, plikami statycznymi (grafiką) i projektem Angulara. Obydwa języki posiadają moduły główne - tam przechowywane są konfiguracje i najczęściej używane klasy. Każda większa funkcjonalność posiada oddzielny folder. To samo tyczy się aplikacji admina która jest traktowana jako oddzielny moduł. Przedrostek „g” występujący w nazwach klas i modułów podyktowany jest zbieżnością nazw takich jak login czy user z natywnymi elementami użytych platform.



Rysunek 1.5: Modułowy podział projektu.

Rozdział 2

Angular

AA
Język Typescript został wykorzystany do stworzenia aplikacji klienta - kodu wykonywanego przez przeglądarkę internetową. Język ten jest nadzbiorem języka JavaScript i jest do niego kompilowany. logiki

Rozdział 3

Spring

Podjęto decyzję by warstwę serwerową napisać z pomocą szkieletu Spring. Wybrano go między innymi ze względu na bogatą dokumentację, mnogość przydatnych modułów i dostępność szybkich sposobów na konfigurację sprawnej aplikacji serwerowej. Spring to szkielet tworzenia aplikacji (ang. framework) przeznaczony dla Javy Enterprise Edition, czyli serwerowej odmiany tego języka. Powstał jako alternatywa dla oficjalnych standardów serwerowych Javy takich jak Enterprise JavaBeans. Spring nie narzuca jednego modelu programowania. Posiada wiele modułów które są przydatne przy pisaniu aplikacji, jednocześnie będąc całkowicie opcjonalnymi. Znaną cechą Springa LITERATURA jest „**odwrócenie sterowania**” (ang. Inversion of Control), czyli wzorec w którym wykonywaniem programu steruje szkielet a nie kod programisty. To Spring decyduje kiedy wykonać dane akcje. Przykładem odwróconego sterowania jest chociażby "**wstrzykiwanie zależności**" (ang. Dependency Injection), kiedy to utworzone, zainicjowane obiekty przekazywane są obiektom które ich potrzebują. Obiekty nie tworzą wtedy samodzielnie instancji innych obiektów. Kolejną ważną cechą Springa jest „**programowanie aspektowe**” czyli wzorec w którym rozdziela się fizycznie funkcjonalności i definiuje punkty komunikacji między nimi, dostępne wszędzie tam gdzie są potrzebne.

3.1 Metody konfiguracji

Konfiguracja projektu w Springu może odbywać się na dwa sposoby - przy pomocy plików XML lub adnotacji. Obydwa te podejścia przedstawiono w kursie [2] Poniższy przykład prezentuje te same ustawienia w obydwu reprezentacjach:

XML	Adnotacje
<pre><beans> <bean id = "myClass" class = "com.koziejaj.MyClass" /> </beans></pre>	<pre>@Configuration public class MyClassConfig { @Bean public MyClass myClass(){ return new MyClass(); } }</pre>

Konfiguracja XML odbywa się w oddzielnych plikach, zaś przy pomocy adnotacji - bezpośrednio w kodzie którego dotyczy. Warto wiedzieć że konfiguracja adnotacjami wykonuje się przed XML. W projekcie użyto adnotacji. Wydają się one czytelniejsze, są też kierunkiem w którym zmierza rozwój Springa. Wyjątkiem są pliki z rozszerzeniem „*properties*”. Przechowują one stałe wartości, takie jak numer portu serwera, dane uwierzytelniające bazy danych.

3.2 Spring Boot

Brak narzuconych standardów szkieletu aplikacji ma także i swoje wady. Spring wymaga dużej ilości konfiguracji, a także dobrania modułów i bibliotek które będą nam potrzebne. Moduł Spring Boot zawiera wstępną konfigurację i najpotrzebniejsze biblioteki. Jednocześnie umożliwia dostosowanie projektu do specyficznych wymagań. Podczas uruchomienia sprawdza których elementów konfiguracji brakuje i dodaje je zgodnie z posiadaną wiedzą na temat projektu. Moduł ten można uznać za fundament stworzonej aplikacji. Ilość kodu wymaganą do napisania aplikacji serwerowej w ramach Spring Boot przedstawiają poniższe listingi, zgodne z podejściem przedstawionym w [1]:

```
package hello;
```

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Witaj Swiecie! Pozdrowienia od Spring Boot!\n";
    }
}
```

```
}
```

Adnotacja „*@RestController*” informuje moduł Spring MVC że klasa będzie przystosowana do usługi zapytań sieciowych. Adnotacja „*@RequestMapping("/")*” oznacza ścieżkę która spowoduje wywołanie metody.

```
package hello;
```

```
import java.util.Arrays;
```

```
import org.springframework.boot.CommandLineRunner;
```

```
import org.springframework.boot.SpringApplication;
```

```
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.annotation.Bean;
```

```
@SpringBootApplication
```

```
public class Application {
```

```
    public static void main(String[] args) {
```

```
        SpringApplication.run(Application.class, args);
```

```
    }
```

```
    @Bean
```

```
    public CommandLineRunner commandLineRunner(ApplicationContext  
        ctx) {
```

```
        return args -> {
```

```
            System.out.println("Klasy Projektu:");
```

```
            String[] beanNames = ctx.getBeanDefinitionNames();
```

```
            Arrays.sort(beanNames);
```

```
            for (String beanName : beanNames) {
```

```
                System.out.println(beanName);
```

```
            }
```

```
        };
```

```
    }
```

}

Adnotacja „*@SpringBootApplication*” jest wygodnym połączeniem kilku innych adnotacji:

- „*@Configuration*” - oznacza klasę jako źródło metod z adnotacją *@Bean*, które powinny być dostępne w kontekście aplikacji.
- „*@EnableAutoConfiguration*” - nakazuje Spring Boot dodać do kontekstu aplikacji wszystkie metody „*@Bean*” na podstawie ścieżki projektu, innych metod „*@Bean*” i dostępnych konfiguracji.
- „*@ComponentScan*” - skanuje pakiet (w tym przypadku „hello”) w poszukiwaniu komponentów, konfiguracji i serwisów.

Metoda „*SpringApplication.run()*;” uruchamia aplikację, zaś metoda zwracająca „*CommandLineRunner*” jest uruchamiana podczas startu. Rysunek 2.1 przedstawia uruchomioną aplikację, zaś Rysunek 2.2 - obsłużenie zapytania.

```
dServletContainer : Tomcat started on port(s): 8080 (http)
2017-05-19 14:19:48.415 INFO 6348 --- [           main] hello.Application
: Started Application in 6.988 seconds (JVM running for 7.689)

Klasy Projektu:
application
basicErrorController
beanNameHandlerMapping
beanNameViewResolver
characterEncodingFilter
conventionErrorViewResolver
defaultServletHandlerMapping
defaultValidator
defaultViewResolver
dispatcherServlet
dispatcherServletRegistration
duplicateServerPropertiesDetector
embeddedServletContainerCustomizerBeanPostProcessor
error
```

Rysunek 3.1: Fragment uruchomienia skompilowanego przykładu.

```
PS D:\NAUKA\SemestrVII\inz\spring-boot-quickstart\gs-spring-boot\initial> curl 1
ocalhost:8080
Witaj Swiecie! Pozdrowienia od Spring Boot!
PS D:\NAUKA\SemestrVII\inz\spring-boot-quickstart\gs-spring-boot\initial>
```

Rysunek 3.2: Zapytanie wysłane do przykładu.

3.3 Maven

Maven to w narzędzie do zarządzania projektem - jego zależnościami (używanymi bibliotekami) i strukturą (podziałem na moduły). Maven automatyzuje budowę projektu zgodnie z poniższymi krokami [3].

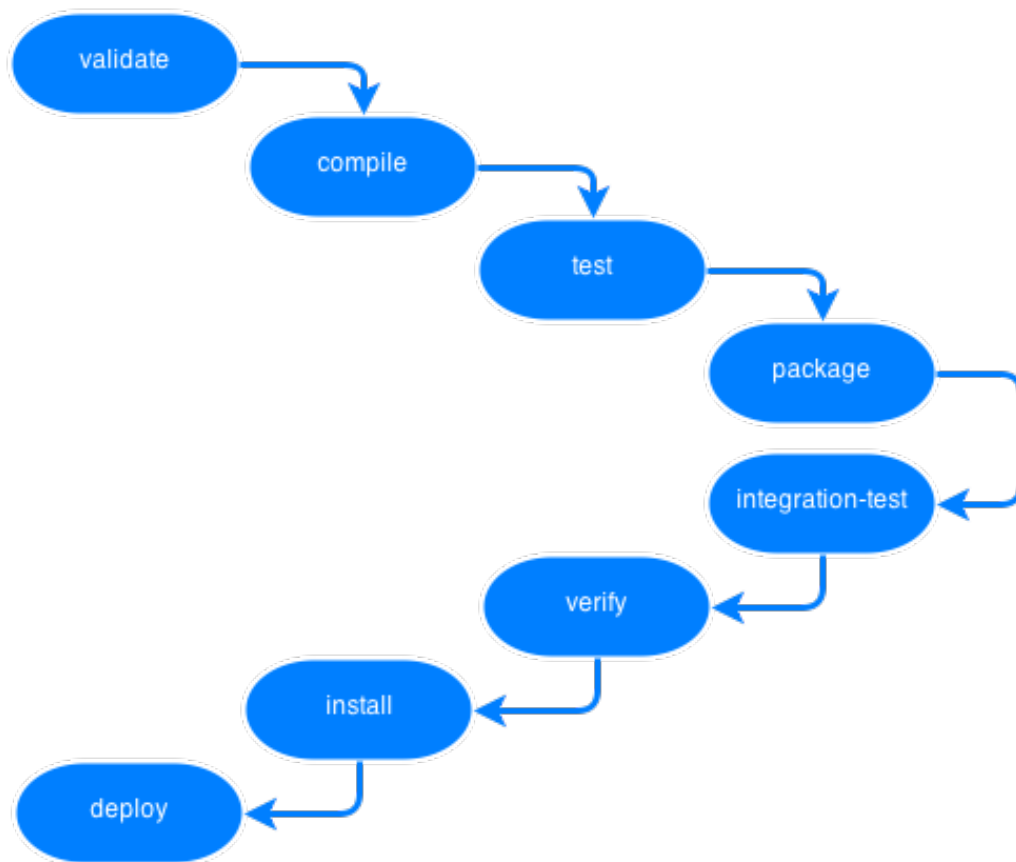
1. Validate - walidacja kodu. Sprawdzana jest poprawność kodu źródłowego, umożliwiającą jego kompilację.
2. Compile - kompilacja, czyli przetworzenie kodu źródłowego na kod bajtowy, wykonywany przez maszynę wirtualną Javy.
3. Test - wykonanie testów jednostkowych.
4. Package - budowa pojedynczej paczki wykonywalnej (plik .JAR) lub wykonywanej przez serwer (plik .WAR).
5. Integration-test - wykonanie testów integracyjnych.
6. Verify - weryfikacja jakościowa paczki zbudowanej w kroku 4.
7. Install - instalacja paczki. Oznacza to umieszczenie paczki z kroku 4 w repozytorium lokalnym lub zdalnym.
8. Deploy - umieszczenie projektu w repozytorium.

Konfiguracja Maven znajduje się w pliku POM.xml. Poniższy przykład przedstawia skróconą wersję pliku POM stworzonego projektu:

```
<!-- Nagłówek określający używaną wersję -->
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
: //www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/maven-v4_0_0.xsd">

<!-- Podstawowe informacje na temat projektu -->
<modelVersion>4.0.0</modelVersion>
<groupId>com.koziejaj.client</groupId>
<artifactId>gamify</artifactId>
<packaging>jar</packaging>
<version>0.1-dev</version>
<name>gamify</name>
<url>http://maven.apache.org</url>

<!-- Dziedziczenie po istniejącym projekcie -->
<parent>
```



Rysunek 3.3: Uproszczony cykl życia projektu Maven.

```

<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.4.2.RELEASE</version>
</parent>

<!-- Zależności – lista bibliotek które należy pobrać -->
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>
</dependencies>

<!-- Dodatkowe ustawienia -->
<properties>
<java.version>1.8</java.version>
</properties>

</project>

```

3.4 Przydatne moduły

Oto lista najważniejszych modułów użytych w projekcie:

- Spring MVC - przy pomocy klasy „*DispatcherServlet*” przekazuje zapytania do odpowiednich kontrolerów, tak jak na pierwszym fragmencie kodu z Podrozdziału 3.2 Spring Boot. W przypadku wykonanego projektu, jest częścią modułu spring-boot-starter-web.
- Spring Data - rozbudowana, wielomodułowa biblioteka, wspierająca warstwę dostępu do danych. Wypiera popularną niegdyś bibliotekę Hibernate. W przypadku Gamify, użyto dwóch modułów:
 - JPA - służy do mapowania obiektowo-relacyjnego zgodnie ze standardem JPA. Pobrany jako moduł spring-boot-starter-data-jpa.

- REST - Umożliwia tworzenie repozytoriów z automatycznie generowanymi metodami CRUD. Obiekt takiego repozytorium odpowiada tabeli w bazie danych. Mogą być one z łatwością rozbudowane o szczegółowe metody obróbki danych. Przykładowo, metoda

```
GLogin findByLoginAndPassword(String login, String password);
```

wyszuka użytkowników o określonym loginie i hasle bez potrzeby pisania jakiegokolwiek logiki biznesowej. REST umożliwia wysyłanie zapytań do repozytoriów w sposób analogiczny do kontrolerów 3.2, jednak ta funkcjonalność nie została wykorzystana w projekcie. Moduł pobrany jako: spring-boot-starter-data-rest.

- Spring Security - biblioteka zapewniająca bezpieczeństwo aplikacji, szerzej opisana w rozdziale Bezpieczeństwo 5.

3.5 Implementacja REST API

Implementacja serwerowej części aplikacji zostanie omówiona na przykładzie modułu panelu Administratora. Ograniczy to analizę do sześciu klas, zorganizowanych podobnie do wcześniejszego Rysunku 1.1.

3.5.1 Mapowanie obiektowo - relacyjne

Klasy omówione w tej sekcji są definicjami tabel bazy danych. Zostały one wytworzone zgodnie z poradnikiem „Accessing Data JPA” [1]. Klasa taka odpowiada pojedynczemu rekordowi tabeli i pozwala na jego modyfikację. Jedną z funkcjonalności panelu Administratora jest tworzenie i przyznawanie Odznak. Jest to taki wirtualny medal za specjalne zasługi, który Administrator może nadać Użytkownikowi. Instancja takiego medalu musi zawierać dwie informacje - unikatowy login Użytkownika który go posiada i nazwę Odznaki. Nazwa ta odpowiada plikowi graficznemu Odznaki w plikach statycznych projektu.

```
package com.koziejaj.client.GAdmin;
```



```

import ...
//Adnotacja oznaczająca klasę odpowiadającą tabeli bazy danych
@Entity
//Adnotacja wczytująca definicję klucza głównego (primary key) – wyjaś
    nienie w kolejnym listingu
@IdClass(GBId.class)
public class GBadge{

    //Adnotacje klucza głównego
    @Id
    private String login;
    @Id
    private String badge;

    //Konstruktory
    public GBadge() {
    }

    public GBadge(String i, String v) {
        login = i;
        badge = v;
    }

    //Metody dostępne
    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getBadge() {
        return badge;
    }

    public void setBadge(String badge) {
        this.badge = badge;
    }
}

```

Chcąc wykorzystywać klucz główny złożony z kilku kolumn, należy zdefiniować specjalną klasę. Taki klucz gwarantuje że dany Użytkownik posiada co najwyżej jedną instancję danej odznaki.

```
package com.koziejaj.client.GAdmin;

import java.io.Serializable ;

//klasa klucza głównego implementując serializację, co pozwala szkieletowi
//aplikacji porównywać ich wartości zapisane jako strumień bajtów.
public class GBId implements Serializable {

    private String login;
    private String badge;

    public GBId(){}

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getBadge() {
        return badge;
    }

    public void setBadge(String badge) {
        this.badge = badge;
    }
}
```

Administrator może zmieniać różne parametry wpływające na aplikację Użytkownika. Jest to funkcjonalność bardzo prostego „Systemu zarządzania treścią” (ang. CMS). Na ten moment obsługiwane wartości to: zawartość strony powitalnej, słowa oznaczające Punkty i Poziomy w aplikacji Użytkownika, arkusz css aplikacji, wzór na progi punktowe kolejnych Poziomów. Dodanie

kolejnej funkcjonalności tego typu wymagałoby dodania rekordu w tabeli i zmodyfikowania aplikacji Użytkownika. Do przechowywania zmian potrzebna jest bardzo prosta klasa:

```
package com.koziejaj.client.GAdmin;

import ...
@Entity
public class GLayout {

    @Id
    private String id;
    private String value;

    public GLayout() {}

    public GLayout(String i, String v) {
        id = i;
        value = v;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    //Metoda ToString przesłonięta by ułatwić debugowanie aplikacji.
    @Override
    public String toString() {
```

```

    return String.format(
        "GLayout[id=%s, value=%s]",
        id, value);
    }
}

```

3.5.2 Repozytoria

Zarówno klasa „*GBadge*” jak „*GLayout*” mają swoje repozytoria. Można traktować je jako instancje tabel bazy danych. To w repozytoriach szuka się rekordów i zapisuje zmiany na stałe, tak jak zostało to opisane w Literaturze [1]. Poniżej znajduje się kod repozytoriów wraz z komentarzami.

```

package com.koziejaj.client.GAdmin;

import ...

//Adnotacja oznaczająca repozytorium REST i jego ścieżkę dostępu.
//Funkcjonalność nie używana w projekcie.
@RepositoryRestResource(collectionResourceRel = "GBadges", path = "
    GBadges")

//implementując interfejs CrudRepository otrzymujemy zestaw metod
//takich jak save, findAll, findOne, count, delete i możemy rozszerzać
//klasę przy pomocy słów findBy, removeBy, countBy, Top, First i tym
//podobnych. Należy podać parametry<T, ID> oznaczające typ
//przechowywanych danych i typ klucza głównego
public interface GBadgeRepository extends CrudRepository<GBadge,
    String> {

    //Metoda zwróci wszystkie Odznaki podanego Użytkownika
    List<GBadge> findByLogin( String login);
    //Adnotacja oznaczająca wykonanie metody (usunięcia wszystkich Odznak
    //danego typu) w pojedynczej transakcji bazy danych. Jest to wymóg
    //formalny, zabezpieczający przed próbą operowania na usuniętych już
    //rekordach.
    @Transactional
    List<GBadge> removeByBadge(@Param("badge") String badge);

```

```
}
```

```
package com.koziejaj.client.GAdmin;  
  
import ...  
  
@RepositoryRestResource(collectionResourceRel = "GUsers", path = "  
    GUsers")  
public interface GLayoutRepository extends CrudRepository<GLayout,  
    String> {  
}  

```

3.5.3 Kontroler

Kontroler to klasa nasłuchująca zapytań pod zdefiniowanymi adresami URL i wykonująca logikę biznesową operując na wyżej opisanych klasach mapowań i repozytoriów. Podstawowa budowa została zaprezentowana w Podrozdziale 3.2

```
package com.koziejaj.client.GAdmin;  
  
import ...  
  
@RestController  
public class GAdminController {  
  
    //Adnotacja wstrzykująca obiekty zgodnie z wzorcem Dependency  
    Injection. Jak widać, moduł panelu Administratora wymaga użycia  
    klas z innych modułów.  
    @Autowired  
    private GQuestRepository gQuestRep;  
    @Autowired  
    private GUserRepository gUserRep;  
    @Autowired  
    private GLoginRepository gLoginRep;  
    @Autowired  
    private GUserQRepository guserQRep;
```

```

@Autowired
private GLayoutRepository gLayoutRep;
@Autowired
private GBadgeRepository gBadRep;

//Poza ścieżką możemy zdefiniować typ zapytania HTTP na który
//reaguje metoda.
@RequestMapping(value="/api/gadmin/newquest", method =
    RequestMethod.POST)
//@RequestBody definiuje zmienną odbierającą ciało zapytania – przyk
//ładowo plik JSON. @RequestParam to parametr zapytania, zgodny
//ze wzorcem: URL?parametr=wartość
public Long newQuest(@RequestBody String postData,
    @RequestParam(value="token") String token) throws
    IOException {
    if(gLoginRep.findByTokenAndIsAdmin(token,true) != null) {
        //Klasa ObjectMapper z biblioteki FasterXML umożliwia
        //czytanie plików JSON i zamianę obiektów na odpowiedzi
        //zgodne z tym formatem.
        HashMap<String, Object> result = new ObjectMapper().
            readValue(postData, HashMap.class);

        //ID nowego Zadania to unikatowa, dwunastocyfrowa losowa
        //liczba.
        Long myId = ThreadLocalRandom.current().nextLong
            (1000000000000L, 10000000000000L);
        while (gQuestRep.findById(myId) != null) {
            myId = ThreadLocalRandom.current().nextLong
                (1000000000000L, 10000000000000L);
        }
        //Każde Zadanie ma Datę po której nie może zostać wykonane
        //Jeśli Zadanie dostępne jest zawsze, wystarczy podać
        //bardzo dużą wartość np rok 9999.
        LocalDateTime myDate = LocalDateTime.parse(result.get("
            endOf").toString());
        //Nowe Zadanie zapisywane jest poprzez repozytorium a
        //Administrator otrzymuje ID nowego zadania jako odpowied
        //ź zwrótną.
        gQuestRep.save(new GQuest(myId, result.get("description").
            toString(), (int) result.get("exp"), myDate));
        return myId;
    }
}

```

```

    }
    else
        return null;
}

//Administrator ma podgląd wszystkich Zadań istniejących w aplikacji.
@RequestMapping("/api/gadmin/allquests")
public Iterable<GQuest> gusers(@RequestParam(value="token")
    String token) {
    if(gLoginRep.findByTokenAndIsAdmin(token,true) != null) {
        Iterable<GQuest> model = gQuestRep.findAll();
        return model;
    }
    else
        return null;
}

//Administrator może usunąć konto Użytkownika
@RequestMapping("/api/gadmin/rmguser")
public boolean rmGuser(@RequestParam(value="id") Long id,
    @RequestParam(value="token") String token) {
    if(gLoginRep.findByTokenAndIsAdmin(token, true) != null) {
        String login = gUserRep.findOne(id).getLogin();
        gUserRep.delete(id);
        gLoginRep.delete(login);
        guserQRep.removeByGuserId(id);
        return true;
    }
    else
        return false;
}

//Administrator i Użytkownik mogą pobrać parametry edytowalne
//aplikacji Użytkownika – stronę powitalną, wzór na punkty i tym
//podobne.
@RequestMapping("/api/gadmin/getparams")
public Iterable<GLayout> getParam(@RequestParam(value="token"
    ) String token) {
    if(gLoginRep.findByToken(token) != null) {
        Iterable<GLayout> model = gLayoutRep.findAll();

```

```

        return model;
    }
    else
        return null;
}

//Administrator by móc edytować plik CSS z poziomu aplikacji,
//otrzymuje wersję skonwertowaną na JSON, stąd zamiana znaków
//specjalnych na kody HTML.
@RequestMapping("/api/gadmin/getcss")
public String getCss(@RequestParam(value="token") String token)
throws IOException {
    if(gLoginRep.findByToken(token) != null) {
        byte[] encoded = Files.readAllBytes(Paths.get("src/main/
            webapp/styles.css"));
        String model = new String(encoded, Charset.defaultCharset()
            );
        model = model.replace("\r\n", "%D%A");
        model = model.replace("\"", "%22");
        model = model.replace("\\", "%5C");
        model = model.replace("{", "%7B");
        model = model.replace("}", "%7D");
        model = "\"" + model + "\"";
        return model;
    }
    else
        return null;
}

//Administrator może zmienić wartości parametrów edytowalnych
//aplikacji Użytkownika.
@RequestMapping("/api/gadmin/setparams")
public String setParams(@RequestBody ArrayList<GLayout>
    postData, @RequestParam(value="token") String token) throws
    IOException {
    if(gLoginRep.findByTokenAndIsAdmin(token, true) != null) {
        gLayoutRep.save(postData);
        return new ObjectMapper().writeValueAsString("Zmiany
            zostały zapisane");
    }
    else
        return null;
}

```



```

}
//Gdy Administrator chce nadpisać CSS, kody HTML są zamieniane z
    powrotem na znaki a całość jest zapisywana w pliku.
@RequestMapping("/api/gadmin/setcss")
public String setCss(@RequestBody String model, @RequestParam(
    value="token") String token) throws IOException {
    if(gLoginRep.findByTokenAndIsAdmin(token, true) != null) {
        model = model.substring(1, model.length() - 1);
        model = model.replace("%D%A", "\r\n");
        model = model.replace("%22", "\"");
        model = model.replace("%5C", "\\");
        model = model.replace("%7B", "{");
        model = model.replace("%7D", "}");
        PrintWriter out = new PrintWriter("src/main/webapp/styles.
            css");
        out.print(model);
        out.close();
        return new ObjectMapper().writeValueAsString("Zmiany
            zostały zapisane");
    }
    else
        return null;
}
//Administrator ma podgląd wszystkich Odznak które może przyznać.
    Zwracane są tu nazwy plików graficznych które odpowiadają
    zapisom w tabeli. Odznaki są widoczne dzięki uniwersalnej ścieżce
    API zwracającej obrazki, znajdujące się w innym module.
@RequestMapping("/api/gadmin/allbadges")
public List<String> allBadges(@RequestParam(value="token")
    String token) {
    if(gLoginRep.findByToken(token) != null) {
        File folder = new File("target/classes/static/images/badges"
            );
        File[] listOfFiles = folder.listFiles();
        List<String> model = new ArrayList<String>();
        for (int i = 0; i < listOfFiles.length; i++) {
            if (listOfFiles[i].isFile())
                model.add(listOfFiles[i].getName());
        }
        return model;
    }
}

```

```

        else
            return null;
    }

    //Administrator może usunąć Odznakę. Jest ona odbierana wszystkim
    //Użytkownikom.
    @RequestMapping("/api/gadmin/rmbadge")
    public boolean rmBadge(@RequestParam(value="id") String id,
        @RequestParam(value="token") String token) {
        if(gLoginRep.findByTokenAndIsAdmin(token, true) != null) {
            File file = new File("target/classes/static/images/badges/"
                + id);
            file.delete();
            gBadRep.removeByBadge(id);
            return true;
        }
        else
            return false;
    }

    //Administrator może przyznać Odznakę liście wybranych Użytkowników
    //w.
    @RequestMapping("/api/gadmin/givebadge")
    public String gvBadge(@RequestParam(value="badge") String badge,
        @RequestBody String postData, @RequestParam(value="token")
        String token) throws IOException {
        if(gLoginRep.findByTokenAndIsAdmin(token, true) != null) {
            HashMap<String, Boolean> result = new ObjectMapper().
                readValue(postData, HashMap.class);
            for (Map.Entry<String, Boolean> entry : result.entrySet()) {
                String key = entry.getKey();
                boolean value = entry.getValue();
                if (value == true)
                    gBadRep.save(new GBadge(key, badge));
            }
            return new ObjectMapper().writeValueAsString("Odznaki
                zostały przyznane");
        }
        else
            return null;
    }
}

```

Rozdział 4

Baza danych Oracle

Dane zapisywane przez serwer muszą być przechowywane w sposób uporządkowany, umożliwiając łatwy dostęp do żądanych informacji i wydajny nawet przy dużej ilości danych. Wymagania te idealnie spełnia baza danych Oracle. Posiada ona rozległe możliwości optymalizacji, a dane w tabelach mogą być przeglądane w sposób przypominający arkusz kalkulacyjny, co jest przydatne podczas testów. Pierwszy system zarządzania bazą danych Oracle powstał w 1979 roku i do dziś rozwiązania tej firmy pozostają w czołówce najpopularniejszych baz danych na świecie. Jest to oprogramowanie zamknięte, w dużej mierze płatne. Istnieje na szczęście bezpłatna wersja Express, która została wykorzystana w projekcie.

4.1 Integracja z projektem

Konfiguracja połączenia z bazą danych Oracle wymaga od nas dwóch kroków. Najpierw dodajemy sterownik do projektu z pomocą Maven i POM.xml:

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc7</artifactId>
  <version>12.1.0.1</version>
</dependency>
```

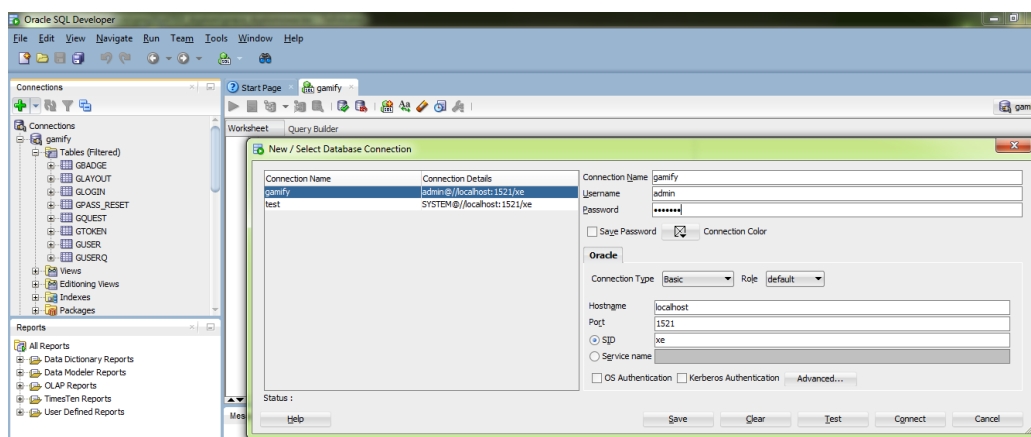
Następnie w pliku application.properties dodajemy poniższe linijki.

```
spring.datasource.url=jdbc:oracle:thin:@//localhost:1521/xespring.datasource.username=admin
```

```
spring.datasource.password=test123
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
```

```
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.hibernate.ddl-auto=update
```

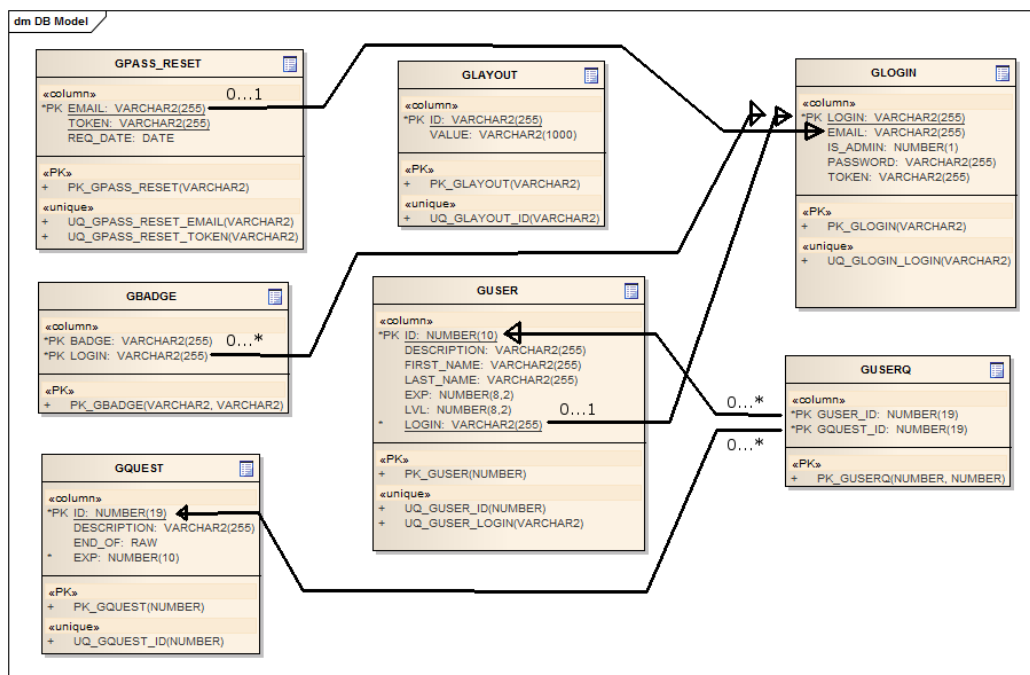
Informacje takie jak numer portu login, hasło są podawane podczas tworzenia nowej bazy danych w GUI narzędzia Oracle SQL Developer, przedstawionym na Rysunku 4.1. Alternatywnie można wykorzystać SQL*Plus, czyli program linii komend. Rysunek 4.2 przedstawia schemat bazy danych projektu.



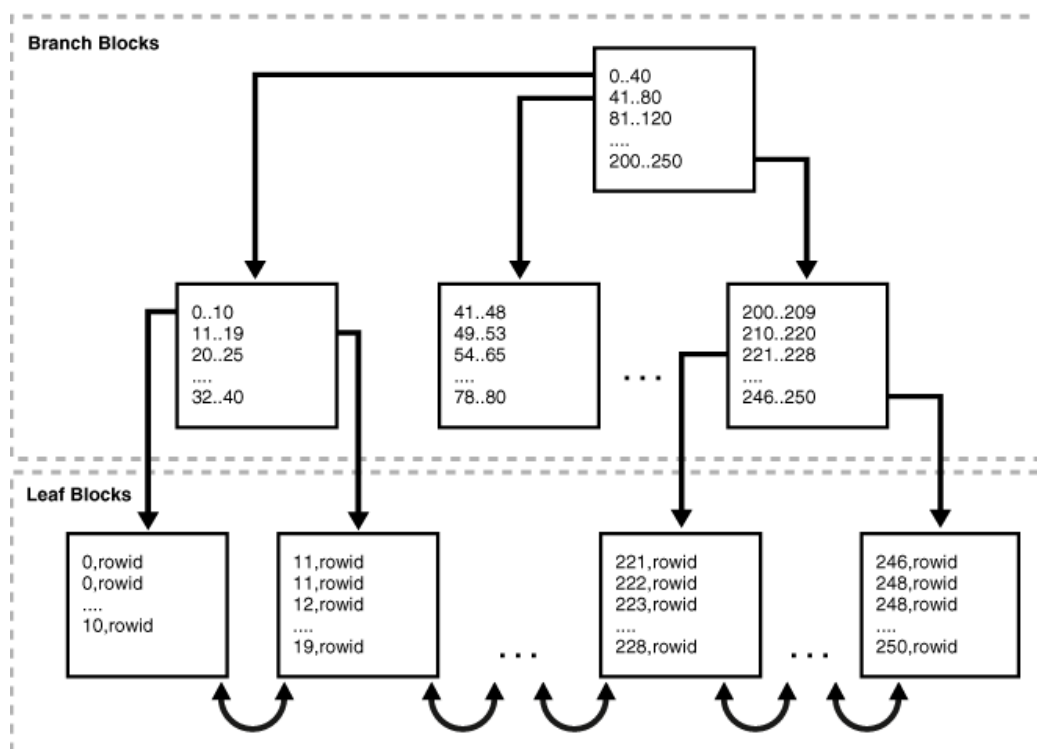
Rysunek 4.1: SQL Developer pozwala na kompleksowe zarządzanie bazą danych

4.2 Optymalizacja

Mechanizmami optymalizującymi zapytania do bazy danych jest między innymi **indeksowanie** i **partycjonowanie**. Indeksowanie tabeli po danej kolumnie/kolumnach powoduje przechowywanie wszystkich rekordów w oddzielnej strukturze w sposób posortowany. Standardowe indeksy Oracle opierają się na B-drzewach, których przykład został pokazany na Rysunku 4.3, zaczerpniętym z oficjalnej dokumentacji [5]. Standardowo, indeks tabeli tworzony jest po kluczu głównym. Można dodać je też samodzielnie. Poniższa komenda doda do tabeli „*GUSER*” indeksowanie po kolumnie „*LOGIN*”, która jest wyszukiwana równie często co klucz główny „*ID*”.



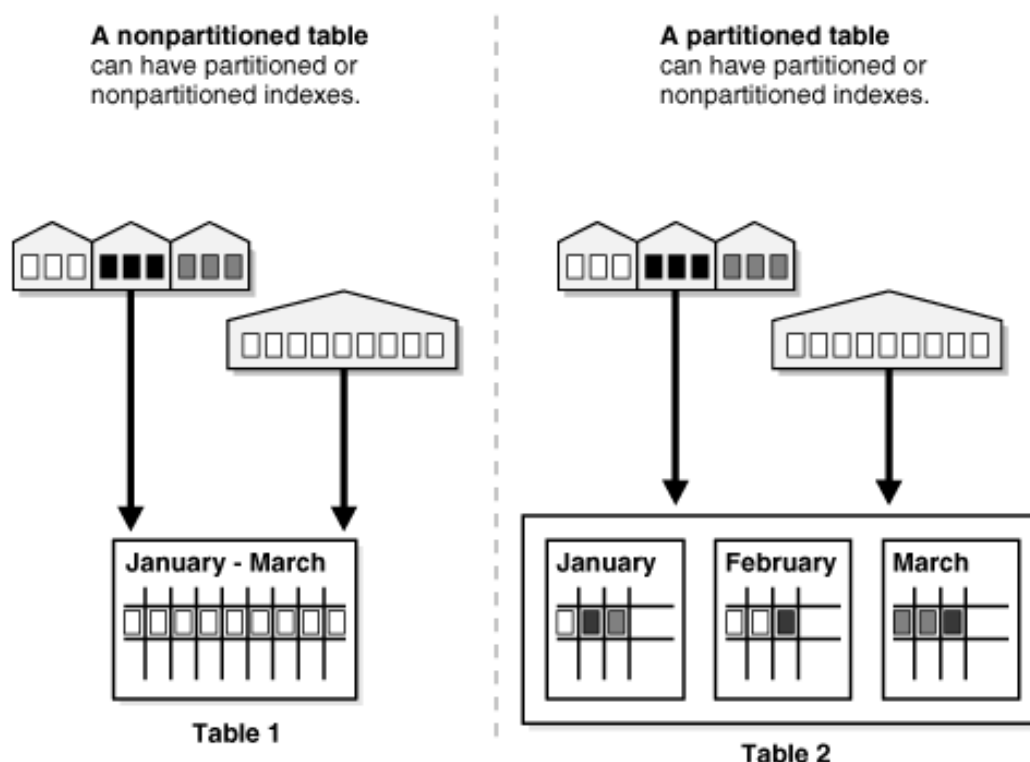
Rysunek 4.2: PK oznacza klucz główny. Strzałki to klucze obce, kierunek grotu wskazuje na tabelę z której pochodzi klucz obcy. Oznaczenia liczbowe wskazują ile takich samych kluczy obcych może być w tabeli.



Rysunek 4.3: Liście B-drzewa przechowują rekordy, gałęzie usprawniają przeszukiwanie.

```
CREATE INDEX GUSER_LOGIN ON GUSER (LOGIN);
```

Partycjonowanie dzieli z kolei tabelę na części na podstawie wartości kolumny. Idealnym przykładem jest tu trzymanie historycznych danych dzieląc na partycje miesiące czy lat. Zgodnie z dokumentacją, należy rozważyć partycjonowanie każdej tabeli przekraczającej 2 GB wielkości. Różnicę dobrze obrazuje rysunek 4.4 [5]: Chcąc partycjonować tabelę „*GQUEST*” po roku zakończe-



Rysunek 4.4: Mechanizmy partycjonowania i indeksowania można ze sobą łączyć.

nia zadania, należy wykonać poniższą komendę. Niestety po jej wykonaniu okazało się że wersja Oracle Express nie obsługuje funkcji partycjonowania.

```
--Tabela tymczasowa -- taka sama jak GUSER
CREATE TABLE TEMP
( ID NUMBER(19,0),
  DESCRIPTION VARCHAR(255),
```



```

        END_OF TIMESTAMP(6),
        EXP NUMBER(10,0)
    )
--Partycjonowanie po dacie
PARTITION BY RANGE (END_OF)
INTERVAL(NUMTOYMINTERVAL(1, 'YEAR'))
( PARTITION y0 VALUES LESS THAN (TO_TIMESTAMP('
    2016/01/01 00:00:00', 'YYYY/MM/DD HH:MI:SS')),
  PARTITION y1 VALUES LESS THAN (TO_TIMESTAMP('
    2017/01/01 00:00:00', 'YYYY/MM/DD HH:MI:SS')),
  PARTITION y2 VALUES LESS THAN (TO_TIMESTAMP('
    2018/01/01 00:00:00', 'YYYY/MM/DD HH:MI:SS')),
  PARTITION y3 VALUES LESS THAN (TO_TIMESTAMP('
    2019/01/01 00:00:00', 'YYYY/MM/DD HH:MI:SS')) );
--Zmiana definicji oryginalnej tablicy
exec dbms_redefinition.start_redef_table( user, 'GQUEST', 'TEMP' );

```

Rozdział 5

Bezpieczeństwo

Każda aplikacja operująca w sieci komputerowej, jest narażona na ataki hakerów. Często spotykane wiadomości o udanych atakach na znane systemy informatyczne potwierdzają, że nie jest to zagadnienie trywialne. Dbanie o zabezpieczenia jest obowiązkiem każdego programisty. Na szali leżą tu poufne dane, a w przypadku niektórych aplikacji - mienie i bezpieczeństwo ludzi.

5.1 Ataki CSRF

Ataki typu Cross-Site Request Forgery [7] polegają na oszukaniu Użytkownika celem wykonania przez niego niezamierzonych akcji w aplikacji od której jest zalogowany. Oszust może nakłonić Użytkownika do kliknięcia w link prowadzący do innej strony, której budowa spowoduje zmianę lub usunięcie danych Użytkownika. Oszust wykorzystuje tu cudze uprawnienia w aplikacji. Popularną metodą zabezpieczenia się przed tego typu atakiem jest wysłanie losowego tokena (unikatowego dla Użytkownika i sesji) i zapisanie go w pliku Cookie. Token ten jest wysyłany z każdym zapytaniem z powrotem do serwera i zmieniany po każdym zamknięciu przeglądarki. Unie możliwia to atak, ponieważ strona oszusta nie ma dostępu do pliku Cookie oryginalnej aplikacji. Ta metoda, wykorzystana w projekcie, nie wymagała żadnych zmian po stronie projektu Angular. Pobiera on i wysyła tokeny pod warunkiem że otrzyma plik Cookie o nazwie "XSRF-TOKEN", wysłany z nagłówkiem "X-XSRF-TOKEN". Po stronie Springa, należało dodać poniższy kod.

```
package com.koziejaj.client.GSecurity;
```

```

import ...

//Poniższa klasa jest oznaczona jako konfiguracja modułu Spring Security
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true, jsr250Enabled =
    true)
public class SecurityConfiguration extends
    WebSecurityConfigurerAdapter {

    //Dodanie obiektu typu HttpSecurity jako argumentu wywołania, umożliwiającego jego modyfikację.
    @Override
    protected void configure(HttpSecurity http) throws Exception {

        // Standardowo, biblioteka Security ma włączoną obsługę
        // zabezpieczenia CSRF. Jedyne co należy zrobić to ustawić sposób
        // wysyłania tokena do pliku Cookie. Domyślnie token wysyłany jest jako
        // atrybut zapytań.
        http.csrf().csrfTokenRepository(CookieCsrfTokenRepository.
            withHttpOnlyFalse());

    }
}

```

5.2 Protokół HTTPS

Protokół HTTPS to szyfrowana wersja protokołu HTTP. Jego wykorzystanie utrudnia wykradnięcie wrażliwych danych jak na przykład hasło użytkownika wysyłane przy logowaniu. Wykorzystanie HTTPS wymaga wygenerowania certyfikatu SSL [7] który zawiera klucz publiczny i różne informacje o serwerze. Na potrzeby projektu certyfikat został wygenerowany i podpisany samodzielnie, co w przeglądarkach powoduje wyświetlenie ostrzeżenia. W środowisku produkcyjnym, certyfikat powinien być podpisany przez Urząd Certyfikacyjny. Posiadając certyfikat, należało w pliku `application.properties` dodać poniższe linie:

```
server.ssl.key-store: keystore.koziejaj #Nazwa pliku
server.ssl.key-store-password: Qazsedc #Hasło
server.ssl.keyStoreType: PKCS12 #Format pliku
server.ssl.keyAlias: tomcat #Nazwa klucza publicznego wewnątrz pliku
```

5.3 Logowanie

Bezpieczne logowanie zostało zaimplementowane w następujący sposób:

1. Użytkownik wysyła login i hasło do serwera
2. Serwer sprawdza czy dane są poprawne. Jeśli tak, wysyła w odpowiedzi 128 bitowy, losowy i unikatowy token. Token jest zapisywany w tabeli „*GLOGIN*”.
3. Użytkownik wysyła token razem z każdym zapytaniem do API.
4. Serwer obsługuje zapytanie tylko jeśli token się zgadza. W niektórych przypadkach serwer sprawdza czy token jest przypisany do tego Użytkownika który go wysłał lub czy jest przypisany do Administratora. Dzięki temu Użytkownik nie może zmienić danych innego Użytkownika czy korzystać z uprawnień Administratora.
5. Jeżeli Użytkownik błędnie poda hasło 20 razy w ciągu jednego dnia, jego adres IP zostaje zablokowany do końca dnia. Jest to zabezpieczenie przed atakami typu brute-force.

Zgodnie z wymaganiami, hasło musi mieć co najmniej 8 znaków, małe i duże litery oraz cyfry. Daje to 62 znaki w zbiorze i $2 * 10^{14}$ kombinacji. Dla ataku brute-force dokonywanego z 1 adresu IP, przyjmując że średnio hasło zostanie złamane po sprawdzeniu połowy kombinacji, czas łamania hasła wynosi: $10^{14}/20 = 5 * 10^{12}$ dni, czyli około 14 miliardów lat. Oczywiście atak z wielu adresów IP i z wykorzystaniem tęczowych tablic będzie dużo bardziej skuteczny. Dlatego warto stosować jak najmocniejsze hasła i zapewniać mechanizmy ich odzyskiwania. Czas łamania 128-bitowego tokena na milionie komputerów wykonujących miliard prób na sekundę wynosi $2^{127} * 10^{-6} * 10^{-9} s =$ około 11 miliardów lat. Co więcej, tokeny te zmieniają się z każdym logowaniem. Kod odpowiadający za generowanie losowego tokena:

```
private String generateToken()
{
```

```

        String newToken = UUID.randomUUID().toString();
        //Ten warunek gwarantuje że w tym samym momencie nie ma dwóch takich samych tokenów
        if (gLoginRep.findByToken(newToken) == null)
            return newToken;
        else
            return generateToken();
    }

```

Serwis odpowiadający za zliczanie nieudanych prób logowania i blokowanie adresów IP:

```

package com.koziejaj.client.GLogin;

import ..

@Service
public class GLoginBanService {
    //Maksymalna liczba prób na jeden adres IP
    private final int MAX_ATTEMPT = 20;
    //Struktura przechowująca adresy IP i liczbę ich nieudanych prób
    private LoadingCache<String, Integer> attemptsCache;

    public GLoginBanService() {
        super();
        //Zmienna attemptsCache jest czyszczona po 1 dniu.
        attemptsCache = CacheBuilder.newBuilder()
            .expireAfterWrite(1, TimeUnit.DAYS).build(new
                CacheLoader<String, Integer>() {
                    public Integer load(String key) {
                        return 0;
                    }
                });
    }

    //Metoda wywoływana przy nieudanej próbie
    public void loginFailed(String key) {
        int attempts = 0;
        try {

```

```

        attempts = attemptsCache.get(key);
    } catch (ExecutionException e) {
        attempts = 0;
    }
    attempts++;
    attemptsCache.put(key, attempts);
}
//Metoda wywoływana przy każdej próbie logowania
public boolean isBlocked(String key) {
    try {
        return attemptsCache.get(key) >= MAX_ATTEMPT;
    } catch (ExecutionException e) {
        return false;
    }
}
}

```

Każde zapytanie do API ma sprawdzany token:

```

@RequestMapping("/api/gusers")
//Parametr token jest odbierany przy każdym zapytaniu
public Iterable<GUser> gusers(@RequestParam(value="token")
    String token) {
    //Jeśli takiego tokena nie ma, zapytanie się nie wykonuje i zwraca
    null.
    if(gLoginRep.findByToken(token) != null) {
        Iterable<GUser> model = repository.findAll();
        return model;
    }
    else
        return null;
}

```

5.4 Reset i zmiana hasła

Zalogowany Użytkownik, może zmienić hasło, wysyłając stare hasło, nowe hasło, oraz swój login i token logowania. Jeżeli Użytkownik zapomni hasła,

może skorzystać z formularza dostępnego ze strony logowania. Po podaniu swojego adresu email, otrzymuje wiadomość z adresem URL (zawierającym 128 bitowy token) do formularza, pozwalającym na wpisanie nowego hasła. Formularz przekazuje do API adres email, nowe hasło i token. Jeśli dane się zgadzają, hasło zostaje zmienione.

5.5 SQL Injection

Atak ten [7] polega na wprowadzeniu dodatkowego kodu wykonywalnego (w języku SQL) do danych przysyłanych przez Użytkownika na serwer. Przykładowo, wysyłając jako imię wyszukiwanego użytkownika: „*x*’; *DROP TABLE GUSER*; *SELECT '1'*”, zapytanie wykonywane po stronie serwera mogłoby przyjąć formę:

```
SELECT * FROM GUSER WHERE LOGIN = 'x'; DROP TABLE  
GUSER; SELECT '1'
```

Zgodnie z testami, aplikacja wydaje się być odporna na tego typu ataki. Warstwy mapowania i repozytoriów samodzielnie dokonują filtrowania wprowadzonych danych. Kod SQL nie jest wykonywany bezpośrednio w żadnym miejscu aplikacji, tym bardziej w połączeniu z danymi przysłanymi przez Użytkownika.

Rozdział 6

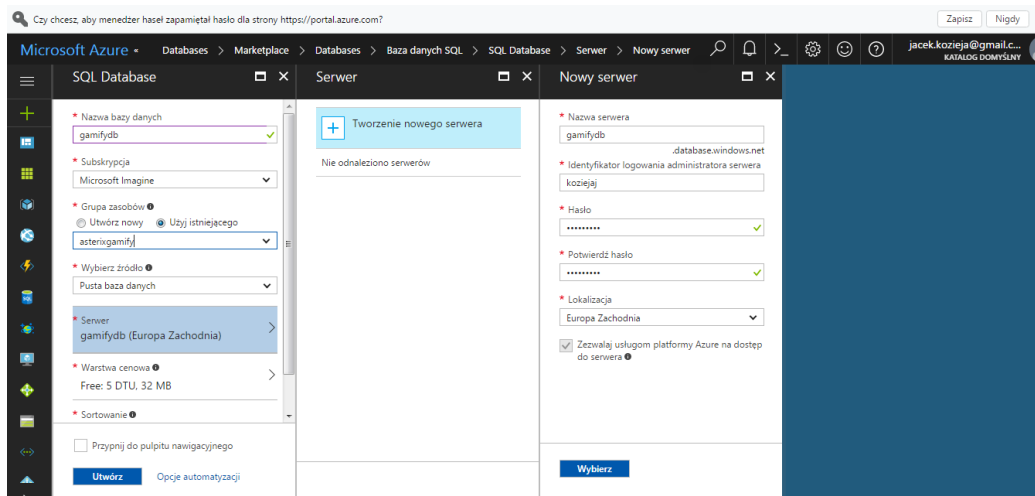
Wdrożenie systemu

Aby udostępnić aplikację szerszemu gronu użytkowników, zdecydowano się na wdrożenie jej w chmurze obliczeniowej. To rozwiązanie pozwala na upublicznienie aplikacji bez inwestycji w infrastrukturę serwerową. Co więcej, wiele firm realizujących tego typu usługi posiada w swojej ofercie darmowe pakiety i wersje próbne. W tym wypadku, zdecydowano się na platformę Azure od Microsoftu. Posiada ona darmowy pakiet, dostępny w ramach programu Imagine dla studentów Wydziału Elektrycznego. Sam proces można było podzielić na dwie części - wdrożenie bazy danych i wdrożenie aplikacji.

6.1 Migracja bazy danych

Serwer bazy danych Oracle jest dostępny na platformie Microsoftu tylko dla posiadaczy licencji Oracle w wersji Standard i Enterprise. Nieudane próby założenia darmowej wersji bazy, tylko to potwierdziły. Założono bazę danych opartą o serwer MSSQL. Wymagało to ręcznego odtworzenia wszystkich tabel. Na szczęście przepisanie kodu PL/SQL (Oracle) na Transact-SQL (Microsoft) nie sprawia większego problemu:

PL/SQL	Transact-SQL
<pre>CREATE TABLE "GBADGE" ("BADGE" VARCHAR2(255 CHAR) NOT NULL, "LOGIN" VARCHAR2(255 CHAR) NOT NULL, PRIMARY KEY ("BADGE", "LO- GIN"))</pre>	<pre>CREATE TABLE "GBADGE" ("BADGE" VARCHAR(255) NOT NULL, "LOGIN" VARCHAR(255) NOT NULL, PRIMARY KEY ("BADGE", "LO- GIN"))</pre>

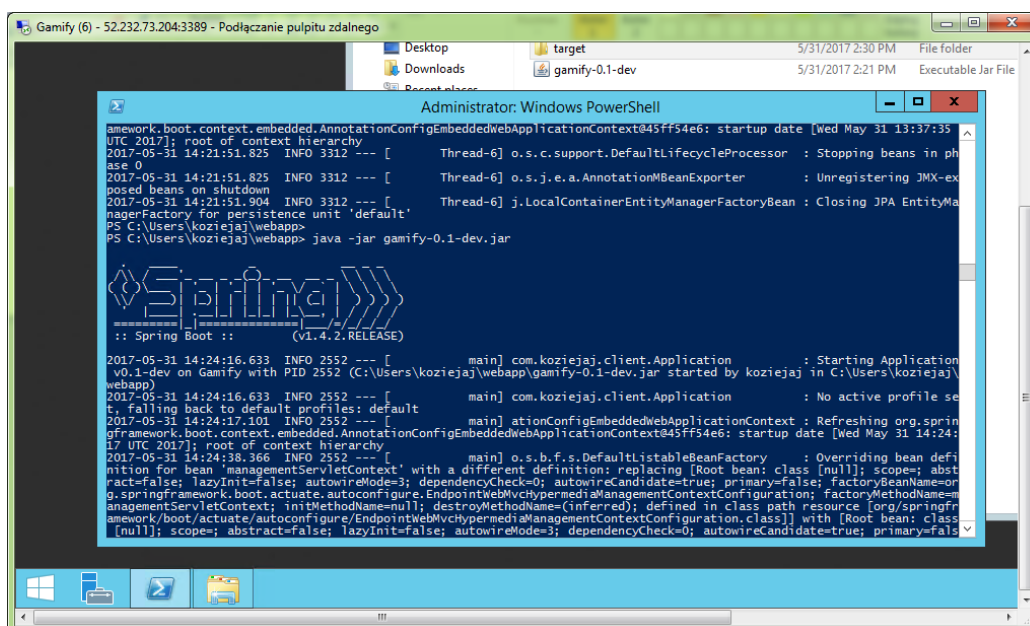


Rysunek 6.1: Ekran tworzenia nowej bazy danych SQL na Azure.

Należało także skonfigurować połączenie w pliku `application.properties` (analogicznie do wcześniejszego) i usunąć starą konfigurację.

6.2 Migracja aplikacji

Próby uruchomienia aplikacji poprzez plik `.jar` lub `.war`, korzystając ze zautomatyzowanej usługi Azure App Services zakończyły się niepowodzeniem. Ostatecznie uruchomiono maszynę wirtualną Windows Server 2012, zainstalowano na niej pakiet Java i otworzono port 80, będący standardowym portem protokołu HTTP. Maszyna jest dostępna poprzez protokół RDP. Na tak przygotowanym środowisku, uruchomienie pliku `.jar` aplikacji odbyło się bez przeszkód. Plik ten poza skompilowanym kodem, zawiera także kontener Apache Tomcat, który pełni tu funkcję serwera aplikacji. Jego działanie przedstawiono na Rysunku 6.2.



Rysunek 6.2: Postawienie własnej maszyny wirtualnej okazało się być najłatwiejszym rozwiązaniem w przypadku opisywanego projektu.

Rozdział 7

Działanie aplikacji

Rozdział 8

Wnioski

Zakres pracy został zrealizowany w zdecydowanej większości. Jedyne braki dotyczą elementów które zostały uznane za zbędne lub niespójne z resztą projektu. Są to:

- Administrator nie ma możliwości zakładania nowych kont Użytkowników. Służy do tego formularz rejestracyjny.
- Zrezygnowano z efektów dźwiękowych, ponieważ mogłyby być irytujące dla Użytkownika podczas codziennej pracy.
- Pominęto możliwość zapisywania się na konkretne zadania przed ich wykonaniem, by później odebrać za nie nagrody. Pomimo że jest to ciekawa funkcjonalność, zaspokaja tą samą potrzebę co odbieranie nagród przy pomocy kodu.

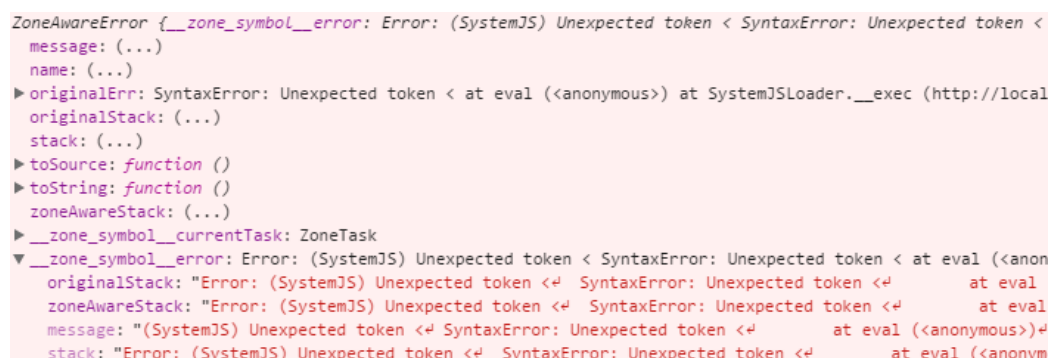
Wytworzona aplikacja stanowi dobry materiał wyjściowy dla aplikacji internetowych w języku Java. Modułowa architektura sprawia że może zostać łatwo rozbudowana o kolejne funkcjonalności. Budowa oparta na REST API umożliwia dopisanie kolejnych aplikacji klienckich - przykładowo na platformę Android z wykorzystaniem tej samej warstwy serwerowej. Wykres na Rysunku 8.1 przedstawia medianę czasów ładowania aplikacji w porównaniu z popularnymi portalami. Aplikacja teoretycznie powinna dążyć do prędkości uzyskiwanej przez stronę startową facebooka - obydwie witają użytkownika ekranem logowania. Czas ten jest jednak dużo gorszy. Wynika to z (między wieloma innymi czynnikami) filozofii aplikacji klienckiej napisanej w szkielecie Angular. Jako tak zwane „single-page app”, wczytuje ona cały kod aplikacji przy pojedynczym wczytaniu strony - pliki HTML i JS wszystkich podstron. Przy przejściu na kolejne podstrony następują jedynie odpowiednie zapytania do API. Takie podejście daje dużo lepsze wyniki w aplikacjach z których korzysta się dłużej, przemieszczając po wielu zakładkach. Aplikacji Angular nie



Rysunek 8.1: Prędkość wczytywania aplikacji na pierwszy rzut oka nie zachwyca.

warto często odświeżać, powinna sama dbać o aktualność przedstawionych danych. Dłuższy czas pierwszego ładowania jest tu kompromisem, wynagradzanym przez późniejsze szybsze działanie aplikacji.

Podczas projektu natknięto się na wiele błędów, których przyczyny okazały się trudne do odnalezienia. Szkielet Angular potrafił zwrócić błąd wewnątrz własnej, wewnętrznej biblioteki, nie dając żadnej wskazówki dotyczącej faktycznego błędu w kodzie. Przykład jest widoczny na Rysunku 8.2. Tylko pisząc kod linijka po linijce i testując co chwilę, możliwe było odnaj-



```
ZoneAwareError {__zone_symbol__error: Error: (SystemJS) Unexpected token < SyntaxError: Unexpected token <
  message: (...)
  name: (...)
  ▶ originalErr: SyntaxError: Unexpected token < at eval (<anonymous>) at SystemJSLoader.__exec (http://local
  originalStack: (...)
  stack: (...)
  ▶ toSource: function ()
  ▶ toString: function ()
  zoneAwareStack: (...)
  ▶ __zone_symbol__currentTask: ZoneTask
  ▼ __zone_symbol__error: Error: (SystemJS) Unexpected token < SyntaxError: Unexpected token < at eval (<anon
    originalStack: "Error: (SystemJS) Unexpected token <# SyntaxError: Unexpected token <# at eval
    zoneAwareStack: "Error: (SystemJS) Unexpected token <# SyntaxError: Unexpected token <# at eval
    message: "(SystemJS) Unexpected token <# SyntaxError: Unexpected token <# at eval (<anonymous>)#
    stack: "Error: (SystemJS) Unexpected token <# SyntaxError: Unexpected token <# at eval (<anonym
```

Rysunek 8.2: Błąd biblioteki SystemJS tak naprawdę oznaczał problem z definicją ścieżki do biblioteki, której chciano użyć w projekcie.

dowanie źródeł problemów. W połączeniu z długim czasem rekompilacji i restartu serwera (około 30 sekund na wydajnej maszynie), proces wytwórczy można ocenić jako dosyć żmudny. Sposób połączenia aplikacji klienckiej i serwerowej we wspólnym projekcie, korzystając z dwóch szkieletów, także z początku nie był zagadnieniem trywialnym. Rodzi się pytanie - czy tyle warstw aplikacji jest faktycznie potrzebne? Na pewno szkielety i biblioteki usprawniają pracę programisty. Warto jednak stosować się do zasad YAGNI i KISS - nie dodawać do projektu elementów niepotrzebnych i unikać przesadnej złożoności.

Bibliografia

- [1] Pivotal Software, „Guides”
<https://spring.io/guides>
- [2] Tutorials Point, „Spring - Bean Definition”, „Spring - Java Based Configuration”
<http://www.tutorialspoint.com/spring/>
- [3] Apache Software Foundation, „Documentation”
<https://maven.apache.org/guides/>
- [4] Scott Urman, Ron Hardman, Michael McLaughlin, „Oracle Database 10g : programowanie w języku PL/SQL”, Helion, 2008
- [5] Oracle, „Indexes and Index-Organized Tables”, „Partitions, Views, and Other Schema Objects”
https://docs.oracle.com/cd/E11882_01/server.112/e40540/toc.htm
- [6] Wikipedia, „Grywalizacja”
<https://pl.wikipedia.org/wiki/Grywalizacja>
- [7] OWASP, „SQL Injection”, „Cross-Site Request Forgery (CSRF)”, „Insecure Transport”
<https://www.owasp.org>
- [8] Bruce Eckel, „Thinking in Java : edycja polska”, Helion, 2006

Opinia

o pracy dyplomowej magisterskiej wykonanej przez dyplomanta

Zdolnego Studenta i Pracowitego Kolegę

Wydział Elektryczny, kierunek Informatyka, Politechnika Warszawska

Temat pracy

TYTUŁ PRACY DYPLOMOWEJ

Promotor: **dr inż. Miły Opiekun**

Ocena pracy dyplomowej: **bardzo dobry**

Treść opinii

Celem pracy dyplomowej panów dolnego Studenta i Pracowitego Kolegi było opracowanie systemu pozwalającego symulować i opartego o oprogramowanie o otwartych źródłach (ang. Open Source). Jak piszą Dyplomanci, starali się opracować system, który łatwo będzie dostosować do zmieniających się dynamicznie wymagań, będzie miał niewielkie wymagania sprzętowe i umożliwiał dalszą łatwą rozbudowę oraz dostosowanie go do potrzeb. Przedstawiona do recenzji praca składa się z krótkiego wstępu jasno i wyczerpująco opisującego oraz uzasadniającego cel pracy, trzech rozdziałów (2-4) zawierających opis istniejących podobnych rozwiązań, komponentów rozpatrywanych jako kandydaci do tworzonego systemu i wreszcie zagadnień wydajności wirtualnych rozwiązań. Piąty rozdział to opis przygotowanego przez Dyplomantów środowiska obejmujący opis konfiguracji środowiska oraz przykładowe ćwiczenia laboratoryjne. Ostatni rozdział pracy to opis możliwości dalszego rozwoju projektu. W ramach przygotowania pracy Dyplomanci zebrali i przedstawili w bardzo przejrzysty sposób duży zasób informacji, co świadczy o dobrej orientacji w nowoczesnej i ciągle intensywnie rozwijanej tematyce stanowiącej zakres pracy i o umiejętności przejrzystego przedstawienia tych wyników. Praca zawiera dwa dodatki, z których pierwszy obejmuje wyniki eksperymentów i badań nad wydajnością, a drugi to źródła skryptów budujących środowisko.

Dyplomanci dość dobrze zrealizowali postawione przed nimi zadanie, wykazali się więc umiejętnością zastosowania w praktyce wiedzy przedstawionej w rozdziałach 2-4. Uważam, że cele postawione w założeniach pracy zostały pomyślnie zrealizowane. Proponuję ocenę bardzo dobrą (5).

(data, podpis)

Recenzja

pracy dyplomowej magisterskiej wykonanej przez dyplomanta

Zdolnego Studenta i Pracowitego Kolegę

Wydział Elektryczny, kierunek Informatyka, Politechnika Warszawska

Temat pracy

TYTUŁ PRACY DYPLOMOWEJ

Recenzent: **prof. nzw. dr hab. inż. Jan Surowy**

Ocena pracy dyplomowej: **bardzo dobry**

Treść recenzji

Celem pracy dyplomowej panów dolnego Studenta i Pracowitego Kolegi było opracowanie systemu pozwalającego symulować i opartego o oprogramowanie o otwartych źródłach (ang. Open Source). Jak piszą Dyplomanci, starali się opracować system, który łatwo będzie dostosować do zmieniających się dynamicznie wymagań, będzie miał niewielkie wymagania sprzętowe i umożliwiał dalszą łatwą rozbudowę oraz dostosowanie go do potrzeb. Przedstawiona do recenzji praca składa się z krótkiego wstępu jasno i wyczerpująco opisującego oraz uzasadniającego cel pracy, trzech rozdziałów (2-4) zawierających bardzo solidny i przejrzysty opis: istniejących podobnych rozwiązań (rozdz. 2), komponentów rozpatrywanych jako kandydaci do tworzonego systemu (rozdz. 3) i wreszcie zagadnień wydajności wirtualnych rozwiązań, zwłaszcza w kontekście współpracy kilku elementów sieci (rozdział 4). Piąty rozdział to opis przygotowanego przez Dyplomantów środowiska obejmujący opis konfiguracji środowiska oraz przykładowe ćwiczenia laboratoryjne (5 ćwiczeń). Ostatni, szósty rozdział pracy to krótkie zakończenie, które wylicza także możliwości dalszego rozwoju projektu. W ramach przygotowania pracy Dyplomanci zebrali i przedstawili w bardzo przejrzysty sposób duży zasób informacji o narzędziach, Rozdziały 2, 3 i 4 świadczą o dobrej orientacji w nowoczesnej i ciągle intensywnie rozwijanej tematyce stanowiącej zakres pracy i o umiejętności syntetycznego, przejrzystego przedstawienia tych wyników. Drobne mankamenty tej części pracy to zbyt skrótowe omawianie niektórych zagadnień technicznych, zakładające dużą początkową wiedzę czytelnika i dość niestaranne podejście do powołań na źródła. Utrudnia to w pewnym stopniu czytanie pracy i zmniejsza jej wartość dydaktyczną (a ta zdaje się być jednym z celów Autorów), ale jest zrekompensowane zawartością merytoryczną. Praca zawiera dwa dodatki, z których pierwszy obejmuje wyniki eksperymentów i badań nad wydajnością, a drugi to źródła skryptów budujących środowisko. Praca zawiera niestety dość dużą liczbę drobnych błędów redakcyjnych, ale nie wpływają one w sposób istotny na jej czytelność i wartość. W całej pracy przewijają się samodzielne, zdecydowane wnioski

Autorów, które są wynikiem własnych i oryginalnych badań. Rozdział 5 i dodatki pracy przekonują mnie, że Dyplomanci dość dobrze zrealizowali postawione przed nimi zadanie. Pozwala to stwierdzić, że wykazali się więc także umiejętnością zastosowania w praktyce wiedzy przedstawionej w rozdziałach 2-4. Kończący pracę rozdział szósty świadczy o dużym (ale moim zdaniem uzasadnionym) poczuciu własnej wartości i jest świadectwem własnego, oryginalnego spojrzenia na tematykę przedstawioną w pracy dyplomowej. Uważam, że cele postawione w założeniach pracy zostały pomyślnie zrealizowane. Proponuję ocenę bardzo dobrą (5).

(data, podpis)