

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, trzech rozdziałów zawierających opis wykorzystanych technologii (szkielet Angular, 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. W rozdziale szóstym można zapoznać się z działaniem aplikacji w praktyce - zawiera on opis funkcjonalności z perspektywy Użytkownika końcowego. 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 three chapters depict used technologies (Angular framework, 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 sixth chapter includes description of functionalities from an end-user perspective - it shows a practical use of the application. 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 niczych 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	6
1.4 Wzorzec MVC	6
2 Angular	9
3 Spring	10
3.1 Metody konfiguracji	10
3.2 Spring Boot	11
3.3 Maven	14
3.4 Przydatne moduły	16
3.5 Implementacja REST API	17
3.5.1 Mapowanie obiektowo - relacyjne	17
3.5.2 Repozytoria	21
3.5.3 Kontroler	22
4 Baza danych Oracle	29
4.1 Integracja z projektem	29
4.2 Optymalizacja	30
5 Bezpieczeństwo	35
5.1 Ataki CSRF	35
5.2 Protokół HTTPS	36
5.3 Logowanie	37
5.4 Reset i zmiana hasła	40
5.5 SQL Injection	40
6 Wdrożenie systemu	41
6.1 Migracja bazy danych	41

6.2 Migracja aplikacji	42
7 Działanie aplikacji	44
7.1 Rejestracja i Reset hasła	44
7.2 Autoryzacja Użytkownika	44
7.3 Konfiguracja konta	50
7.4 Ranking	50
7.5 Wykonanie zadania	50
7.6 Wyświetlenie własnych osiągnięć	50
7.7 Autoryzacja Administratora	50
7.8 Usuwanie kont	55
7.9 Edycja aplikacji	55
7.10 Zarządzanie Zadaniami	55
7.11 Zarządzanie Odznakami	55
8 Wnioski	61
Bibliografia	65
Spis rysunków	67

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 zdobywac 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 [7] „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

Wymagania, dotyczące funkcjonalności aplikacji „Gamify” zostały przedstawione na poniższej liście:

1. Proces autoryzacji Użytkownika.
2. Proces rejestracji Użytkownika.
3. Proces zmiany zapomnianego hasła Użytkownika.

4. Konfiguracja konta Użytkownika - edycja publicznie dostępnych informacji, zmiana hasła.
5. Wyświetlanie listy Rankingowej Użytkowników.
6. Odebranie nagrody za wykonane Zadanie.
7. Wyświetlanie osiągnięć Użytkownika - wykonanych Zadań i zdobytych Odznak.
8. Proces autoryzacji Administratora.
9. Usuwanie kont Użytkowników.
10. Edycja aplikacji Użytkownika - definiowanie zasad i elementów grywalizacji w pewnym ograniczonym stopniu. Administrator może zmienić zawartość strony powitalnej. Administrator może zmienić niektóre pojęcia, wyświetlane w aplikacji Użytkownika. Administrator może edytować plik CSS, wykorzystywany w aplikacji. Administrator może edytować liczbę punktów potrzebnych do uzyskania kolejnych Poziomów.
11. Dodawanie nowych Zadań do wykonania.
12. Dodawanie i przyznawanie Odznak.

Jeśli chodzi o wymagania jakościowe, wczytywanie aplikacji nie powinno średnio trwać dłużej niż 5 sekund. Aplikacja powinna gwarantować odpowiedni poziom bezpieczeństwa - tożsamość Użytkowników powinna być weryfikowana podczas wykonywania wszystkich istotnych akcji, a wykradnięcie krytycznych danych powinno być utrudnione. Plany stworzenia aplikacji przybliżyć mogą poniższe przykładowe Przypadki Użycia, wykonane zgodnie z rozdziałem 6 podręcznika [11]:

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:

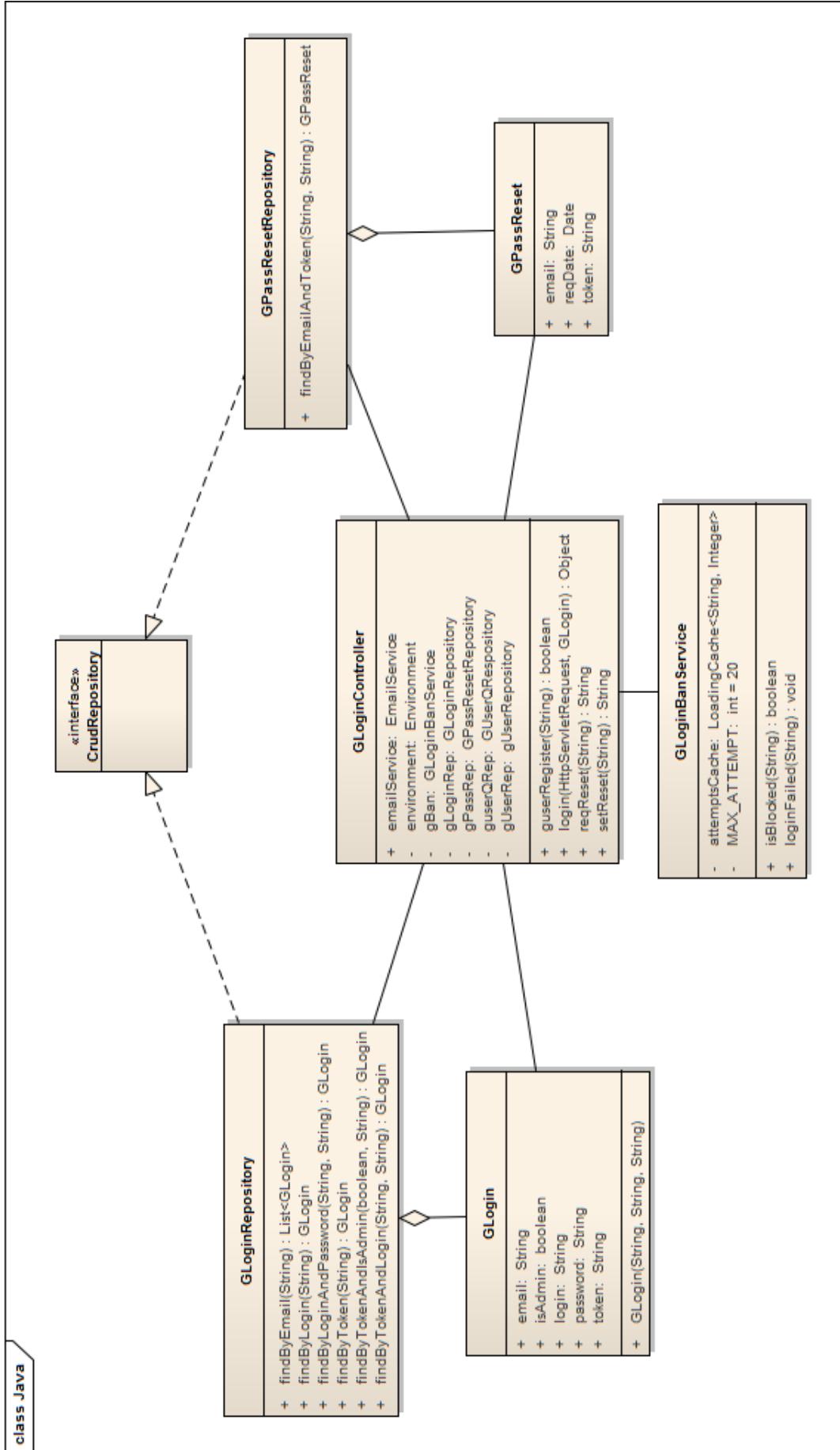
Dodanie nowego Zadania

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.

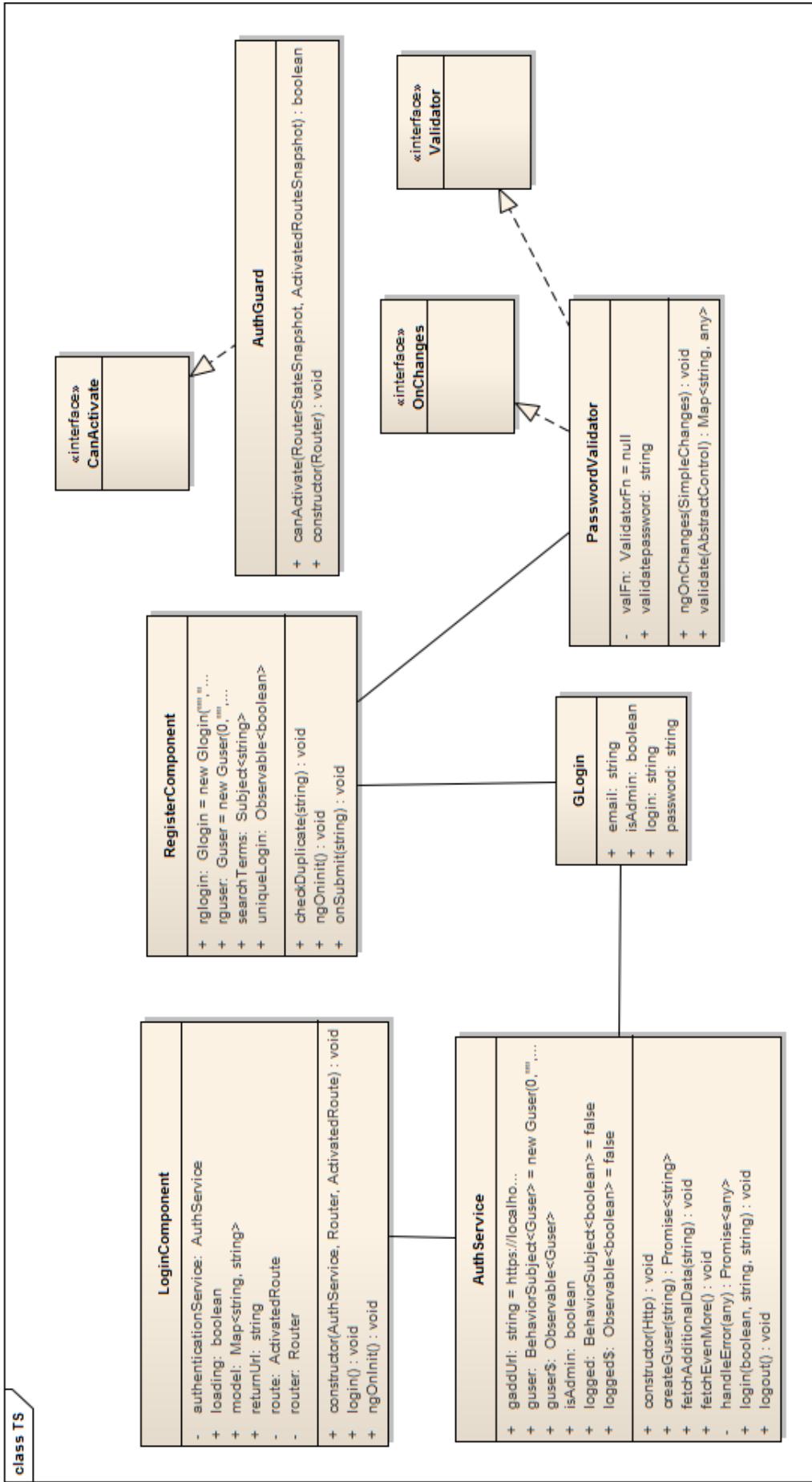
Przynajmniej Odznaki Użytkownikom

1. Administrator loguje się do Systemu.
2. Administrator wybiera opcję „Odznaki”.
3. System wyświetla okno edycji Odznak.
4. Administrator wybiera opcję „Przyznaj 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.

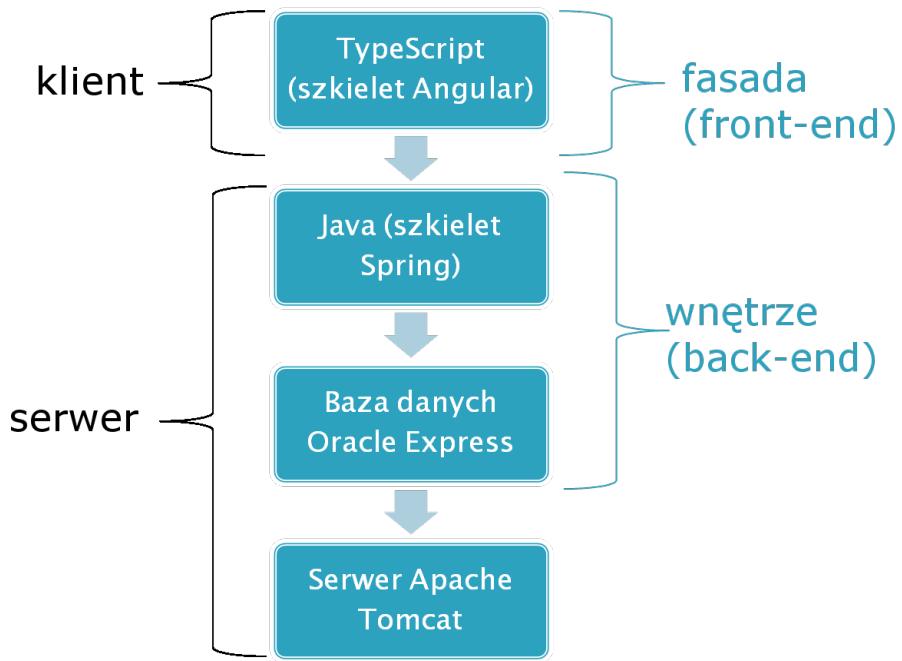
Diagramy klas na Rysunkach 1.1 i 1.2 opisują strukturę modułu GLogin - oddzielnie dla wnętrza („back-end”) i fasady („front-end”). Wykonane zostały zgodnie z podręcznikiem [11], rozdział 3. Stworzone diagramy są reprezentatywne dla całości projektu, zaś stworzenie pełnej dokumentacji uważam za odpowiedni temat dla oddzielnej pracy dyplomowej. Prywatne zmienne posiadające publiczne metody dostępowe 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.



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

1.3 Użyte technologie

Do wykonania aplikacji użyto technologii, przedstawionych na Rysunku 1.3. Jak widać, aplikacja została wykonana w architekturze Klient-Serwer, opisanej przez Stevensa [10]. Fasada oznacza kod wykonywany przez przeglądarkę internetową po stronie Klienta. Warstwa Klienta obsługuje żądania Użytkownika, wysyła zapytania do Serwera i prezentuje odpowiedzi Użytownikowi. Wnętrze oznacza kod wykonywany po stronie Serwera. Serwer nasłuchiwa żądań Klienta, wykonuje logikę biznesową związaną z danym zadaniem 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). Narażają 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 [12], oraz jest wykorzystywany przez wybrane szkielety aplikacji. MVC to wzorzec architektoniczny służący do organizowania struktury systemów interaktywnych. Składa się on z trzech części:

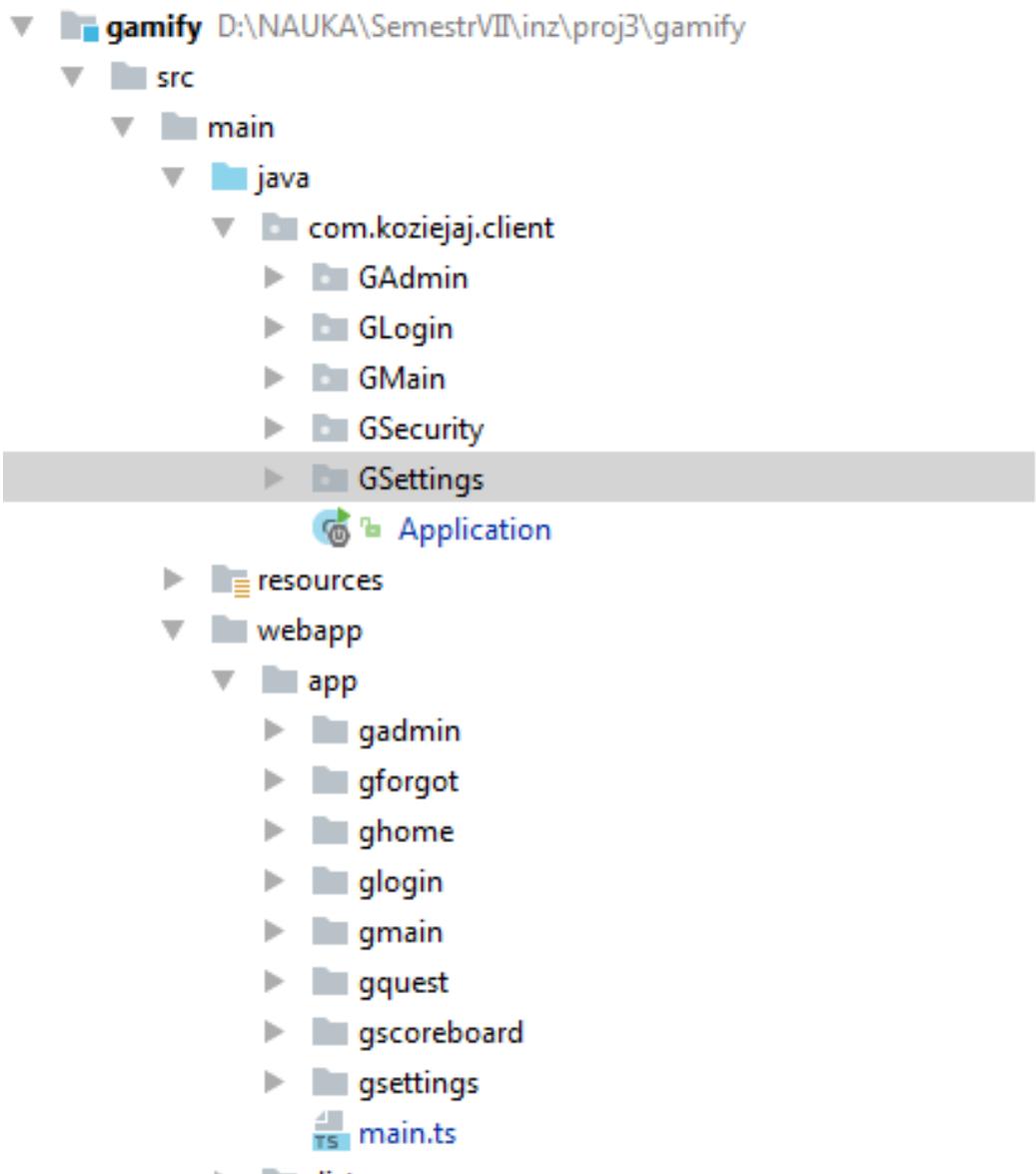
1. Model - reprezentuje dane i wykonuje logikę biznesową.
2. Widok - wyświetla dane pobrane z Modelu.
3. Kontroler - reaguje na dane wejściowe użytkownika. Przesyła żądania wykonania logiki biznesowej do Modelu i zmian Widoku.

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. Chcąc dowiedzieć się więcej na temat protokołu HTTP, warto sięgnąć do źródeł, czyli oficjalnej dokumentacji: [17] Modułowa struktura projektu jest przedstawiona na Rysunku

```
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.

1.5. Przedstawia się ona następująco: wewnątrz projektu Springa znajdują się 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

Do wytworzenia warstwy klienckiej użyto szkieletu Angular w wersji drugiej. Wybrano go ze względu na ciekawy, rzadko spotykany w aplikacjach internetowych sposób działania. Cały kod jest wczytywany przez przeglądarkę internetową na samym początku korzystania z aplikacji. Skutki takiego rozwiązania są pokazane w Rozdziale 8 Wnioski. Szkielet ten jest rozwijany przez firmę Google, która wyznacza trendy współczesnych usług sieciowych. Bez względu na specjalizację i miejsce programisty w stosie technologicznym, warto się z nim zapoznać. Kto wie kiedy przyjdzie nam pisać logikę biznesową, współpracującą z tym szkieletem. Szkielet ten wykorzystuje język Typescript. Jest to nadzbiór języka JavaScript i jest do niego kompliowany. Przykłady definicji klas były prezentowane wcześniej, na Rysunku 1.2. Rozwiązania używane w fasadzie aplikacji internetowych są bez wątpienia najszybciej rozwijającym się elementem stosu technologicznego. Już w momencie pisania pracy moda na Angulara ustępuje modzie na bibliotekę React, rozwijaną przez firmę Facebook. Jako że wnętrze aplikacji działa niezależnie od fasady, nie będzie ona opisywana szerzej w tej pracy. Zainteresowani, mogą zgłębić wiedzę na ten temat, czytając oficjalną dokumentację [1]. W przystępny sposób przeprowadza ona użytkownika przez proces tworzenia funkcjonalnej aplikacji i pozwala zaznajomić się z konceptami stojącymi za tym szkieletem.

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 Java Enterprise Edition, czyli serwerowej odmiany tego języka. Powstał jako alternatywa dla oficjalnych standardów serwerowych Java takich jak Enterprise JavaBeans. Spring nie narzuca jednego modelu programowania. Posiada wiele modułów które są przydatne przy pisaniu aplikacji, jednocześnie będąc całkowicie opcjonalnymi. Jego cechy są opisane w dokumentacji [16] Znaną cechą Springa jest „**odwrócenie sterowania**” (ang. Inversion of Control), czyli wzorzec 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 wzorzec 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 [3] 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 oddzielnnych 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 wstępna 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 [2]:

```
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 powinno 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 3.1 przedstawia uruchomioną aplikację, zaś Rysunek 3.2 - 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
basicErrorHandler
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 localhost: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

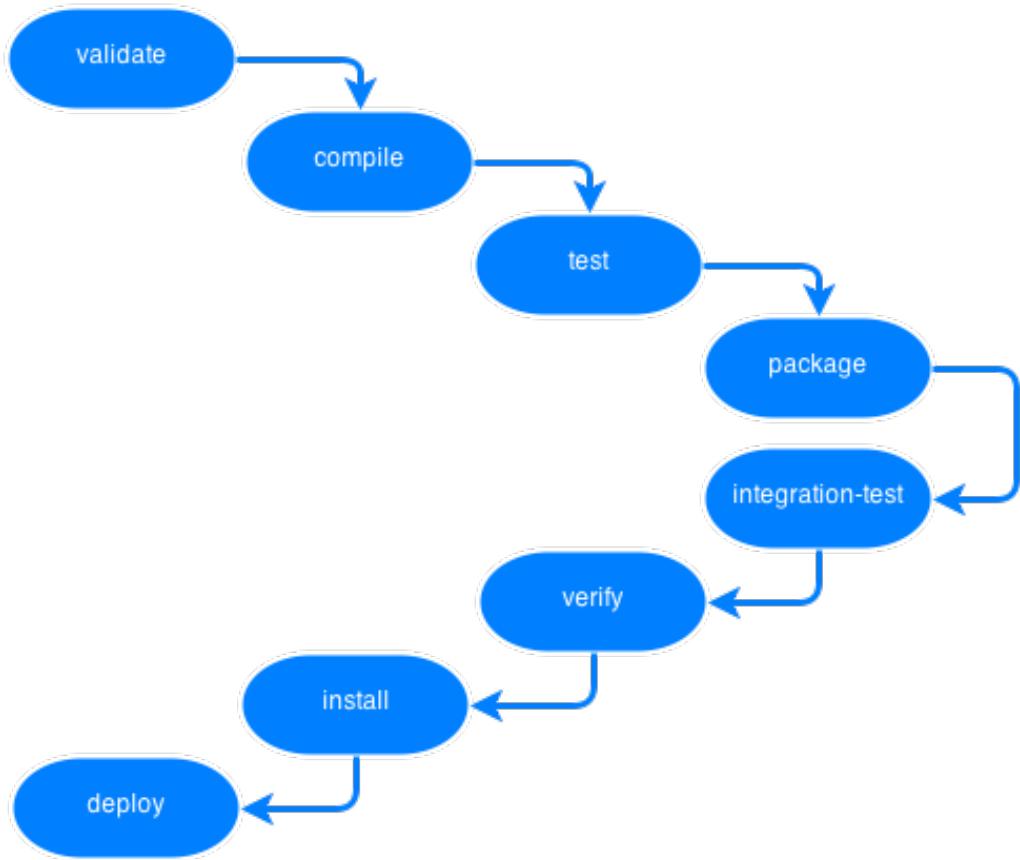
Maven to w narzędzie do zarządzania projektem - jego zależnościami (używanymi bibliotekami) i strukturą (podziałem na moduły). Maven automatyzuje budowę projektu zgodnie z poniższymi krokami, opisywanymi w pozycji [4]. Rysunek 3.3, przedstawiający cykl życia projektu, został zaczerpnięty z kursu [13]

1. Validate - walidacja kodu. Sprawdzana jest poprawność kodu źródłowego, umożliwiająca jego komplikację.
2. Compile - komplikacja, 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 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>
```



Rysunek 3.3: Uproszczony cykl życia projektu Maven.

```

<!-- Dziedziczenie po istniejącym projekcie -->
<parent>
<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 haśle bez potrzeby pisania jakiekolwiek logiki biznesowej. REST umożliwia wysyłanie zapytań do repozytoriów w sposób analogiczny do kontrolerów (Podrozdział 3.2), jednak ta funkcjonalność nie została wykorzystana w projekcie. Moduł pobrany jako: spring-boot-starter-data-rest.

- Spring Security - biblioteka zapewniająca bezpieczeństwo aplikacji, szczegółowo 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” [2]. 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 ...
//Anotacja oznaczająca klasę odpowiadającą tabeli bazy danych
@Entity
//Anotacja wczytująca definicję klucza głównego (primary key) – wyjątk
nienie w kolejnym listingu
@IdClass(GBId.class)
public class GBadge{

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

    //Konstruktory
    public GBadge() {
    }

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

    //Metody dostępowe
    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 implementuje 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 [2]. 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. Tworząc logikę biznesową, wykorzystano wiele metod i obiektów charakterystycznych dla języka Java. Ich szczegółowe omówienie można znaleźć w należącej do klasyki literatury informatycznej pozycji [9].

```
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
        (1000000000001, 10000000000001);
    while (gQuestRep.findById(myId) != null) {
        myId = ThreadLocalRandom.current().nextLong
            (10000000000001, 1000000000000001);
    }
    //Każde Zadanie ma Date 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
    ż zwrotną.
    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 Uzytkownik mogą pobrać parametry edytowalne aplikacji Uzytkownika – stronę powitalną, wzór na punkty i tym podbne.
@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.
Zwarcane są tu nazwy plików graficznych które odpowiadają
zapisom w tabeli. Odznaki są widoczne dzięki uniwersalnej ścieżce
API zwracającej obrazki, znajdującej 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.

@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, umożliwiający 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.

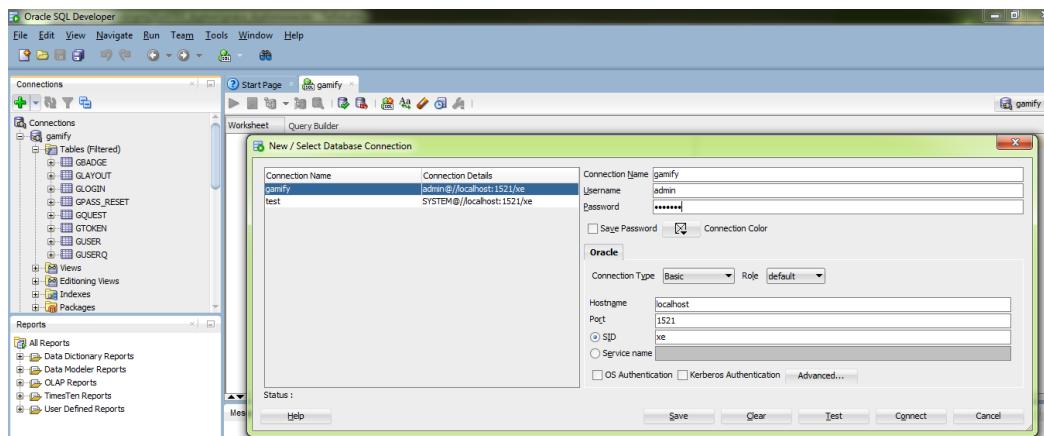
```

spring.datasource.url=jdbc:oracle:thin:@//localhost:1521/xe
spring.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

```

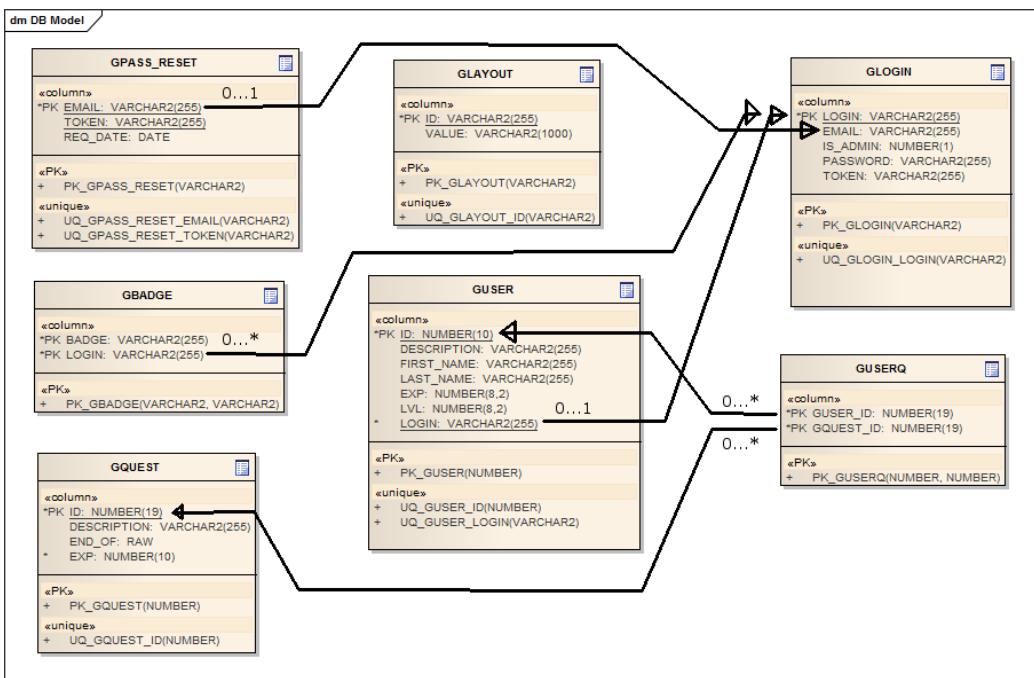
Informacje takie jak numer portu login, hasło są podawane podczas tworzenia nowej bazy danych w GUI narzędzia Oracle SQL Developer, przedstawionym na Rysunku 4.1. Alternatywnie można wykorzystać SQL*Plus, czyli program linii komend. Rysunek 4.2 przedstawia schemat bazy danych projektu.



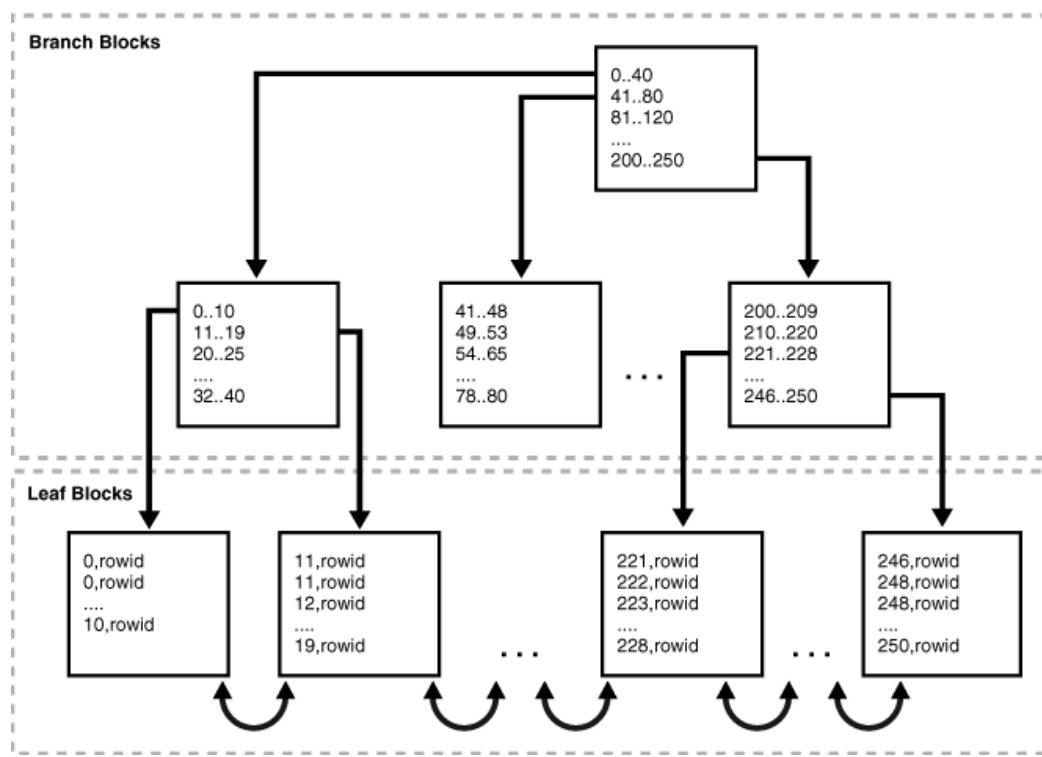
Rysunek 4.1: SQL Developer pozwala na kompleksowe zarządzanie bazą danych

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 [6]. Standardowo, indeks tabeli dworzony jest po kluczu głównym. Można dodać je też samodzielnie. Poniższa



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.

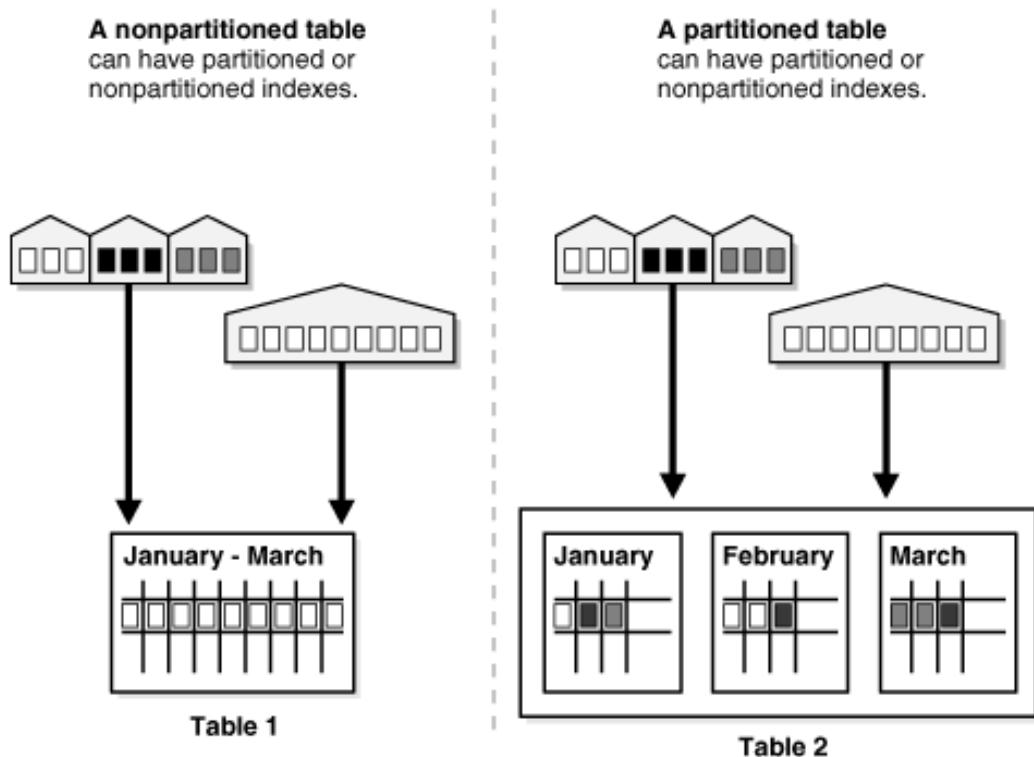


Rysunek 4.3: Liście B-drzewa przechowują rekordy, gałęzie usprawniają przeszukiwanie.

komenda doda do tabeli „*GUSER*” indeksowanie po kolumnie „*LOGIN*”, która jest wyszukiwana równie często co klucz główny „*ID*”.

```
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 partieje miesięcy czy lat. Zgodnie z dokumentacją, należy rozważyć partycjonowanie każdej tabeli przekraczającej 2 GB wielkości. Różnicę dobrze obrazuje rysunek 4.4, również zaczerpnięty z [6]: Chcąc partycjonować tabelę



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

„*GQUEST*” po roku zakończenia zadania, należy wykonać poniższą komendę. Niestety po jej wykonaniu okazało się że wersja Oracle Express nie obsługuje funkcji partycjonowania. Pisząc kod w PL/SQL, będący odmianą języka SQL dla produktów Oracle, posiliłowano się podręcznikiem Urmana i innych [5].

```
--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 [8] 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. Unieważnia 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, umoz
    liwia 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 na plik Cookie. Domyslnie 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 [8] 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: QazsedeC #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 tyczowych tablic będzie dużo bardziej skuteczny. Dlatego warto stosować jak najmocniejsze hasła i zapewniać mechanizmy ich odzyskiwania. Czas łamania 128-bitowego tokenu 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. W rozważaniach na temat entropii i siły haseł, przydatne okazały

sie być materiały z dziedziny kryptografii, jak chociażby [14]. 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 [8] polega na wprowadzeniu dodatkowego kodu wykonywalnego (w języku SQL) do danych przesyłanych 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 przesyłanymi przez Użytkownika.

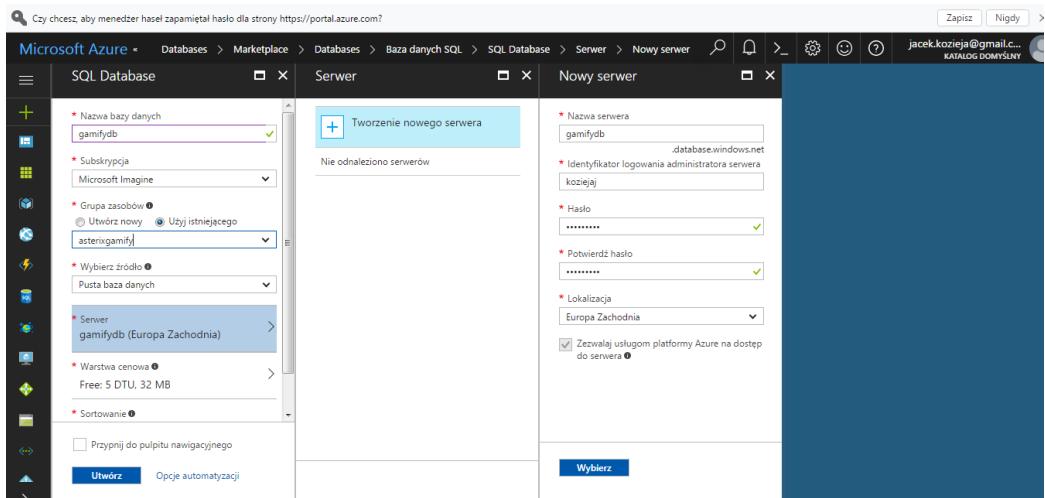
Rozdział 6

Wdrożenie systemu

Aby udostępnić aplikację szerszemu gronu użytkowników, zdecydowano się na wdrożenie jej w chmurze obliczeniowej. To rozwiązanie pozwala na upublicznenie 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. Niedane próby założenia darmowej wersji bazy, tylko to potwierdziły. Założono bazę danych opartą o serwer MSSQL. Rysunek 6.1 przedstawia okno tworzenia nowej bazy danych - jak wszystkich usługach platformy Azure, służy do tego przystępne GUI. Wymagało to ręcznego odtworzenia wszystkich tabel. Na szczęście przepisanie kodu PL/SQL (Oracle) na Transact-SQL (Microsoft) nie sprawia większego problemu, co widać na tabelce porównawczej.



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

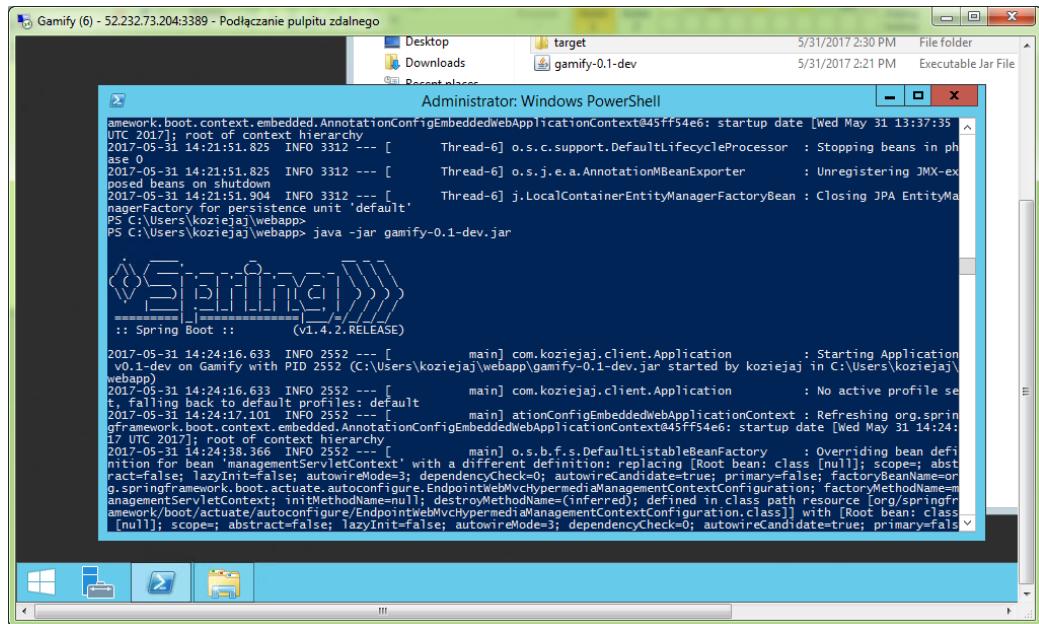
PL/SQL	Transact-SQL
<pre>CREATE TABLE "GBADGE" ("BADGE" VARCHAR2(255) CHAR NOT NULL, "LOGIN" VARCHAR2(255) CHAR NOT NULL, PRIMARY KEY ("BADGE", "LOGIN"))</pre>	<pre>CREATE TABLE "GBADGE" ("BADGE" VARCHAR(255) NOT NULL, "LOGIN" VARCHAR(255) NOT NULL, PRIMARY KEY ("BADGE", "LOGIN"))</pre>

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 konte-

ner Apache Tomcat, który pełni tu funkcję serwera aplikacji. Jego działanie przedstawiono na Rysunku 6.2.



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered was "java -jar gamify-0.1-dev.jar". The output displays Spring Boot's auto-configuration process, including the loading of the main application class and the configuration of various beans. The output ends with the message "Application running!阅读全文".

```
Administrator: Windows PowerShell
Desktop target 5/31/2017 2:30 PM File folder
Downloads gamify-0.1-dev 5/31/2017 2:21 PM Executable Jar File
Gamify (6) - 52.232.73.204:3389 - Podłączanie pulpitu zdalnego
Administrator: Windows PowerShell
PS C:\Users\koziejaj\webapp> java -jar gamify-0.1-dev.jar
:: Spring Boot :: (v1.4.2.RELEASE)
2017-05-31 14:24:16.633 INFO 2552 --- [           main] com.ko ziejaj.client.Application      : Starting Application v0.1-dev on Gamify with PID 2552 (C:\Users\ko ziejaj\webapp\gamify-0.1-dev.jar started by ko ziejaj in C:\Users\ko ziejaj\webapp)
2017-05-31 14:24:16.633 INFO 2552 --- [           main] com.ko ziejaj.client.Application      : No active profile set, falling back to default profiles: default
2017-05-31 14:24:17.101 INFO 2552 --- [           main] o.s.f.context.embedded.AnnotationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@45ff54e6: startup date [Wed May 31 14:24:17 UTC 2017]; root of context hierarchy
2017-05-31 14:24:38.366 INFO 2552 --- [           main] o.s.b.f.s.DefaultListableBeanFactory   : Overriding bean definition for bean 'managementContext' with a different definition: replacing Root bean: class [null]; scope=prototype; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; destroyMethodName=null; defined in class path resource [org/springframework/boot/actuate/autoconfigure/EndpointWebMvcHypermediaManagementContextConfiguration.class] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; destroyMethodName=null; defined in class path resource [org/springframework/boot/actuate/autoconfigure/EndpointWebMvcHypermediaManagementContextConfiguration.class]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; destroyMethodName=null; defined in class path resource [org/springframework/boot/actuate/autoconfigure/EndpointWebMvcHypermediaManagementContextConfiguration.class]]
```

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

Rozdział 7

Działanie aplikacji

Rozdział ten jest swoistym „podręcznikiem użytkownika”, przedstawia działanie aplikacji zgodnie z zakresem pracy, który został zdefiniowany w dokumencie „Plan pracy inżynierskiej”.

7.1 Rejestracja i Reset hasła

Na stronie głównej znajdują się odnośniki, pozwalające na rejestrację jak i ustanowienie nowego hasła gdy Użytkownik go zapomni. Rysunek 7.1 przedstawia formularz rejestracji. Podczas wypełniania sprawdzana jest unikalność loginu, poprawność hasła, zgodność adresu email ze wzorcem. Wszystkie pola poza „Opis” są wymagane. Rysunek 7.2 przedstawia działanie validatorów poszczególnych pól. W wypadku gdy Użytkownik zapomni hasła, może podać swój e-mail i otrzymać wiadomość z odnośnikiem. Po kliknięciu w odnośnik, Użytkownik jest przenoszony do panelu zmiany hasła na nowe. Rysunek 7.3 przedstawia ten proces.

7.2 Autoryzacja Użytkownika

Logowanie do aplikacji jest przedstawione na Rysunku 7.4. Próby przejścia do innych podstron bez zalogowania kończy się przekierowaniem do panelu logowania. Rysunek 7.5 to strona główna aplikacji, widoczna po zalogowaniu. Na górnym pasku znajdują się odnośniki do kolejnych podstron. W dolnym lewym rogu znajdują się informacje na temat zalogowanego Użytkownika, włącznie z awatarem (obrazkiem reprezentującym Użytkownika) i odnośnikiem do ustawień konta (ikona zębatki).

Gamify

Niezabezpieczona | https://localhost:7000/register

Rejestracja

Login

PilnyStudent

Hasło - musi zawierać minimum 8 znaków, dużą literę, małą literę i cyfrę.

Powtórz hasło

E-mail

abc@ee.pw.edu.pl

Imię

Bogdan

Nazwisko

Kowalski

Opis

Lubię rysować i jeść ziemnaki.

Zarejestruj

The screenshot shows a registration form titled "Rejestracja" on a website named "Gamify". The browser address bar indicates a self-signed certificate at https://localhost:7000/register. The form fields are as follows:

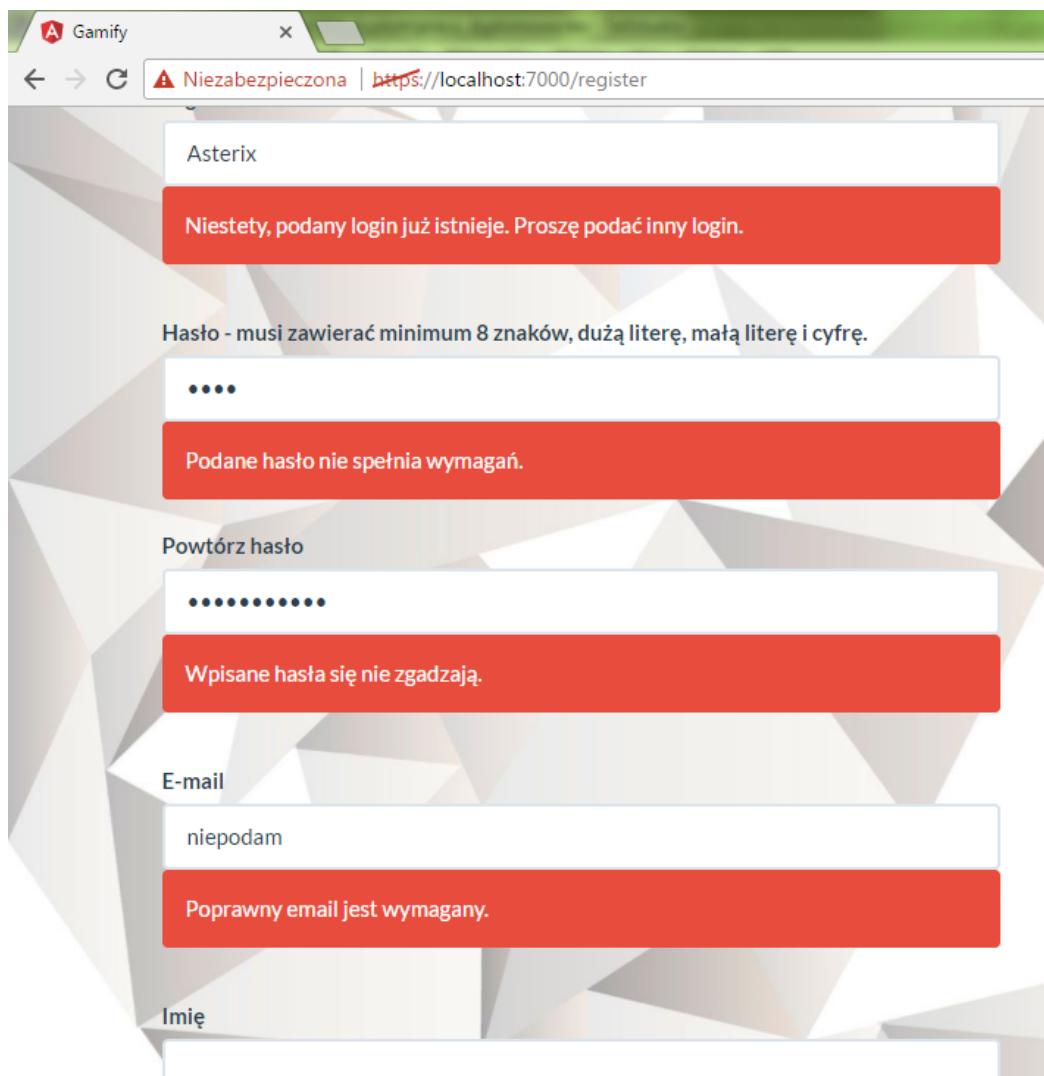
- Login: PilnyStudent
- Hasło - musi zawierać minimum 8 znaków, dużą literę, małą literę i cyfrę.

- Powtórz hasło

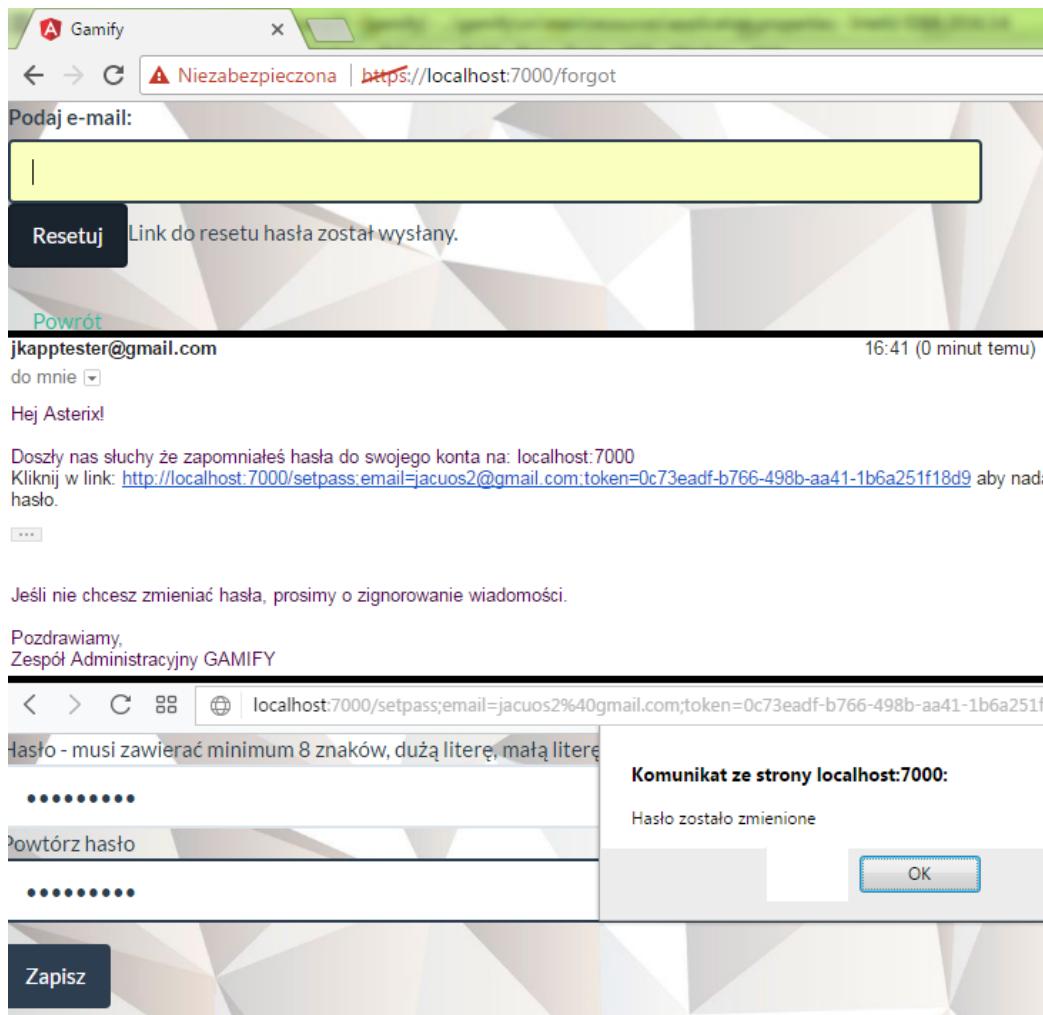
- E-mail: abc@ee.pw.edu.pl
- Imię: Bogdan
- Nazwisko: Kowalski
- Opis: Lubię rysować i jeść ziemnaki.

A green "Zarejestruj" button is located at the bottom of the form.

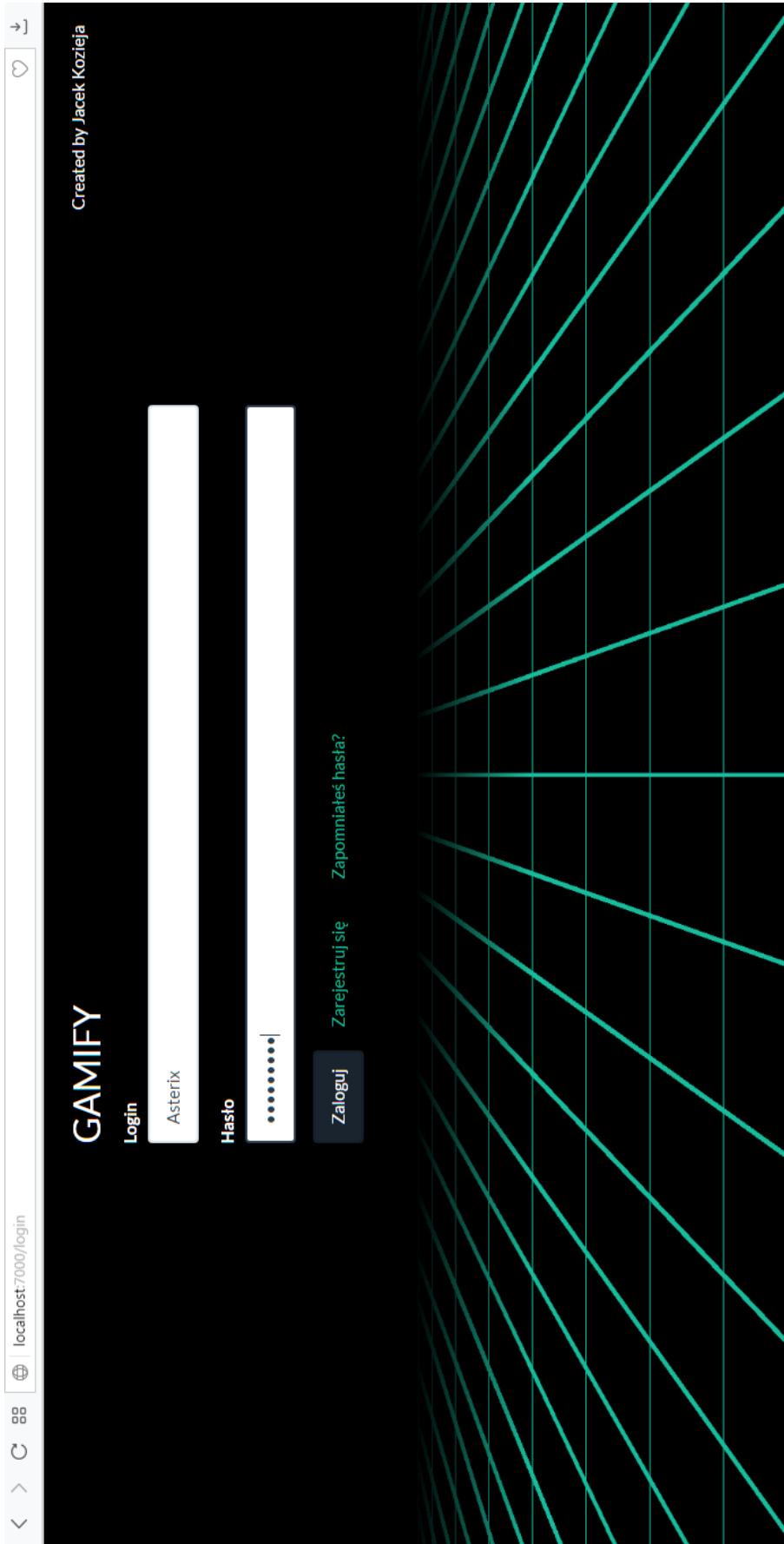
Rysunek 7.1: Wypełniony formularz rejestracji



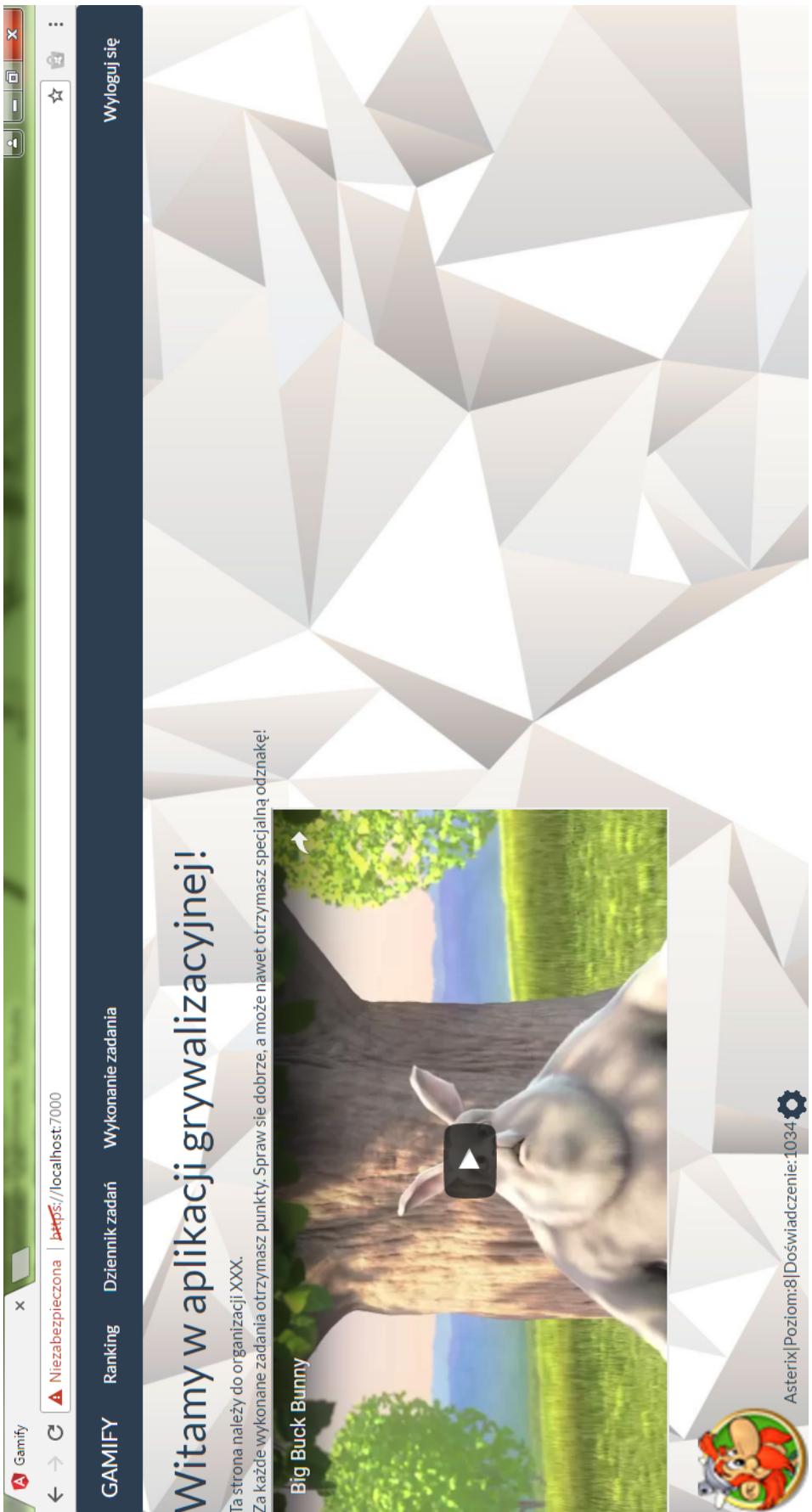
Rysunek 7.2: Błędnie wypełniony formularz rejestracji



Rysunek 7.3: Proces zmiany hasła



Rysunek 7.4: Ekran logowania



Rysunek 7.5: Strona główna aplikacji

7.3 Konfiguracja konta

Panel ustawień konta, umożliwia ustawienie nowego awatara, zmianę hasła i opisu. Rysunek 7.6 przedstawia ten panel podczas próby zmiany avatara. Po kliknięciu w stary avatar, otwiera się okno wyboru nowego pliku graficznego.

7.4 Ranking

Panel „*Ranking*” zawiera listę wszystkich Użytkowników. Okno wyszukiwania pozwala na filtrowanie listy po imieniu, nazwisku lub loginie. Małe guziki widoczne przy każdej kolumnie służą do sortowania listy. Rysunek 7.7 przedstawia panel „*Ranking*” posortowany malejaco według punktów.

7.5 Wykonanie zadania

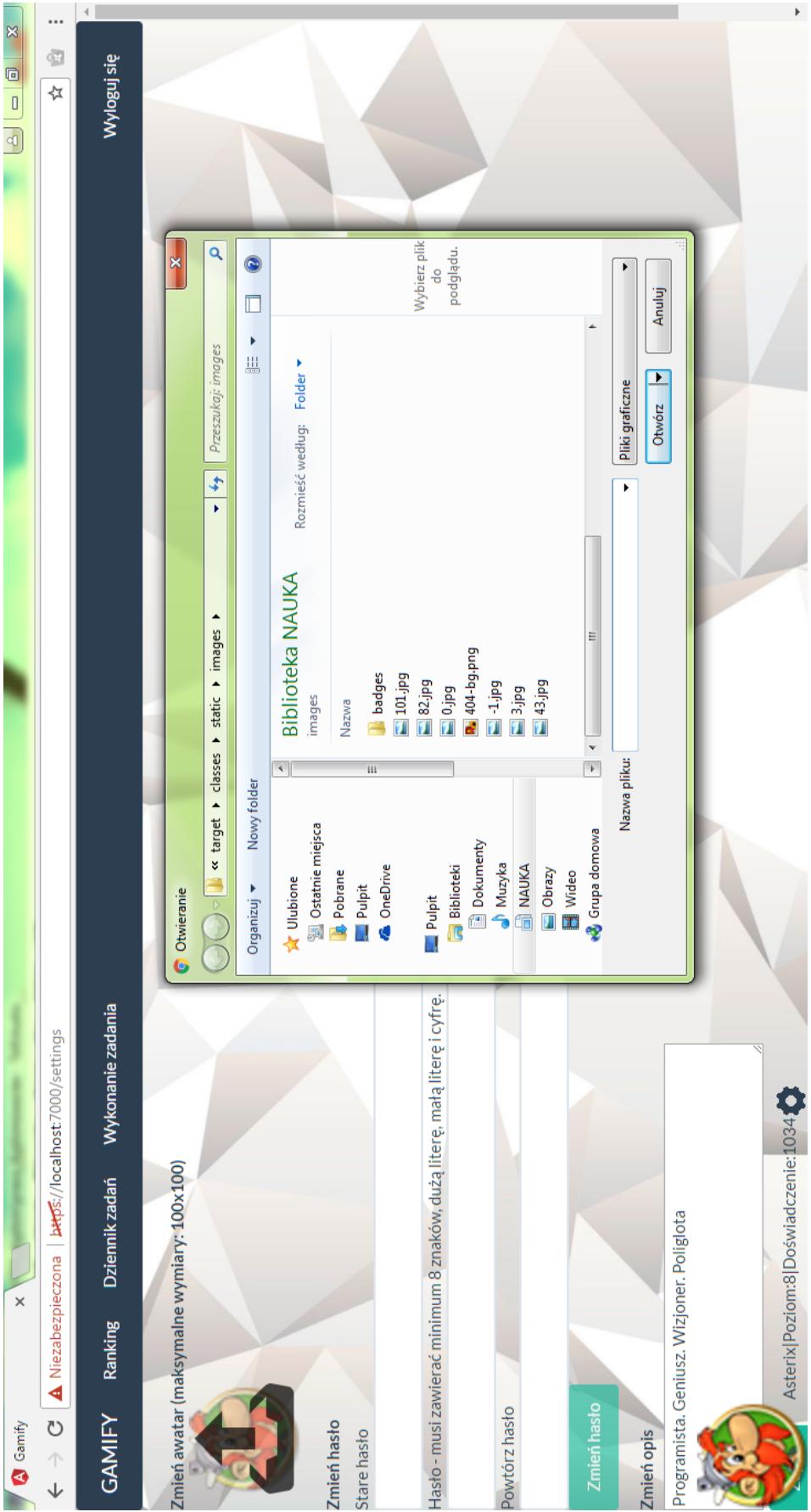
Panel „*Wykonanie zadania*” pozwala wpisać kod Zadania w celu otrzymania za nie Punktów. Kod jest otrzymywany od Administratora. Punkty mogą również zostać przekazane w formie kodu QR, który działa tak samo jak wpisanie kodu zadania. Rysunek 7.8 przedstawia proces wykonania Zadania.

7.6 Wyświetlenie własnych osiągnięć

W panelu „*Dziennik zadań*” widoczne są Zadania wykonane przez Użytkownika a także przyznane mu Odznaki. Przykład widoczny jest na Rysunku 7.9.

7.7 Autoryzacja Administratora

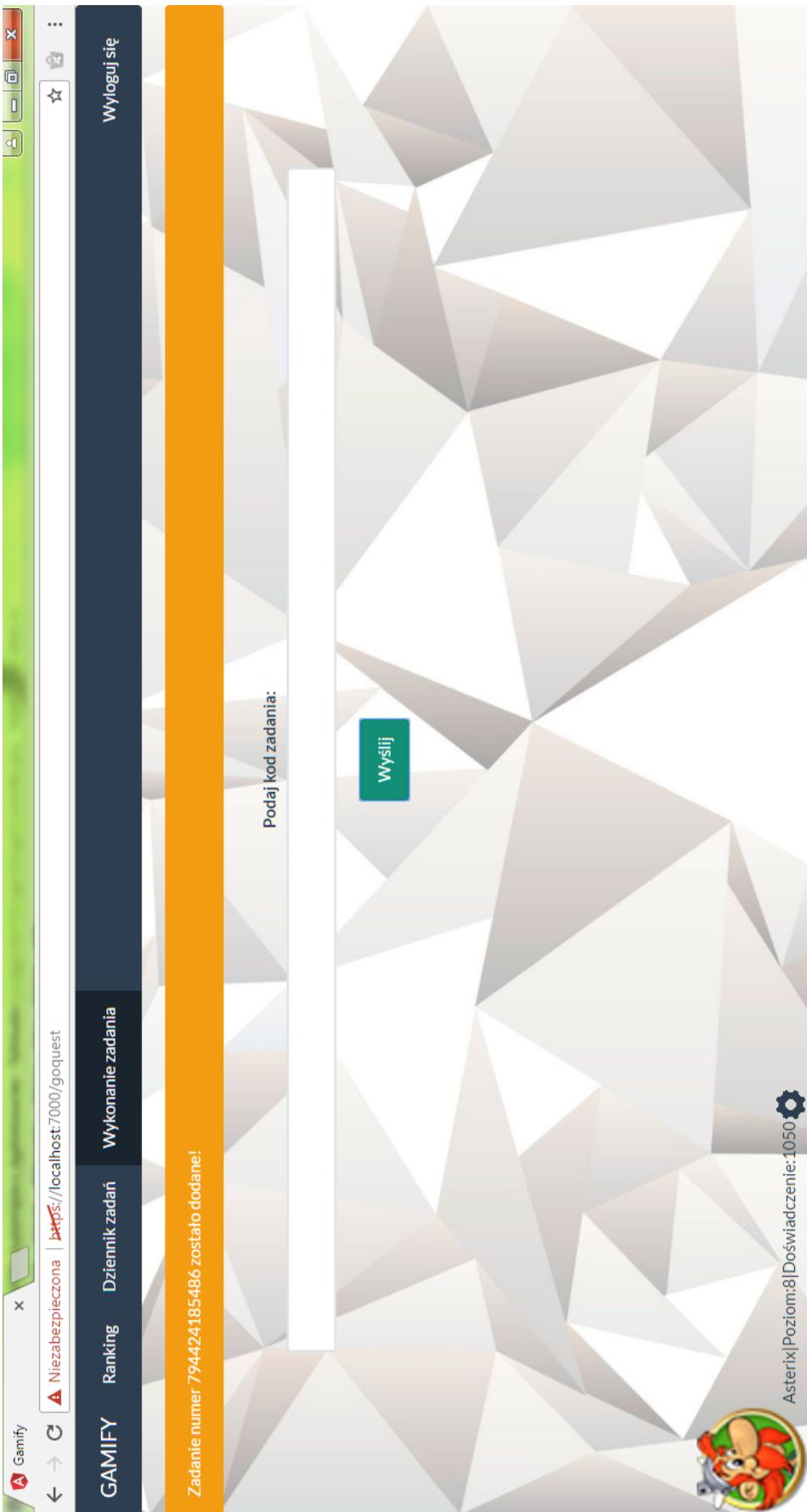
Pod ścieżką „/gadmin” znajduje się część aplikacji przeznaczona dla Administratora. Proces logowania wygląda identycznie co dla zwykłego Użytkownika, jednak przepuszczani są dalej tylko ci z odpowiednimi uprawnieniami. Rysunek 7.10 to widok okna logowania Administratora.



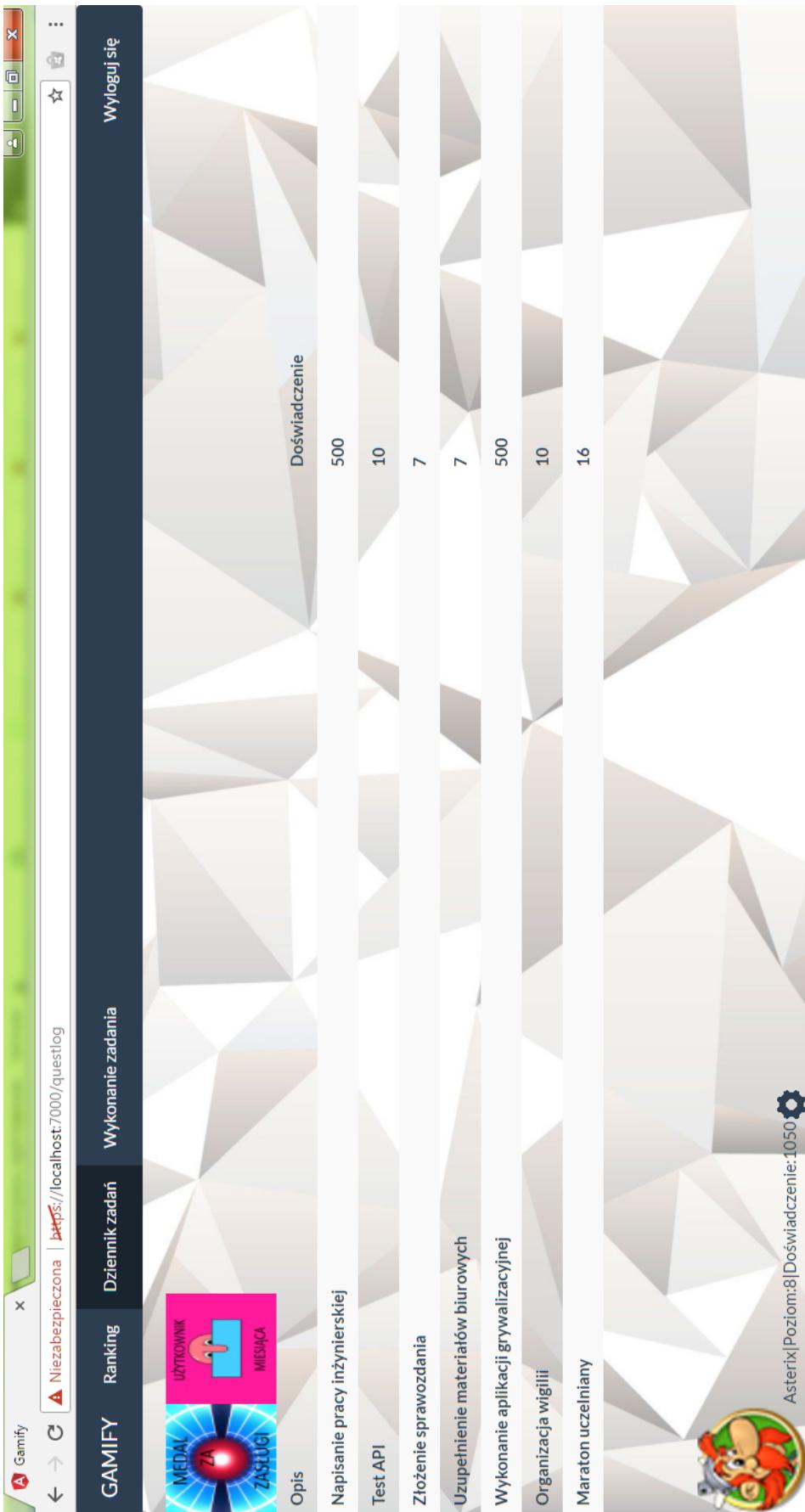
Rysunek 7.6: Panel ustawień konta

GAMIFY					
	Ranking	Dziennik zadań	Wykonanie zadania		
Asterix	8	Kozieja	8	1034	Programista. Geniusz. Wizjoner. Poligota
Cartman	9	Korgul	1	100	Valhalla
PanPrezes	10	Nowak	2	40	Ja tu rządzę
Dalton	11	Pachulski	1	20	sdefdfdf
PilnyStudent	12	Kowalski	2	20	Lubię rysować i jeść ziemnaki.
Szymon112233	13	Szymon	1	12	Ktos
Fizyk	14	Piórkowski	1	10	Wszystko jest relatywne
Raptor	15	Einstein	1	10	Mistrz cieni
Robak	16	Taperek	0	0	Polska jest najważniejsza
		Soplica	0	0	
 Asterix Pozłoty:8 Doświadczenie:1034					

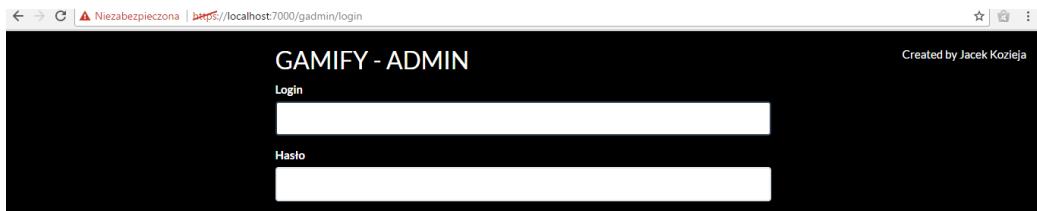
Rysunek 7.7: Panel z listą Użytkowników



Rysunek 7.8: Panel wykonania Zadania



Rysunek 7.9: Panel osiągnięć Użytkownika



Rysunek 7.10: Ekran logowania Administratora

7.8 Usuwanie kont

W panelu „*Użytkownicy*”, widok jest analogiczny do tego z panelu „*Ranking*”. Tutaj jednak istnieje możliwość usunięcia kont. Próbę wykorzystania tej funkcjonalności przedstawia Rysunek 7.11.

7.9 Edycja aplikacji

Panel „*Motyw*” skrywa opcje, pozwalającą na dostosowanie aplikacji w ograniczonym stopniu. Można tu zmienić nazwy kilku pojęć używanych w aplikacji, zmienić nagłówek widoczny na pasku nawigacyjnym. Można także zmienić zawartość strony startowej (obsługuje tagi html) i zawartość pliku css. Na końcu bardzo ciekawa opcja - wzór na progi punktowe, używany przy wyliczaniu awansów Użytkowników na kolejne poziomy. Rysunek 7.12 to widok panelu „*Motyw*” z aktualnie dobranymi wartościami.

7.10 Zarządzanie Zadaniami

Panel „*Zadania*” pozwala dodać nowe Zadanie i przeglądać te już istniejące. Na Rysunku 7.13 widoczny jest wygenerowany link a także kod QR.

7.11 Zarządzanie Odznakami

Pod panelem „*Odznaki*” można stworzyć nową Odznakę. Taka Odznaka może być nagrodą za specjalne zasługi lub oznaczeniem ważnej funkcji Użytkownika. Rysunek 7.14 przedstawia proces dodawania nowej Odznaki. Przyznawanie Odznak odbywa się poprzez wybranie Odznaki a następnie zaznaczenie Użytkowników z listy. Po kliknięciu przycisku potwierdzającego, Odznaki zostaną przyznane. Proces ten został uwieczniony na rysunku 7.15.

GAMIFY-ADMIN | Zadania | Odznaki | Użytkownicy

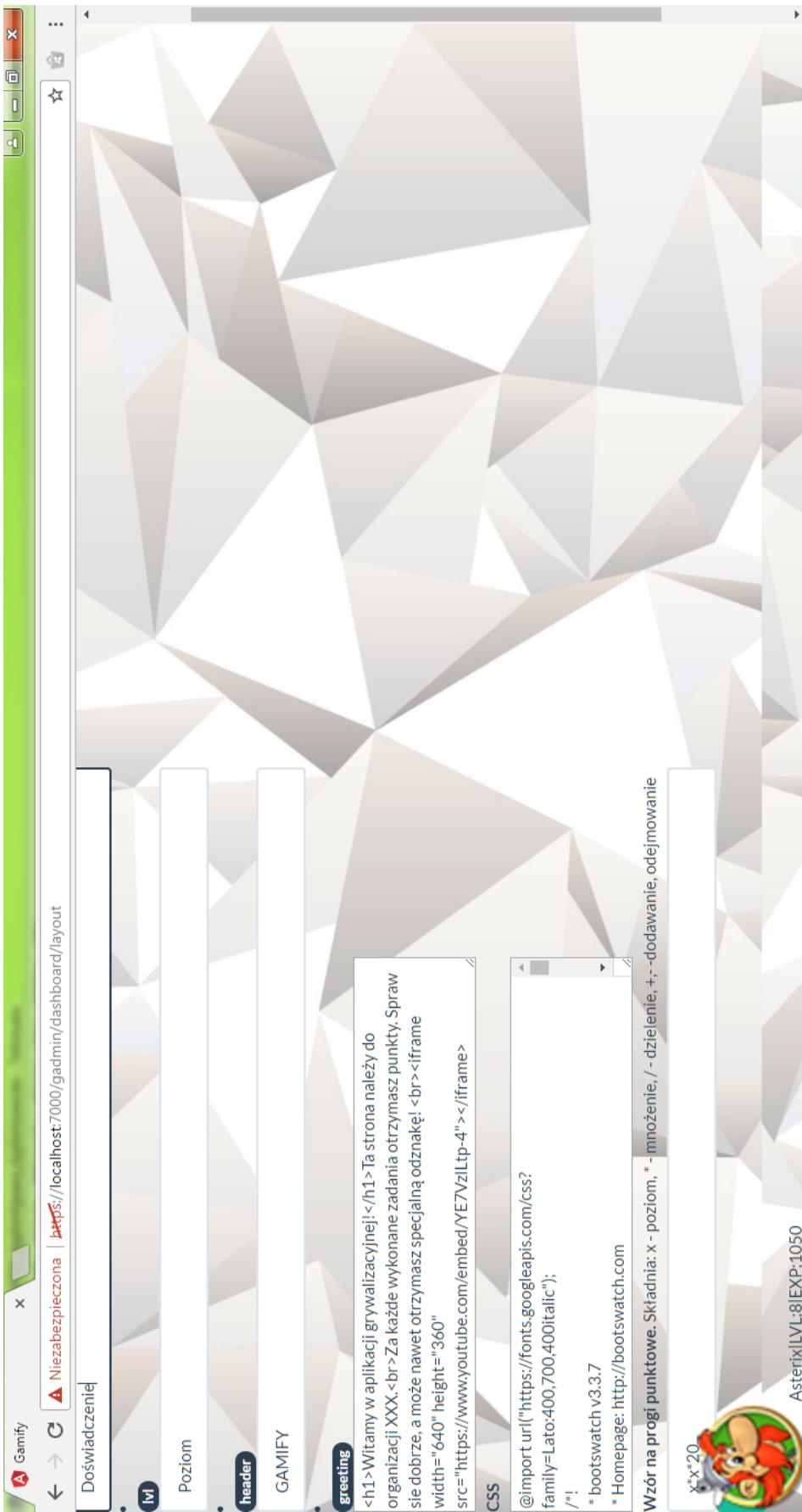
Szukaj...

Imię ▾	Nazwisko ▾	Opis ▾
Asterix	Jacek	Kozieja
PanPrezes	Jan	Nowak
Szymon11233	Szymon	Piórkowski
Cartman	Piotr	Korgul
Dalton	Krzysztof	Pachulski
Raptor	Krzysztof	Taperek
PilnyStudent	Bogdan	Kowalski
Fizyk	Albert	Einstein
Rebak	Jacek	Soplica

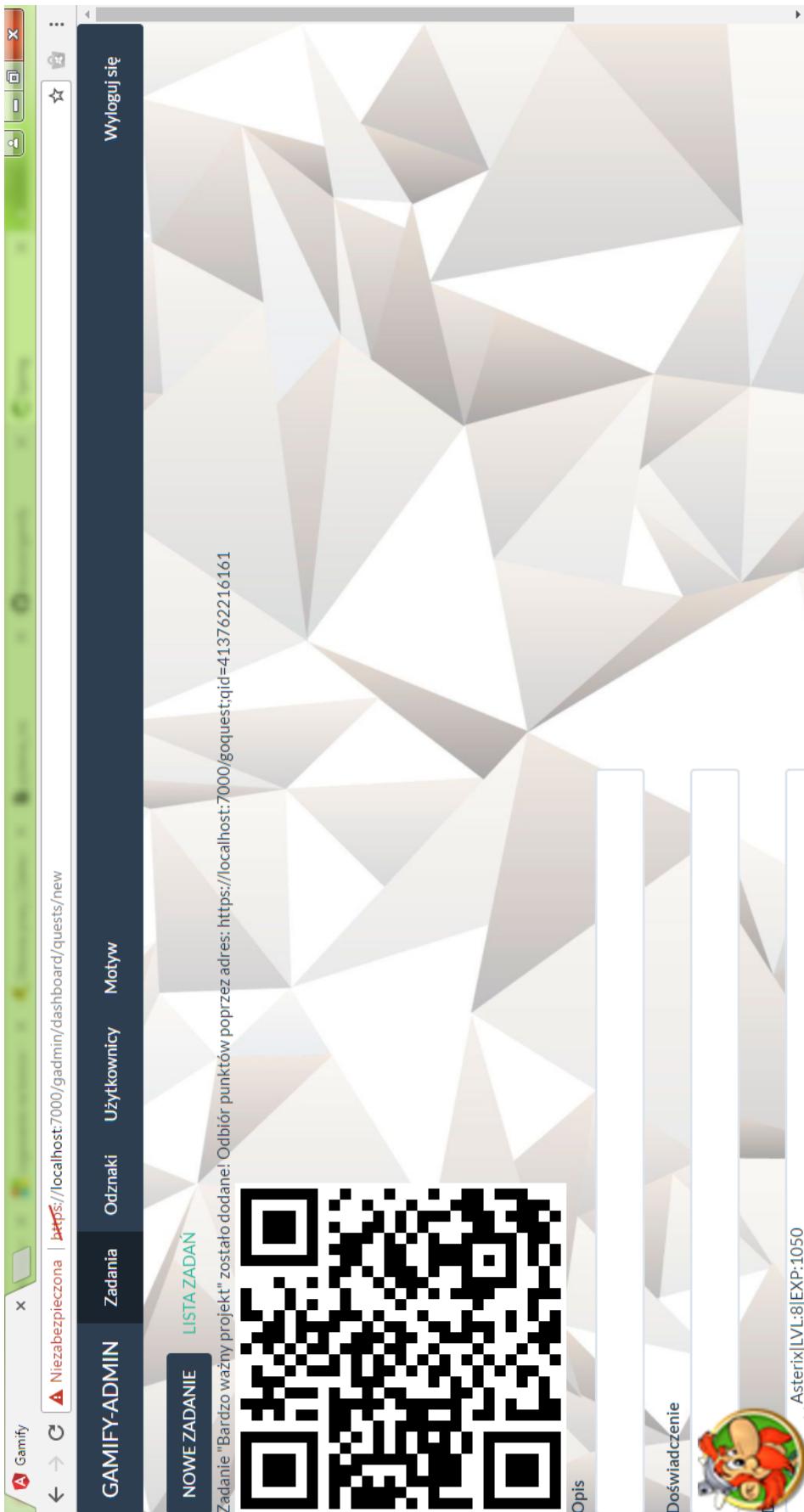
Komunikat ze strony localhost:7000:
Na pewno chcesz usunąć użytkownika Fizyk?

Wyloguj się

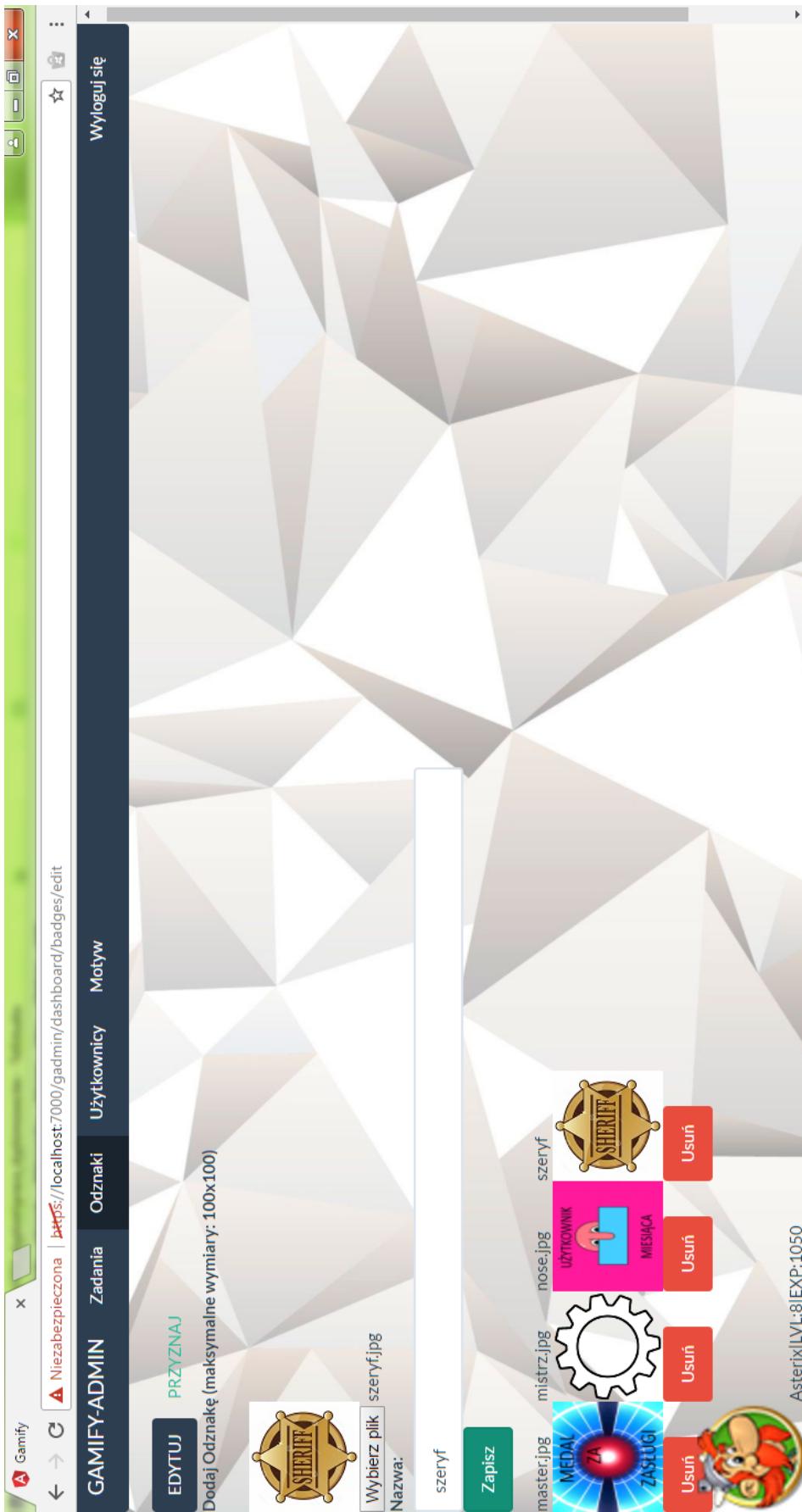
Rysunek 7.11: Panel zarządzania Użytkownikami



Rysunek 7.12: Panel edycji aplikacji



Rysunek 7.13: Panel zarządzania Zadaniami



Rysunek 7.14: Panel zarządzania Odznakami

GAMIFY-ADMIN

PRZYZNAJ

master.jpg	mistrz.jpg	nose.jpg	szeryf.jpg

PRZYZNAJ

Sztukaj...	Imię ▲	Nazwisko ▲	Opis ▲
Asterix	Jacek	Kozięja	1050 Programista. Geniusz. Wizjoner. Poliglota
PanPrezes	Jan	Nowak	40 Ja tu rządzę
Szymon112233	Szymon	Piórkowski	12 Ktos
Cartman	Piotr	Korgul	100 Valhalla
Dalton	Krzysztof	Pacholski	20 siedlfdf
Raptor	Krzysztof	Taperek	0 0 Mistrz cieni
PilnyStudent	Bogdan	Kowalski	20 Lubie rysować i jeść ziemniaki.
Fizyk	Albert	Einstein	10 Wszystko jest relatywne
Robak	Jacek	Soplica	0 0 Polska jest najważniejsza

PRZYZNAJ

Przyznaj

Odznaki zostaną przyznane

Rysunek 7.15: Panel dodawania Odznak

Rozdział 8

Wnioski

Zakres pracy został zrealizowany w zdecydowanej większości. Jedyne różnice względem początkowej wizji 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.
- Pominieto 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. Takie podejście opisane jest w rozdziale poświęconym budowie komponentowej w literaturze [15]. 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 szkieletie 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



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

lepsze wyniki w aplikacjach z których korzysta się dłużej, przemieszczając po wielu zakładkach. Aplikacji Angular nie 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 natknęto się na wiele błędów, których przyczyny okazały się trudne do odnalezienia. Szkielet Angular potrafił zwrócić błąd we wnętrzu 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 odnajdowanie 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 (ang. „You aren't gonna need it”) i KISS (ang. „Keep It Simple, Stupid”) - nie dodawać do projektu elementów niepotrzebnych i unikać przesadnej złożoności.

Podsumowując, można uznać, że cel pracy został osiągnięty. Spełnione zostały założenia dotyczące funkcjonalności, oraz architektury wielowarstwowej, narzucone przez użyte technologie. Aplikacja działa szybko, zaś dzięki wykorzystaniu szkieletu Angular, reakcje na działania Użytkowników następują błyskawicznie. Dalszy rozwój aplikacji jest możliwy i łatwy do przeprowadzenia.

```
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 < 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 (<anonymous>
```

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

Bibliografia

- [1] Google, „TUTORIAL: TOUR OF HEROES”
<https://angular.io/docs/ts/latest/tutorial/>
- [2] Pivotal Software, „Guides”
<https://spring.io/guides>
- [3] Tutorials Point, „Spring - Bean Definition”, „Spring - Java Based Configuration”
<http://www.tutorialspoint.com/spring/>
- [4] Apache Software Foundation, „Documentation”
<https://maven.apache.org/guides/>
- [5] Scott Urman, Ron Hardman, Michael McLaughlin, „Oracle Database 10g : programowanie w języku PL/SQL”, Helion, 2008
- [6] Oracle, „Indexes and Index-Organized Tables”, „Partitions, Views, and Other Schema Objects”
https://docs.oracle.com/cd/E11882_01/server.112/e40540/toc.htm
- [7] Wikipedia, „Grywalizacja”
<https://pl.wikipedia.org/wiki/Grywalizacja>
- [8] OWASP, „SQL Injection”, „Cross-Site Request Forgery (CSRF)”, „Insecure Transport”
<https://www.owasp.org>
- [9] Bruce Eckel, „Thinking in Java : edycja polska”, Helion, 2006
- [10] W. R. Stevens, G. R. Wright, „Biblia TCP/IP tom 1”, RM, 1998.
- [11] Michał Śmiałek, „Zrozumieć UML 2.0. Metody modelowania obiektowego”, Helion, 2005.

- [12] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt, „Pattern-oriented software architecture: On patterns and pattern languages Volume 5”, Wiley, 2007.
- [13] Kobiety do kodu, „Poznaj Javę dzięki naszym praktycznym lekcjom”
<http://kobietydokodu.pl/kurs-javy/>
- [14] Kutyłowski Mirosław, Strothmann Willy-B, „Kryptografia. Teoria i praktyka zabezpieczania systemów komputerowych”, ReadMe, 2000.
- [15] Ian Sommerville, „ Inżynieria oprogramowania”, WNT, 2011.
- [16] Rod Johnson , Juergen Hoeller i inni, „Introduction to the Spring Framework”, Wiley, 2004
<http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html>
- [17] Roy Fielding i inni, „Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”
<http://httpwg.org/specs/rfc7231.html>

Spis rysunków

1.1	Diagram klas w języku Java.	4
1.2	Diagram klas w języku TypeScript.	5
1.3	Języki, szkielety i systemy będące częścią projektu.	6
1.4	Zapytanie do API i odpowiedź. Taka komunikacja pozwala wykorzystać tą samą warstwę modelu w kilku różnych aplikacjach.	7
1.5	Modułowy podział projektu.	8
3.1	Fragment uruchomienia skompilowanego przykładu.	13
3.2	Zapytanie wysłane do przykładu.	13
3.3	Uproszczony cykl życia projektu Maven.	15
4.1	SQL Developer pozwala na kompleksowe zarządzanie bazą danych	30
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.	31
4.3	Liście B-drzewa przechowują rekordy, gałęzie usprawniają przeszukiwanie.	32
4.4	Mechanizmy partycjonowania i indeksowania można ze sobą łączyć.	33
6.1	Ekran tworzenia nowej bazy danych SQL na Azure.	42
6.2	Postawienie własnej maszyny wirtualnej okazało się być najłatwiejszym rozwiązaniem w przypadku opisywanego projektu.	43
7.1	Wypełniony formularz rejestracji	45
7.2	Błędnie wypełniony formularz rejestracji	46
7.3	Proces zmiany hasła	47
7.4	Ekran logowania	48
7.5	Strona główna aplikacji	49
7.6	Panel ustawień konta	51

7.7	Panel z listą Użytkowników	52
7.8	Panel wykonania Zadania	53
7.9	Panel osiągnięć Użytkownika	54
7.10	Ekran logowania Administratora	55
7.11	Panel zarządzania Użytkownikami	56
7.12	Panel edycji aplikacji	57
7.13	Panel zarządzania Zadaniami	58
7.14	Panel zarządzania Odznakami	59
7.15	Panel dodawania Odznak	60
8.1	Prędkość wczytywania aplikacji na pierwszy rzut oka nie z- chwyca.	62
8.2	Błąd biblioteki SystemJS tak naprawdę oznaczał problem z definicją ścieżki do biblioteki, której chciano użyć w projekcie.	64