POLITECHNIKA KRAKOWSKA

IM. TADEUSZA KOŚCIUSZKI

WYDZIAŁ FIZYKI MATEMATYKI I INFORMATYKI

KIERUNEK INFORMATYKA

PAWEŁ OPIELA

**MOBILNY SYSTEM ZARZĄDZANIA MAGAZYNEM ELEKTRONIKA**

**MOBILE ELECTRONICS WAREHOUSE MANAGEMENT SYSTEM**

PRACA MAGISTERSKA

STUDIA STACJONARNE

Promotor: Dr Agnieszka Krok

Kraków 2014

**Spis treści**

[1 Wstęp 4](#_Toc397377602)

[1.1 Definicje 4](#_Toc397377603)

[1.2 Przegląd istniejących rozwiązań 5](#_Toc397377604)

[1.3 Cel i zakres pracy 6](#_Toc397377605)

[2 Projekt wstępny 8](#_Toc397377606)

[2.1 Ogólny opis funkcjonalności 8](#_Toc397377607)

[2.1.1 Aplikacja internetowa 8](#_Toc397377608)

[2.1.2 Aplikacja mobilna 9](#_Toc397377609)

[2.2 Wymagania funkcjonalne 9](#_Toc397377610)

[2.2.1 Aplikacji internetowej 10](#_Toc397377611)

[2.2.2 Aplikacja mobilna 13](#_Toc397377612)

[2.3 Wymagania niefunkcjonalne 14](#_Toc397377613)

[2.4 Wybór technologii 15](#_Toc397377614)

[2.5 Planowanie pracy 16](#_Toc397377615)

[2.6 Komunikacja aplikacji mobilnej z serwerem 17](#_Toc397377616)

[3 Implementacja 20](#_Toc397377617)

[3.1 Aplikacja internetowa 20](#_Toc397377618)

[3.1.1 Konfiguracja środowiska 20](#_Toc397377619)

[3.1.2 Połączenie z bazą danych 22](#_Toc397377620)

[3.1.3 Autentykacja 24](#_Toc397377621)

[3.1.4 Tworzenie ról i konta administratora 26](#_Toc397377622)

[3.1.5 Model bazy danych 28](#_Toc397377623)

[3.1.6 Główny plik szablonu 29](#_Toc397377624)

[3.1.7 Sekcja administratora 30](#_Toc397377625)

[3.1.8 Sekcja użytkownika 30](#_Toc397377626)

[3.2 Komunikacja 30](#_Toc397377627)

[3.3 Aplikacja mobilna 30](#_Toc397377628)

[3.3.1 Konfiguracja środowiska 30](#_Toc397377629)

[4 Testowanie 31](#_Toc397377630)

[5 Wnioski 32](#_Toc397377631)

[6 Bibliografia 33](#_Toc397377632)

# Wstęp

Poważnym problem wśród osób zajmujących się elektroniką zarówno hobbystycznie, amatorsko jak i zawodowo jest brak należytej organizacji pracy poprzez uporządkowanie elementów elektronicznych. Przez co osoby te często same do końca nie wiedzą jakie elementy elektroniczne nazywane również komponentami posiadają w swoich zbiorach.   
Co więcej często zdarza się sytuacją, iż odszukanie konkretnego komponentu sprawia duże problemy. Niejednokrotnie zdarza się iż niezbędny jest tzw. "telefon do przyjaciela" w celu zapytania o poszukiwany element, co nie jest najbardziej efektywnym postępowaniem. Sprawa rażąco komplikuje się gdy różnych składników jest bardzo dużo w małych ilościach, większe ilości zazwyczaj pakowane w specjalnych rolkach przez co ich organizacja nie sprawia tyle kłopotów. W dalszej części zaprezentowany został projekt oraz proces implementacji systemu mobilnego służącego do zarządzania zbiorami elementów elektronicznych, zrealizowany na potrzeby pracy magisterskiej na kierunku informatyka   
na Politechnice Krakowskiej.

## Definicje

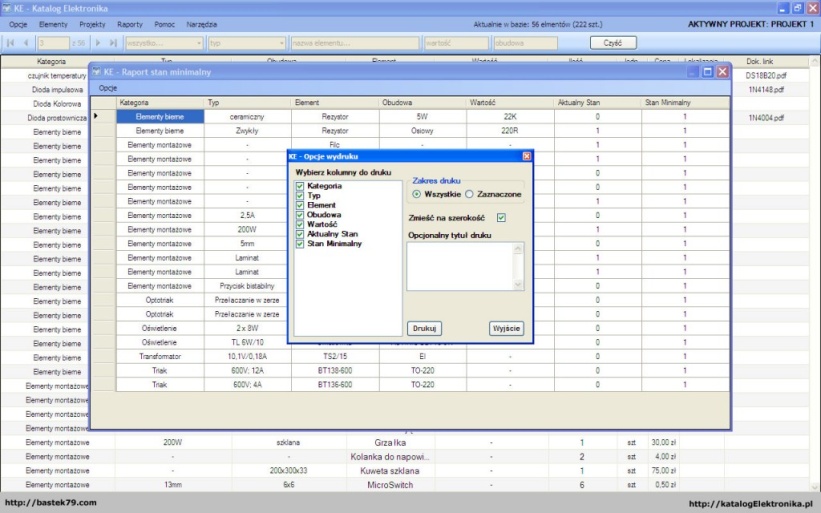
Poniżej przedstawiane zostały definicje słów kluczowych użytych w projekcie:

* Komponenty (elementy) - części elektroniczne, układy scalone itp.
* Magazyn - w rozumieniu projektu należy traktować, jako kontener, grupa dla komponentów. Fizycznie rozumieć należy jako miejsce gdzie znajdują się elementy przykładowo (Pudełko, Półka)
* Projekt jako część funkcjonalna aplikacji – oznacza urządzenie elektroniczne   
  w którego skład wchodzą elementy elektroniczne lub komponenty. Przykładowym projektem może być (Zegar z modułem GPS)

## Przegląd istniejących rozwiązań

Temat organizacji elementów elektronicznych nie jest czymś nowym, na rynku dostępne są różnego rodzaju programy przeznaczone do tego celu. Poniżej zostaną zaprezentowane konkurencyjne rozwiązania:

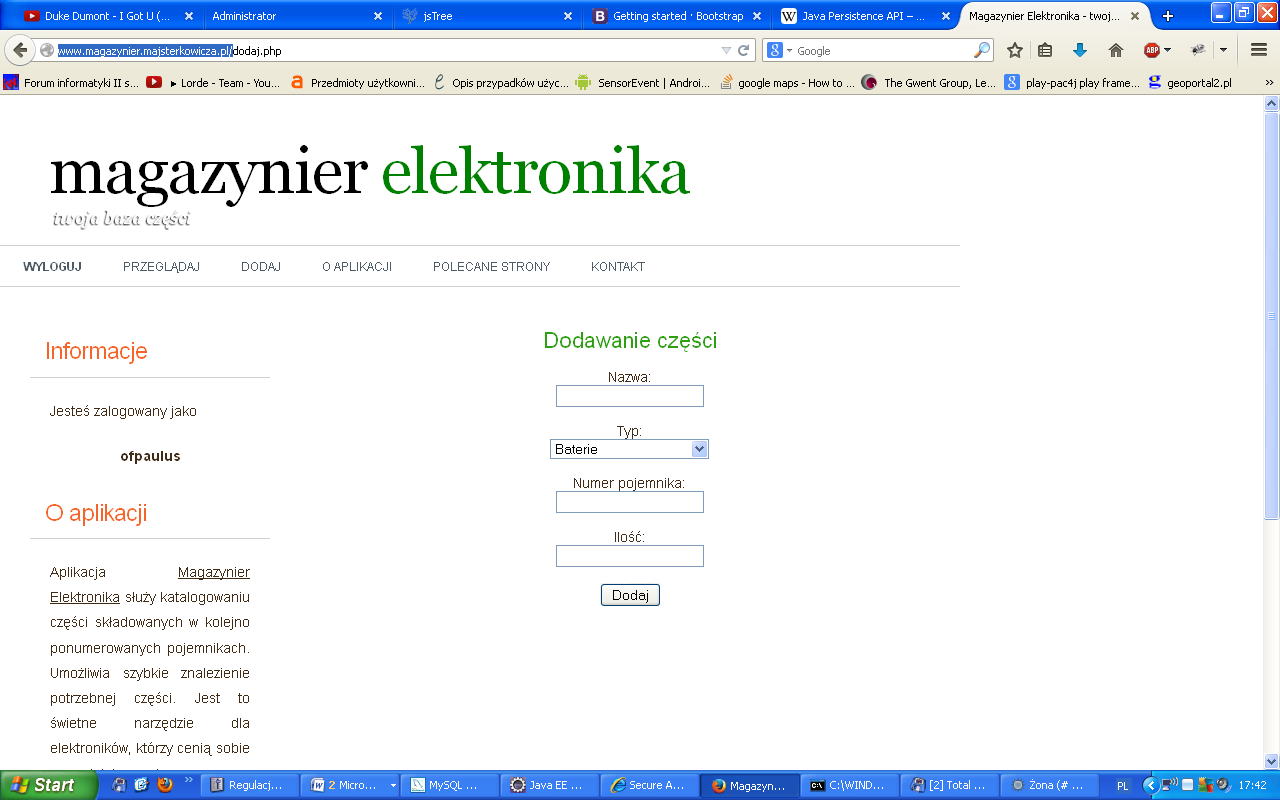
**Katalog elektronika** *http://katalogelektronika.pl/*



Rys 1.1  
Widok aplikacji katalog elektronika

Katalog elektronika jest programem typowo desktopowy, posiada lokalną bazę danych którą można eksportować oraz importować z i do pliku. Posiada moduł dodający elementy  
z plików produkcyjnych generowanych z programu Eagle (oprogramowanie do projektowania płytek drukowanych). Użytkownik dodając nowy element musi wypełnić formularz z dużą ilością pól. Możliwe jest także generowanie raportów ze stanami zerowymi lub minimalnymi.

**Magazynier elektronika** *http://www.magazynier.majsterkowicza.pl/*



Rys 1.2  
Widok strony projektu magazynier elektronika

Magazynier elektronika jest rozwiązaniem internetowym katalogowania zasobów, posiada bardzo prosty interfejs oraz podstawową funkcjonalność dodawania elementów   
oraz ich przeglądania.

## Cel i zakres pracy

Celem pracy jest budowa systemu służącego do funkcjonalnego oraz prostego zarządzania stanem magazynowym komponentów elektronicznych w postaci dwóch aplikacji internetowej oraz mobilnej. Analizując rynek aplikacji pod kątem podobnego istniejącego już rozwiązania należy stwierdzić iż dostępne rozwiązania są albo przesadnie rozbudowane  
i skomplikowane w użyciu lub proste ale ciężkie w użyciu ze względu na mało przyjazny projekt interfejsu użytkownika.

W prezentowanej pracy główny nacisk projektowy kierowany jest w stronę zapewnienia jak największej prostoty z zachowaniem funkcjonalności i niezawodności.

Zakres pracy obejmuje:

* Budowę aplikacji internetowej o podstawowej funkcjonalności:
  + Tworzeniem konta użytkownika,
  + Zmiany hasła,
  + Tworzenia magazynu,
  + Dodawania części do magazynu na podstawie faktury (pdf) lub ręcznie,
  + Zarządzaniem stanem magazynowym poprzez zwiększanie lub zmniejszanie ilości elementów,
  + Nadawanie prawa wglądu do magazynu innym zarejestrowanym użytkownikom systemu.
* Implementację aplikacji mobilnej dla systemu Android za pomocą której można:
  + Zalogować się do bazy serwisu,
  + Przeglądać stany magazynowe,
  + Przeglądać zdefiniowane projekty,
  + Zmniejszać lub zwiększać liczbę poszczególnych elementów,
  + Synchronizować stan magazynowy z aplikacją internetową.

# Projekt wstępny

Rozdział przedstawia opis systemu. Dokładnie sprecyzowaną od ogółu do szczegółu specyfikację dzięki której przybliżona zostanie wizja projektu.

## Ogólny opis funkcjonalności

W podrozdziałach zawarty został opis funkcjonalności aplikacji wchodzących w skład projektu. Jest to wstęp do zrozumienia koncepcji systemu oraz jego logiki.

### Aplikacja internetowa

Projektowany serwis internetowy dostarczać będzie funkcjonalności wyłącznie po udanym zalogowaniu użytkownika do systemu. Dlatego tez należy zaimplementować proces rejestracji, logowania oraz usuwania użytkownika. W systemie występować będą dwie grupy użytkowników administrator oraz zwykły użytkownik.

Rola administratora sprowadza się do uzupełniania listy predefiniowanych komponentów elektronicznych wraz z tworzeniem i przypisywaniem do elementu predefiniowanej kategorii. Należy nadmienić iż w systemie występować będzie globalna lista elementów elektronicznych oraz kategorii czyli typu elementu np. stabilizator, wzmacniacz operacyjny. Kategorie będą reprezentowane jako struktura drzewiasta, każdy element zdefiniowany w globalnej liście musi być przypisany do kategorii, jednakże kategoria nie musi zawierać żadnego elementu. Oprócz ręcznego dodawania elementów oraz kategorii administrator, będzie miał wgląd do listy elementów danego użytkownika lub użytkowników. Dzięki czemu w prosty sposób będzie mógł zaimportować elementy innego użytkownika do globalnej listy elementów. Import kategorii z repozytoriów użytkowników będzie wyglądał w analogiczny sposób jak import elementów.

Zwykły użytkownik aplikacji posiada swoje konto utworzone podczas rejestracji, po zalogowaniu użytkownik może spersonalizować swoje informacje takie jak hasło, imię nazwisko oraz wyświetlaną nazwę. Podstawową funkcjonalnością jest możliwość dodawania magazynów, które cechują się unikalna nazwą oraz dodatkowym opisem który jest opcjonalny. Każdy użytkownik posiada na stałe zdefiniowany magazyn o nazwie „Globalny” – w którym znajdują się elementy ze wszystkich magazynów danego użytkownika, jest to jedyna różnica w porównaniu z pozostałymi. Dodawanie elementu do magazynu polega   
na wprowadzeniu nazwy elementu, liczby sztuk oraz kategorii, przy czym kategoria jest opcjonalna. Przy wprowadzaniu nazwy elementu serwis powinien przeszukać globalną listę elementów oraz zwrócić listę propozycji, każda propozycja zawierać będzie również kategorię elementu, dzięki czemu użytkownik nie będzie musiał ręcznie wybierać kategorii. Domyślnie użytkownik nie posiada kategorii kategorie są sukcesywnie dodawane przez użytkownika wraz z dodawaniem elementów lub będą mogły zostać zaimportowane   
z globalnej listy kategorii. Kolejną funkcjonalnością będzie możliwość definiowania projektów, do każdego z nich będzie możliwość przypisania odpowiedniej ilość wybranych elementów. Oprócz ręcznego dodawania elementów będzie możliwość automatycznego dodania elementów przykładowo z pliku faktury zapisanej w formacie PDF. Wybierając projekt użytkownik musi posiadać opcję odlicz elementy z bazy magazynów, dodatkowo powinna zostać zaprezentowana liczba sztuk jaką można wyprodukować z bieżącego stanu magazynowego.

### Aplikacja mobilna

Funkcjonalność aplikacji mobilnej dla systemu Android, będzie pozbawiona możliwości tworzenia magazynów, dodawania elementów oraz definiowania projektów jak to miało miejsce w aplikacji internetowej. Głównym cele istnienia aplikacji jest możliwość przeglądania magazynów oraz zmiany liczebności elementów oraz prezentacji listy składowej projektów. Aplikacja mobilna po uruchomienie będzie prosić o podanie adresu email  
oraz hasła używanego w serwisie internetowym. Poprawne wprowadzenie danych skutkować będzie pobraniem z serwisu bazy elementów. Zmiana liczebności elementów przez aplikację zapisana zostaje lokalnie. W celu przesłania zmian do aplikacji internetowej użytkownik będzie musiał wybrać opcje synchronizuj bazę.

## Wymagania funkcjonalne

Rozdział przedstawia szczegółowy podział funkcjonalności, zawiera rozwinięte opisy relacji systemu z użytkownikiem które należy uwzględnić w procesie implementacji rozwiązań.

Opisy poszczególnych funkcjonalności zawarte zostały w postaci opisowych przypadków użycia zawartych w tabelach poniżej, które zawierają nazwę funkcjonalności, opis, definicję danych wejściowych w tym także ograniczeń, zwracany oczekiwany rezultat, uwagi w tym zachowanie podczas nie powodzenia, oraz rola użytkownika.

### Aplikacji internetowej

|  |  |
| --- | --- |
| Nazwa | Utworzenie konta administratora |
| Opis | Utworzenie domyślnego konta administratora |
| Wejście | Email oraz hasło zapisane w pliku konfiguracyjnym systemu |
| Wyjście | Utworzone konto administratora podczas instalacji systemu |
| Uwagi | Plik konfiguracyjny musi być zabezpieczony przed dostępem zdalnym |
| Rola | Administrator |

|  |  |
| --- | --- |
| Nazwa | Dodanie elementu do globalnej listy elementów |
| Opis | Dodanie elementu wraz z podaną kategorią.  Kategorie reprezentowane jako struktura hierarchiczna. |
| Wejście | Nazwa elementu oraz kategoria |
| Wyjście | Dodany element do listy globalnej oraz utworzenie kategorii w globalnej liście kategorii. |
| Uwagi |  |
| Rola | Administrator |

|  |  |
| --- | --- |
| Nazwa | Dodanie elementów do globalnej listy elementów z użyciem pomocniczej listy komponentów zdefiniowanych przez użytkownika lub użytkowników. |
| Opis | System wyświetla listę elementów wybranych użytkowników, elementu wybrane przez administratora dodawane są do listy globalnych elementów, wraz z elementami można zaakceptować kategorie wybraną przez użytkownika lub wybrać inną |
| Wejście | Lista elementów wraz z kategoriami |
| Wyjście | Dodany elementów do bazy danych |
| Uwagi |  |
| Rola | Administrator |

|  |  |
| --- | --- |
| Nazwa | Rejestracja użytkownika |
| Opis | Rejestracja nowego użytkownika systemu |
| Wejście | Adres email który jednoznacznie identyfikuje użytkownika, osobiste hasło nie krótsze niż 5 znaków, opcjonalnie: imię i nazwisko |
| Wyjście | Użytkownik zostaje dodany do bazy danych systemu po czym na podany adres email wysyłany jest link aktywujący konto. |
| Uwagi | Adres email jest unikatowy i przypisany do użytkownika, nie można zarejestrować dwa razy tego samego adresu email. |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Resetowanie hasła użytkownika |
| Opis | Użytkownik zapomniał hasła |
| Wejście | Adres email |
| Wyjście | Na podany adres email wysyłany zostaje link do strony z formularzem służącym do wprowadzenia nowego hasła |
| Uwagi |  |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Tworzenie magazynu |
| Opis | Użytkownik tworzy nowy magazyn |
| Wejście | Nazwa magazynu oraz opcjonalny opis |
| Wyjście | Dodany wpis w bazie danych |
| Uwagi | Nazwa magazynu musi być unikatowa |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Dodanie komponentu elektronicznego do magazynu |
| Opis | Dodanie nowego elementu do magazynu.  Jeżeli nazwa elementu występuje już w bazie elementów użytkownika, należy przypisać element do wybranego magazynu wraz z podaną liczbą elementów.  W przypadku gdy element o podanej nazwie występuje już w danym magazynie należy zaktualizować liczbę elementów w magazynie sumując ją z podaną przez użytkownika. |
| Wejście | Identyfikator wybranego magazynu, nazwa komponentu, liczebność elementów, kategoria elementu |
| Wyjście | Dodany wpis w bazie danych wybranego magazynu opisujący element oraz jego liczebność i kategorię do której został przypisany. |
| Uwagi | Jeżeli użytkownik wprowadził nazwę elementu występująca już w bazie danych użytkownika, nie ma możliwości zmiany kategorii już zdefiniowanego elementu w bazie danych. |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Usuwanie magazynu |
| Opis | Użytkownik może usunąć magazyn, w takim wypadku jeżeli zawiera on elementy, należy wyświetlić komunikat z opcjami:   * Usunąć wszystkie elementy z magazynu, * Przenieś do innego wskazanego magazynu, * Przenieś do magazynu domyślnego (Globalny). |
| Wejście | Identyfikator magazynu, typ akcji usuwania lub przenoszenia elementów do innego magazynu. |
| Wyjście | Usunięty magazyn z bazy danych, w zależności od wybranego typu akcji przeniesienie lub usunięcie elementów magazynu. |
| Uwagi | W przypadku gdy usuwany magazyn zawiera komponenty które zdefiniowane są w projekcie lub są zdefiniowane w innym magazynie opcja usuń wszystkie usunie tylko liczebność elementów z usuwanego magazynu. |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Import elementów z faktury PDF |
| Opis | Użytkownik wysyła do systemu plik z fakturą PDF poprzez odpowiedni formularz wybiera odpowiedni typ pliku czyli rodzaj predefiniowanego sklepu lub hurtowni dla której system może przetworzyć plik. |
| Wejście | Plik PDF, typ analizatora |
| Wyjście | Lista elementów zawartych w analizowanym pliku |
| Uwagi | Każdy sklep, hurtowania definiuje fakturę w odrębny sposób, dlatego też dla każdego przypadku należy zdefiniować analizator. |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Definiowanie projektu |
| Opis | Użytkownik dodaje projekt elektroniczny |
| Wejście | Nazwa projektu |
| Wyjście | Dodany wpis w bazie |
| Uwagi | Unikatowa nazwa projektu |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Dodawanie elementów do projektu |
| Opis | Użytkownik dodaje elementy do projektu, elementy muszą być zdefiniowane w bazie danych użytkownika, |
| Wejście | Nazwa elementu, liczebność elementu w projekcie |
| Wyjście | Dodany wpis w bazie danych połączony z projektem |
| Uwagi |  |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Usuwanie projektu |
| Opis | Użytkownik usuwa projekt |
| Wejście | Identyfikator projektu |
| Wyjście | Usunięty wpis w bazie danych |
| Uwagi |  |
| Rola | Użytkownik |

### Aplikacja mobilna

|  |  |
| --- | --- |
| Nazwa | Logowanie do systemu |
| Opis | Logowanie użytkownika z poziomu aplikacji mobilnej do aplikacji internetowej |
| Wejście | Email oraz hasło |
| Wyjście | Autoryzacja użytkownika w aplikacji mobilnej, pobranie z serwera stanu magazynowego oraz listy projektów i zapisuje w lokalnej bazie danych |
| Uwagi |  |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Zmiana liczebności elementu w wybranym magazynie |
| Opis | Użytkownik zmienia liczebności elementu, zmiana odbywa się lokalnie |
| Wejście | Identyfikator elementu, ilość elementu |
| Wyjście | Lokalna zmiana liczebności elementu w bazie danych |
| Uwagi |  |
| Rola | Użytkownik |

|  |  |
| --- | --- |
| Nazwa | Synchronizacja bazy danych |
| Opis | Jeżeli użytkownik jest autoryzowany może dokonać synchronizacji bazy danych z serwerem, jeżeli nie jest zalogowany musi wprowadzić swoje dane. |
| Wejście | Polecenie synchronizacji bazy danych |
| Wyjście | Aplikacja wysyła do serwera listę elementów edytowanych, po czym pobiera listę zmian dokonanych w systemie. |
| Uwagi | Podczas synchronizacji brane pod uwagę są tylko zmiany oznacza to iż dla każdego elementu definiowana jest liczba dodatnia lub ujemna definiująca zmianę od ostatniej synchronizacji, serwer akceptuje zmiany liczebności. |
| Rola | Użytkownik |

## Wymagania niefunkcjonalne

Ogólny interfejs użytkownika systemu musi być prosty, bez zbędnych grafik jaskrawych kolorów utrzymany w łagodnym gradiencie barw. Komunikaty powinny być krótkie i zrozumiałe. Aplikacja internetowa powinna wspierać widok responsywny (dostosowywać się do wymiarów okna przeglądarki), być zgodna ze standardem HTML 5.

Zarówno aplikacja internetowa oraz mobilna muszą zapewniać bezpieczeństwo danych przechowywanych w bazie danych oraz ich poufność.

Względem aplikacji internetowej:

* Aplikacja musi mieć możliwość uruchomienia na dowolnym serwerze wyposażonym w wirtualna maszynę Javy,
* Musi być zapewniona możliwość przyszłego skalowania aplikacji na serwerze.

Względem aplikacji mobilnej:

* Minimalna wersja systemu Android na urządzeniu mobilnym 4.0.

## Wybór technologii

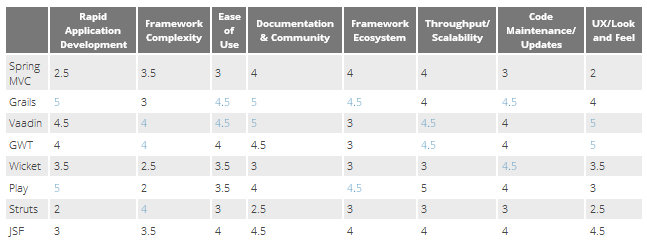
Znając dokładnie wymagania stawiane projektowi w poniższym rozdziale przedstawione zostały aspekty wyboru technologii systemu.

W przypadku aplikacji mobilnej najlepszym wyborem jest użycie standardowego rozwiązania dedykowanego przez dostawce systemu, czyli użycie języka Java z Android SDK.

Wybór rozwiązania dla aplikacji internetowej nie jest tak prosty i oczywisty jak  
 w przypadku aplikacji mobilnej. Jeśli chodzi o wybór języka programowania to musiał być on dokonany pomiędzy dwoma platformami: C# lub Java. W ostateczności wybór podyktowany został zachowaniem kompatybilności języka z aplikacją mobilną. Dla wybranego języka programowania dostępna jest duża liczba bibliotek wspomagających, które z powodzeniem można wykorzystać do realizacji projektu. Na podstawie syntezy wymagań funkcjonalnych i niefunkcjonalnych zostały zdefiniowane kryteria, które determinowały wybór technologii:

* Małe wymagania względem zasobów maszyny na której uruchomiona jest aplikacja.
* Szybkość działania,
* Skalowalność (należy brać pod uwagę scenariusz w którym serwis musi obsługiwanych coraz to większa liczbę użytkowników w czasie),

Kierując się doborem technologii warto wykorzystać opinie użytkowników.   
W Internecie można znaleźć dużą liczbę dyskusji przedstawiających wady oraz zalety poszczególnych rozwiązań. Poniżej (Rys 2.1) przedstawiona została tabela przedstawiająca oceny poszczególnych technologii pod kątem różnych kategorii. Zestawienie zostało przygotowane na podstawie przeprowadzonej sondy w grupie ponad 1800 deweloperów. Artykuł opisujący porównanie został opublikowany przez Simon’a Maple.



Rys 2.1   
Podsumowanie porównania framework’ów   
źródło : http://zeroturnaround.com/rebellabs/the-curious-coders-java-web-frameworks-comparison-spring-mvc-grails-vaadin-gwt-wicket-play-struts-and-jsf/10/

Przytoczony artykuł prezentował całe spektrum kategorii ocenionych w skali od 1 do 5. Dzięki przeprowadzonej analizie dla prezentowanej pracy wybrano Play Framework. Słabe oceny w kategorii łatwość użycia oraz stopień skomplikowania wynikały głownie   
z faktu, iż rozwiązanie używa języka Scala dla systemu szablonów. Dodatkowo aplikacja nie musi być pisana w języku Java, można zamiennie stosować język Scala, gdyż kompilowany jest on do postaci binarnej Javy. Użycie Scali wcale nie dyskryminuje technologii, wręcz przeciwnie zachęca do poznania nowego języka. Dodatkowo wybrane rozwiązanie udostępnia gotowy serwer Play który automatycznie kompiluje wykryte zmiany w plikach źródłowych. W kategorii skalowalności wynik maksymalny 5 punktów został uzyskany dzięki implementacji Akka[[1]](#footnote-1) na której oparty system przetwarzania zapytań Play Framework’a.

## Planowanie pracy

Planowanie pracy oraz jego przebiegu można zdefiniować poprzez określenie kamieni milowych projektu, dzięki którym praca nad realizowanym przedsięwzięciem zostaje uporządkowana w logiczną całość. Poniższe kolejne etapy zostaną szerzej opisane   
w kolejnych rozdziałach.

* Konfiguracja środowiska oraz narzędzi developerskich,
* Utworzenie i konfiguracja pustego projektu aplikacji internetowej z użyciem Play Framework,
* Połączenie z serwerem bazy danych w tym projekcie MySQL,
* Wybór modułu autoryzacji oraz autentykacji użytkowników, adaptacja ról użytkowników,
* Implementacja modeli oraz logiki biznesowej,
* Implementacja pozostałych funkcjonalności,
* Utworzenie pustego projektu dla aplikacji mobilnej dla systemu Android
* Implementacja mechanizmu autoryzacji aplikacji mobilnej do serwisem internetowego
* Utworzenie lokalnej bazy danych, implementacja synchronizacji pomiędzy aplikacjami.

## Komunikacja aplikacji mobilnej z serwerem

Do poprawnej i osiągalnej komunikacji aplikacji z serwerem zostanie zdefiniowana usługa realizująca zadania związane z synchronizacją danych wysyłanych z aplikacji mobilnej komunikacja odbywać się będzie za pomocą obiektów zapisanych w formacie JSON.

Proces autentykacji użytkownika w aplikacji mobilnej odbywa sie poprzez wprowadzenie adresu email oraz hasła, które zostaną wysłane metodą POST do serwera, serwer zwróci wynik w postaci tokenu czyli tymczasowego identyfikatora użytkownika   
w systemie oraz pole „name” składające się z imienia i nazwiska lub puste jeżeli nie wprowadzono informacji o użytkowniku.

Przykład odpowiedzi:

[{"name":"Paweł Opiela","token":"ea8c6d148aa88acf87529fab535aa00c"}]

Jeżeli wprowadzone dane będą niepoprawne wartość pola token będzie pusta :

[{"token":""}]

Proces pobierania danych z serwera odbywać będzie się automatycznie po udanym procesie logowania do systemu z jedynym wyjątkiem gdy zmiany nie zostały zsynchronizowane lub są aktualne. W przypadku gdy dane w bazie danych użytkownika na serwerze uległy zmianie w porównaniu z ostatnio pobraną bazą danych do aplikacji mobilnej która dodatkowo została edytowana użytkownik zostanie poinformowany o nieaktualnej bazie danych.

Informacja na temat zmiany w bazie danych w aplikacji mobilnej zapisywana jest za pomocą funkcji skrótu MD5 generowanego przez serwer z przesyłanej bazy danych, skrót zapisywany jest w aplikacji podczas udanego importu bazy z serwera.

Aby nie pobierać niepotrzebnie bazy danych która jest aktualna, do serwera kierowane jest żądanie o przesłanie aktualnego skrótu bazy danych. Jeżeli skróty bazy danych na serwerze i aplikacji są różne aplikacja wysyła żądanie o przesłanie bazy danych.

Przykładowa baza danych zapisana w formacie JSON:

[{

"components":[{"id": 5, "name": "Atmega 8"}],

"warehouses":[{"id": 2, "name": "Półka A2"}],

"warehouse\_component":[{"id":12, "warehouse\_id":2,  
 "component\_id":5, "count":20}],

"projects":[{"id": 4, "name": "Dcf77"}],

"project\_component":[{"id"34, "project\_id":4,   
 "component\_id":5, "count":1}]

}]

W skład przykładowej bazy wchodzi:

* komponent (components) o nazwie „Atmega 8” o identyfikatorze 5,
* magazyn (warehouses) „Półka A2” o identyfikatorze 2,
* powiązanie (warehouse\_component) magazynu o identyfikatorze 2 i komponentu o identyfikatorze 5 w liczebności 20 sztuk,
* projekt (projects) „Dcf77” o identyfikatorze 4,
* powiązanie (project\_component) projektu o identyfikatorze 4 i komponentu  
  o identyfikatorze 5 w liczebności 1 sztuki.

Po pomyślnym imporcie bazy danych można odłączyć połączenie internetowe, każda zmiana wartości zapisywana jest lokalnie w bazie danych przykładowo zwiększając liczbę elementu  
o 5 sztuk i wybierając synchronizację z serwerem do serwera zostanie wysłany JSON jak poniżej:

[{"token":"ea8c6d148aa88acf87529fab535aa00c", "changes":[{"id": 5,  
 "change":4}]}]

W przypadku zmiany liczebności komponentu o identyfikatorze 5 przykładowo odliczono  
z magazynu 3 sztuki JSON wysyłany do serwera w polu *change* zawiera ujemną liczbę.

[{"token":"ea8c6d148aa88acf87529fab535aa00c", "changes":[{"id": 5,   
 "change":-3}]}]

Każdorazowo wysyłany jest także token uwierzytelniający w przypadku gdy token wygasł lub jest niepoprawny serwer zwróci wynik:

[{"token":""}]

W przypadku powodzenia zostanie przesłana zaktualizowana baza danych którą aplikacja zapisze w swojej lokalnej bazie, zapisze również skrót bazy danych.

# Implementacja

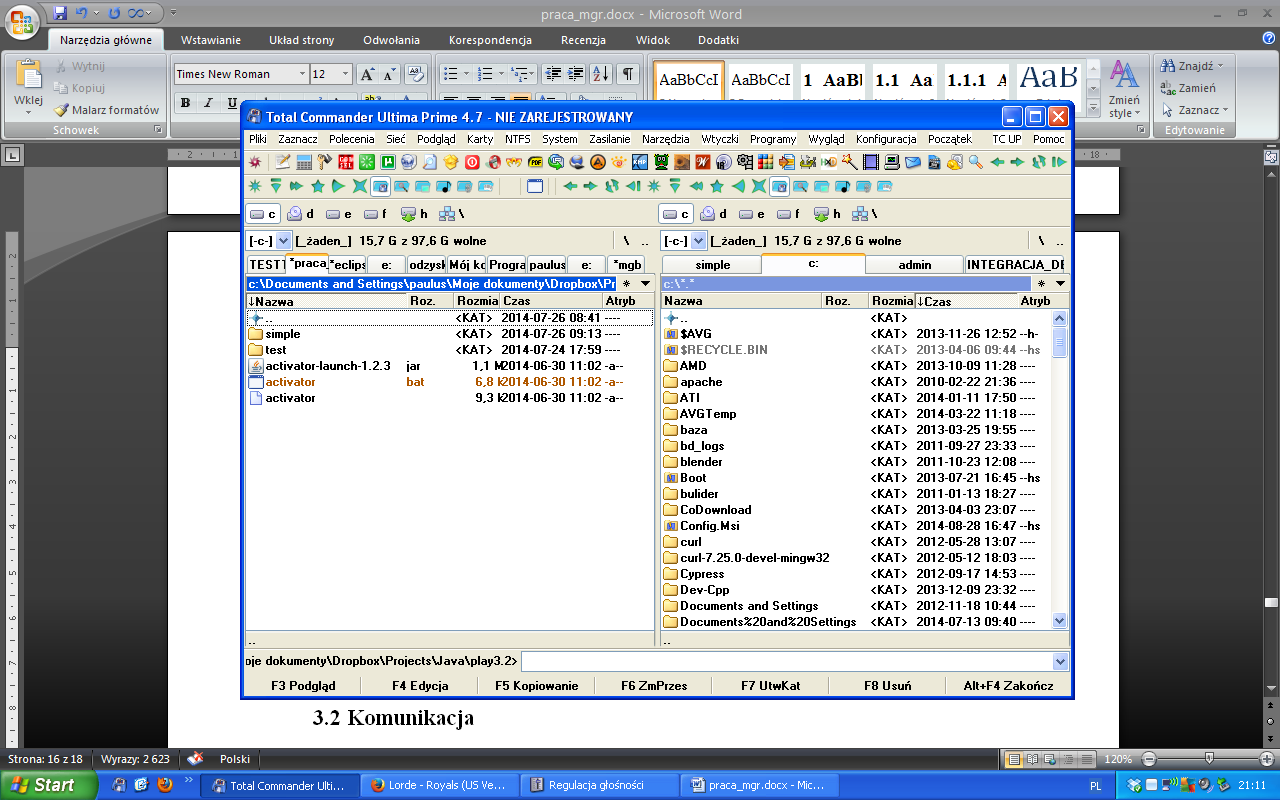
## Aplikacja internetowa

### Konfiguracja środowiska

Konfiguracja środowiska dla Play Framework sprowadza się do kliku kroków. Do tworzenie nowego projektu nowego projektu wykorzystywana jest platforma Typesafe[[2]](#footnote-2). Która zawiera zestaw predefiniowanych szablonów przykładowych aplikacji w różnych konfiguracja, przykładowo Play Framework skonfigurowany do pracy z bazą danych MongoDB oraz wiele innych.

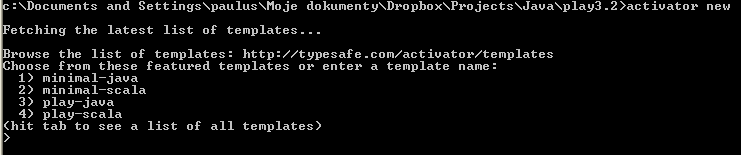
W celu utworzenia nowego projektu z użyciem platformy Typesafe należy:

* pobrać archiwum z programem *typesafe activator* w wersji mini dostępnej do pobrania na stronie *http://www.typesafe.com*, paczka po rozpakowaniu zawiera 3 pliki z czego 2 z nich są skryptami uruchomieniowymi dla systemów Windows oraz Linux,



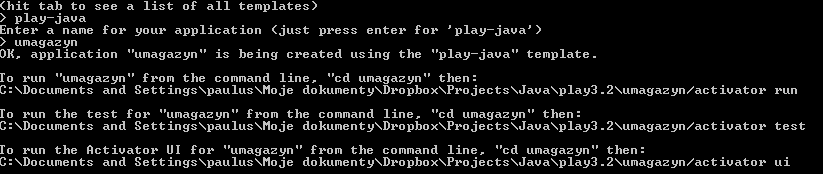
Rys .  
Widok rozpakowanego archiwum typesafe-activator-1.2.10-minimal.zip

* uruchomić konsolę systemową i wywołać skrypt ***activator.bat*** z parametrem ***new***,



Rys .  
Tworzenie nowego projektu

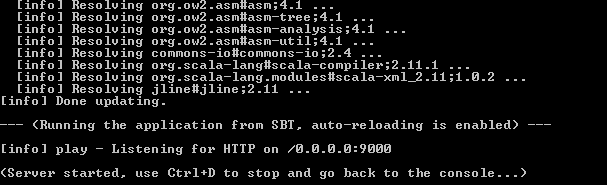
* wybrać interesujący szablon domyślnie widnieją 4 propozycje. Pełną listę szablonów można przeglądnąć po naciśnięciu przycisku tab, aktualnie na dzień 12.08.2014 lista zawiera 174 szablony. W przypadku projektu wybrano *play-java*,
* wprowadzić nazwę projektu **umagazyn**.



Rys .  
Utworzony nowy projekt

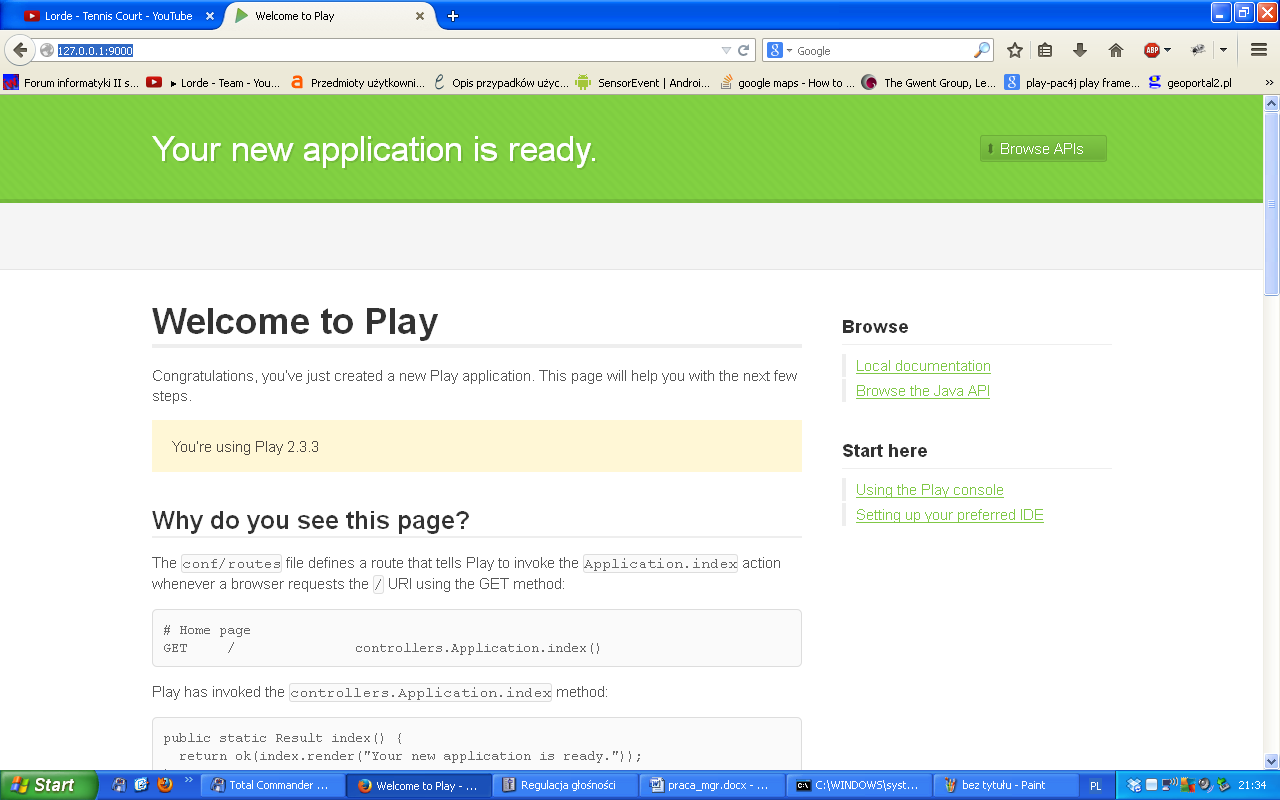
Aplikacja *activator* utworzyła podstawową strukturę plików i folderów projektu.

W celu uruchomienia projektu w nowo utworzonym folderze umagazyn, należy uruchomić skrypt ***activator.bat*** z parametrem ***run.*** Rezultatem uruchomionego skryptu jest rozpoczęcie pobierania różnych modułów dociągających zależności ze zdalnych repozytoriów, cała operacja kończy się komunikatem, iż serwer został uruchomiony.



Rys .  
Pobieranie zakończone, serwer uruchomiony

Nowo utworzony projekt jest skonfigurowany i gotowy do użycia, aby zobaczyć przykładową domyślną stronę nowego projektu w przeglądarce internetowej należy wprowadzić adres *http://127.0.0.1:9000/.*



Rys .  
Domyślna strona nowo utworzonego projektu

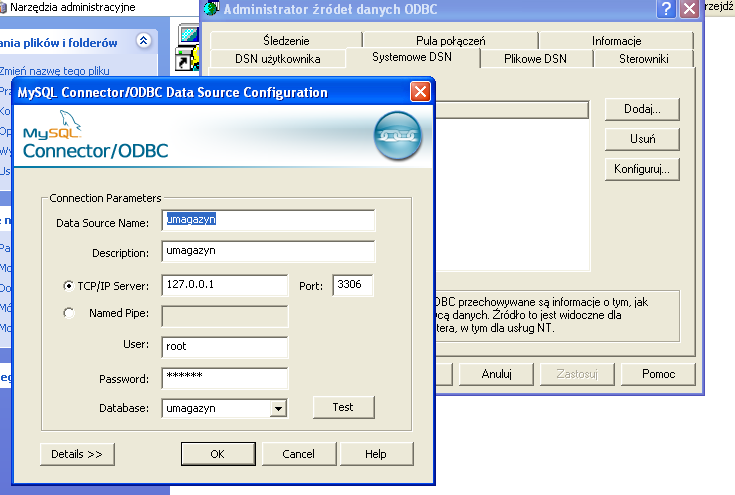
Activator posiada również swój własny edytor uruchamiany poleceniem ***activator ui***. Projektanci nie zapomnieli również o możliwości używania innych środowisk takich jak Eclipse lub IntelliJ. Utworzenie projektu dla środowiska Eclipse z projektu umagazyn sprowadza się do uruchomienia polecenia **activator** **eclipse**.

### Połączenie z bazą danych

Dostęp do bazy danych zapewniony jest domyślnie po uruchomieniu aplikacji, dzięki bazie H2 działającej w pamięci, jest to znakomite baza do różnego rodzaju testów. Jednakże jest to tyko baza tymczasowa zdefiniowana w pamięci operacyjnej na czas działania serwera Play.

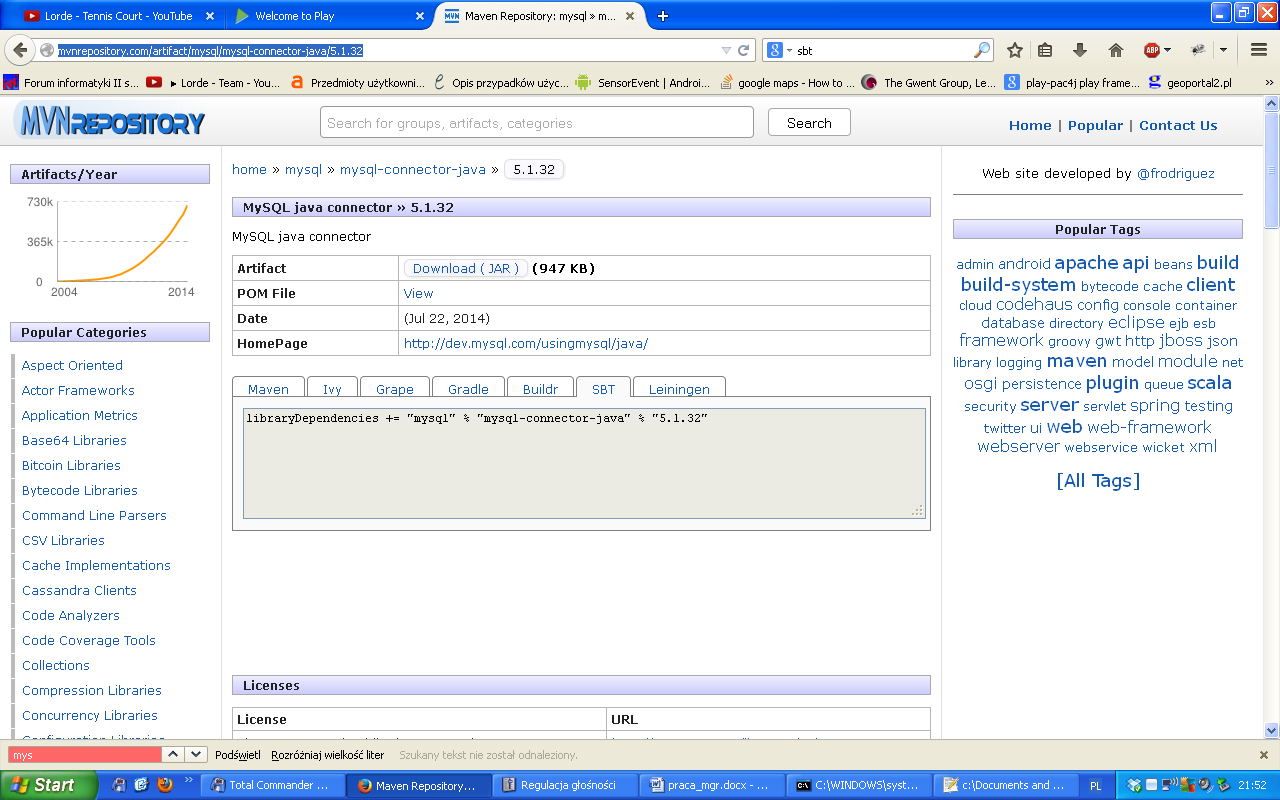
W pracy wykorzystano bazę MySQL która wcześniej została zainstalowana   
w systemie. Dla projektu utworzona została baza o nazwie „umagazyn” z ustawionym kodowaniem znaków na utf-8. Do połączenia z bazą MySQL z poziomu framewrok’a należało utworzyć logiczną nazwę bazy w systemowej usłudze ODBC, aby jednak móc dodać połączenie do bazy MySQL pobrano i zainstalowano w systemie sterownik Connector/ODBC ze strony *http://dev.mysql.com/downloads/connector/odbc/*.

Kolejno dodano nowy wpis DSN (nazwa źródła danych) w usłudze ODBC .



Rys .  
Okno definiowania źródła danych

Play Framework posiada wbudowane interaktywne narzędzie budujące (sbt - *http://www.scala-sbt.org/)* oparte na języku scala. Dzięki temu rozwiązaniu aby dodać moduł do projektu wystarczy edytować plik konfiguracyjny build.sbt i w sekcji libraryDependencies dopisać interesujący moduł. Aby odnaleźć interesujący moduł można skorzystać ze strony Maven repository [*http://mvnrepository.com*](http://mvnrepository.com), która zawiera moduły dla Maven (automatyczne narzędzie budujące dla projektów Java) jednakże dostępne są także wpisy dla innych narzędzi budujących w tym także dla sbt.



Rys .  
Okno serwisu Maven repository  
http://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.32

W sekcji libraryDependencies dodano wpis:

"mysql" % "mysql-connector-java" % "5.1.32"

name := """umagazyn"""

version := "1.0-SNAPSHOT"

lazy val root = (project in file(".")).enablePlugins(PlayJava)

scalaVersion := "2.11.1"

libraryDependencies ++= Seq(

"mysql" % "mysql-connector-java" % "5.1.32",

javaJdbc,

javaEbean,

cache,

javaWs

)

Po uruchomieniu aplikacji ***activator*** ***~run*** zostanie automatycznie pobrany moduł *mysql*. Znak **~** użyty przed parametrem *run* oznacza kompilację plików źródłowych po wykryciu zmian w systemie plików projektu. Aby skonfigurować dostęp do bazy danych należy edytować plik *conf\application.conf* w którym trzeba uzupełnić sekcję *db.default.\** poniższymi wartościami:

db.default.driver=com.mysql.jdbc.Driver

db.default.url="jdbc:mysql://localhost/umagazyn?characterEncoding=UTF-8"

db.default.user=root

db.default.password="xxxxx"

Oprócz ustawienia parametrów dostępu do bazy danych w tym samym pliku konfiguracyjnym odkomentowano poniższą linię poprzez usunięcie znaku # z początku linii.

ebean.default="models.\*"

Jest to konfiguracja modułu Ebean służącego do mapowania relacyjno obiektowego, jest to domyślne rozwiązanie stosowane w Play Framewrok jednakże nic nie stoi na przeszkodzie aby użyć innych modułów przykładowo Hibernate.

### Autentykacja

Dlaczego warto stosować gotowe moduły autentykacji i nie implementować własnego rozwiązania? Główną zaletą stosowania istniejących rozwiązań jest bezpieczeństwo, większość rozwiązań jest publikowana w formie Open Source, dzięki temu jeżeli jakikolwiek użytkownik znajdzie błąd lub lukę natychmiast zgłosi błąd lub/i zaproponuje rozwiązanie, zrobi poprawkę.

Przykładowymi plugin’ami służącymi do zarządzania kontami użytkowników w Play Framework są:

* SecureSocial - *http://securesocial.ws/*
* Play2-Auth - *https://github.com/t2v/play2-auth*
* Play! Authenticate - *http://joscha.github.io/play-authenticate/*

Najbardziej interesującym ciekawym rozwiązaniem okazał się Play! Authenticate, ponieważ oprócz spełnionych wymagań znajdujących się w sekcji wymagań funkcjonalnych, rozwiązanie to zawiera możliwość logowania za pomocą portali społecznościowych   
np. Facebook, Twitter. Dodatkowo zawiera wbudowany moduł deadbolt służący do autoryzacji po stronie szablonu. Aby zabezpieczyć dowolną akcję po stronie kontrolera przed nie autoryzowanym dostępem wystarczy dodać prostą adnotację przed akcją:

@Restrict(@Group("rola"))

Gdzie rola oznacza zdefiniowaną systemie grupę użytkowników. W przypadku szablonów widoku, aby zdefiniować blok wyświetlany wyłącznie dla autoryzowanych użytkowników wystarczy dodać:

@subjectPresentOr() {

Użytkownik zalogowany w systemie

}

Aby uściślić autoryzacje do konkretnej roli użytkownika można posłużyć się blokiem:

@restrict( scala.collection.immutable.List( as( "xxx" ) ) ) {

Autoryzowany użytkownik ze zdefiniowaną rolą xxx

}

Przykładowa aplikacja zawierającą skonfigurowany plugin została pobrana i uruchomiona. Następnie zostały przeniesione pliki źródłowe z przykładowej aplikacji do czystego projektu umagazyn. W celu szczegółowego zapoznania się z architekturą, każdy plik został przeanalizowany pod względem funkcjonalności oraz zastosowanych rozwiązań.

Plik *app\Global.java* zawiera definicje klasy *Global* dziedziczącej po klasie *GlobalSettings* która zawiera globalne ustawienia takie jak zdarzenia startu, stopu, błędu (*onStart*, *onStop*, *onError*). Prezentowany plugin w zdarzeniu *onStart* zawiera definicje akcji występujących podczas zdarzeń związanego z autentykacją użytkownika, przykładowe akcje:

* login – okno logowania, przykładowo użytkownik nie jest zalogowany do systemu, wprowadza adres URL do zasobu który wymaga autoryzacji. W tym wypadku jeżeli akcja byłą zabezpieczona, przykładowo adnotacją @Restrict(@Group("rola")) plugin automatycznie przekieruje żądanie na stronę logowania,
* afterAuth – definicja pierwszej strony po udanym logowaniu,
* afterLogout – definicja strony wyświetlonej po wylogowaniu użytkownika,

PlayAuthenticate.setResolver(new Resolver() {

@Override

public Call login() {

return routes.Application.login();

}

@Override

public Call afterAuth() {

return routes.Application.index();

}

@Override

public Call afterLogout() {

return routes.Application.index();

}

…

});

Odwzorowanie adresu URL na akcję konkretnego kontrolera zawarte jest w pliku conf\routes kolejne kolumny przedstawiają typ żądania HTTP, adres URL oraz akcję kontrolera. Przykładowe mapowania dla adresu domyślnego, logowania oraz akcji z formularza logowania:

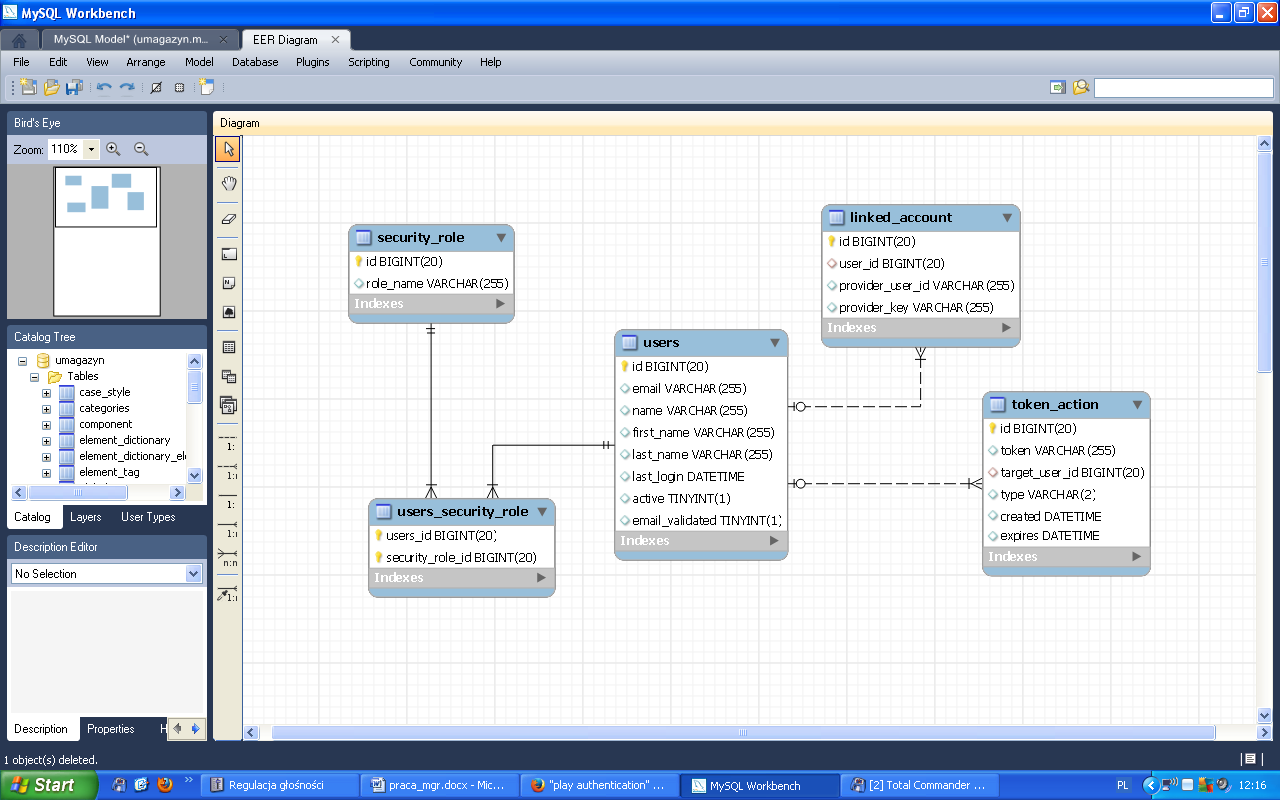
GET / controllers.Application.index

GET /login controllers.Application.login

POST /login controllers.Application.doLogin

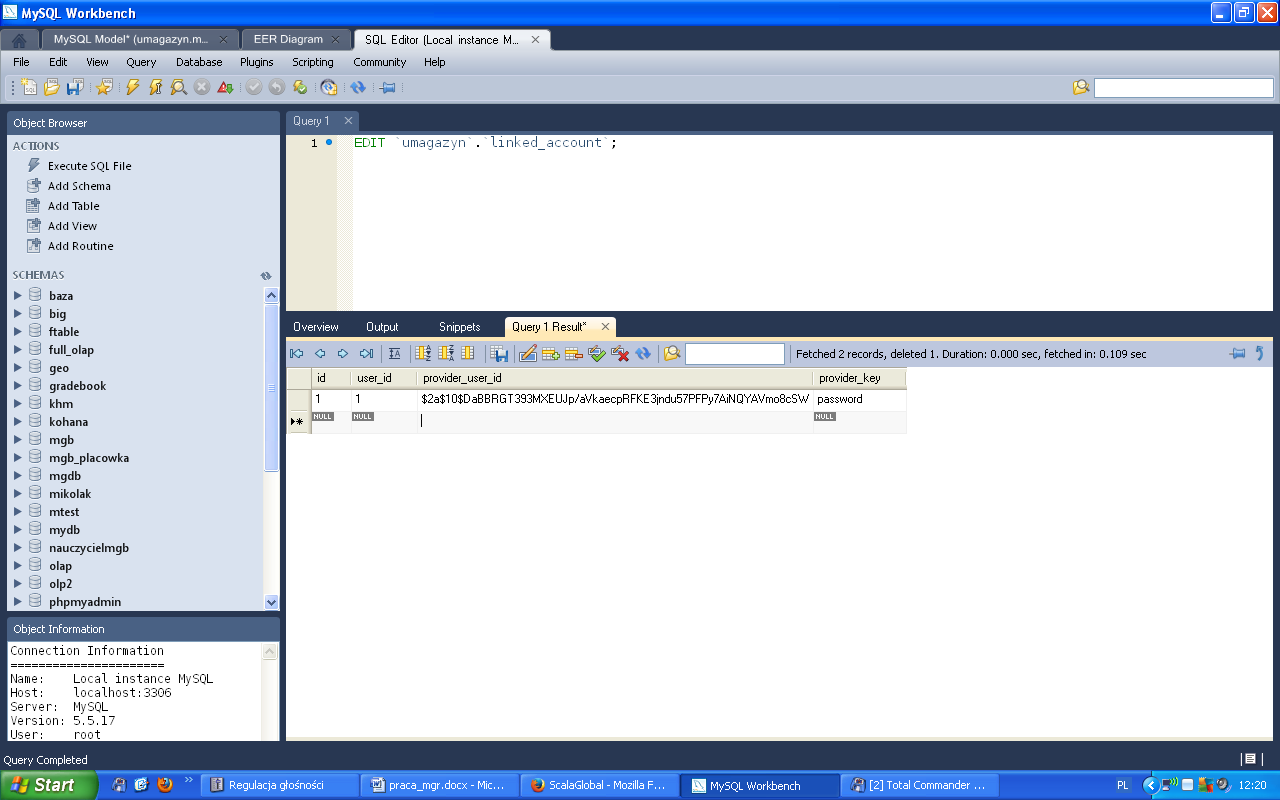
### Tworzenie ról i konta administratora

Model bazy danych stworzony przez przykładową aplikację Play! Authenticate.



Rys .  
Model bazy wygenerowany przez aplikację play! authenticate

Tabela użytkownik (user) zawiera podstawowe informacje na temat konta użytkownika: email, nazwę użytkownika, imię, nazwisko, data ostatniego logowania, informacja czy konto jest aktywne, czy email został zweryfikowany, natomiast nie zawiera pola z hasłem. Żadna tabela będąca w relacji z tabelą user nie zawiera pola hasło, zatem gdzie przechowywane jest hasło ? Ponieważ Play! Authenticate dostarcza możliwości połączenia konta z portalami społecznościowymi, oprócz hasła mysi zostać zdefiniowany dostawca modelu autentykacji informacje na ten temat zawarte są w tabeli *linked\_account*. Przykładowo utworzone konto administratora (opis tworzenia konta administratora przedstawiony w dalszej części) zawiera wpis:



Rys .  
Zawartość tabeli linked\_account

*Provider\_user\_id* jest to pole zawierające zaszyfrowane hasło oprócz hasła bardzo ważnym wpisem jest *provider\_key* który dostarcza informację który moduł autentykacji został użyty przy generacji tokenu np. (password, facebook, google, itd).

Tabela *token\_action* zawiera token który tworzony jest podczas rejestracji nowego użytkownika i służy jako kod aktywacyjny który przesyłany jest email’em. Służy także do funkcjonalności związanej z resetowaniem hasła. Tabela SecurityRole definiuje role użytkowników w systemie.

W systemie występują dwie role administrator (admin) i użytkownik (user). Dodanie domyślnych ról oraz stworzenie konta administratora zostało zaimplementowane w funkcji *initialData* która wołana jest ze zdarzenia *onStart* w klasie *Global*. Jeżeli baza danych nie zawiera dwóch ról użytkownika zostaną one dodane wraz z kontem administratora. Dane konta administratora zostały dodane w pliku konfiguracyjnym *comf\application.conf*

# admin account

account.admin.email = "admin@umagazyn.pl"

account.admin.password = "admin"

Dzięki mapowaniu relacyjno obiektowemu zawartym w module Ebean, sprawdzenie ilości wpisów w tabeli SecurityRole sprowadza się wyłącznie do wywołania polecenia:

SecurityRole.find.findRowCount()

Aby dodać wpis w tabeli SecurityRole wystarczy stworzyć nowy obiekt i wywołać polecenie save().

SecurityRole role = new SecurityRole();

role.roleName = "admin";

role.save();

Pobranie adresu email oraz hasła zdefiniowanych w pliku konfiguracyjnym dla konta administratora wykonuje się za pomocą poleceń:

String email =

play.Play.application().configuration()

.getString("account.admin.email");

String password =

play.Play.application().configuration()  
 .getString("account.admin.password");

Tworzenie konta użytkownika sprowadza się do utworzenia obiektu *user*, ustawienia pola email, konta jako aktywne, adresu email jako zweryfikowany oraz przypisaniem roli administratora.

User user = new User();

user.setEmail(email);

user.setActive(true);

user.setEmailValidated(true);  
 user.setRoles(  
 Collections.singletonList(  
 SecurityRole.findByRoleName("admin")

));

user.save();

user.saveManyToManyAssociations("roles");

Kolejno zostało utworzyć obiekt *UmagazynUsernamePasswordAuthUser* który jest dostawcą hasła dla metody *password* (Rys 3.9) oraz obiekt *LinkedAccount* przechowujący zahaszowane hasło.

UmagazynUsernamePasswordAuthUser authUser =

new UmagazynUsernamePasswordAuthUser (password); LinkedAccount linkedAccount = new LinkedAccount();

linkedAccount.setUser(user);

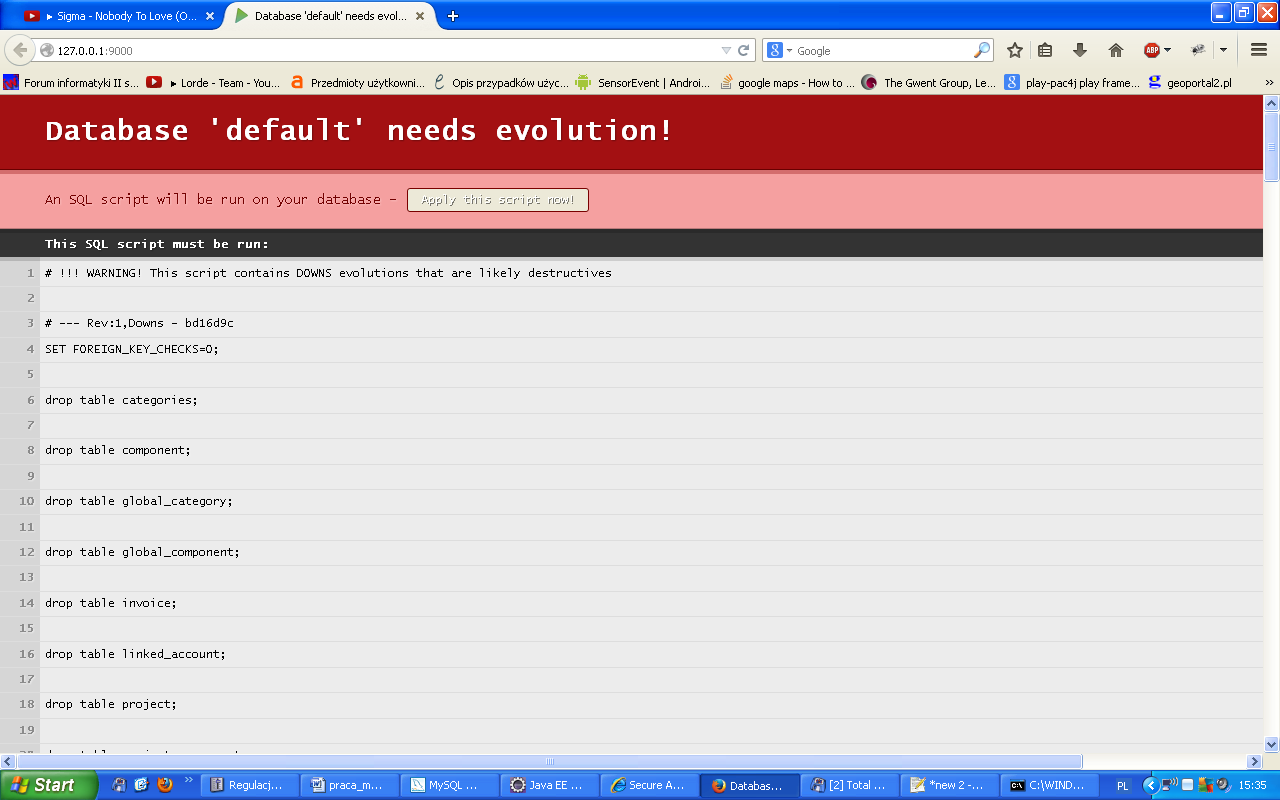
linkedAccount.setProviderKey("password");

linkedAccount.setProviderUserId(authUser.getHashedPassword());

linkedAccount.save();

### Model bazy danych

Tworzenie struktury bazy danych z użyciem Play Framework, sprowadza się do utworzenia klas z użyciem adnotacji zgodnych z JPA. Używany moduł Ebean automatycznie generuje kod SQL bazy danych, utworzy brakujące tabele lub zmieni ich struktury . Akcja aktualizacji bazy danych odbywa się podczas ładowania dowolnej strony po wykryciu zmiany w klasie definiującej model. Przykładowy komunikat aktualizacji bazy danych znajduje się poniżej (Rys 3.10). Akceptacja zmian odbywa się poprzez naciśnięcie przycisku „Apply this scripts now!”



Rys .  
Okno aktualizacji struktury bazy danych

W folderze *app\model\* znajdują się klasy definiujące strukturę bazy danych. Każda klasa reprezentująca tabelę w modelu relacyjnym dziedziczy po klasie Model. Zawiera ona podstawowe metody takie jak zapis, edycja, usuwanie itd. Przykładowa encja poniżej opisuje globalny element elektroniczny zawierający identyfikator, nazwę oraz referencję do globalnej kategorii. Dodatkowo zdefiniowana została statyczna metoda find która zwraca obiekt pomocniczy służący do wyszukiwania danych w bazie, zawierający metody takie jak where, like, eq itp.

@Entity

@Table(name = "global\_components")

**public** **class** GlobalComponent **extends** Model{

**private** **static** **final** **long** *serialVersionUID* = 1L;

@Id

**private** Long id;

**private** String elementName;

@ManyToOne

**private** GlobalCategory globalCategory;

**public** **static** **final** Finder<Long, GlobalComponent> *find* =   
 **new** Finder<Long, GlobalComponent>(

Long.**class**, GlobalComponent.**class**);

}

### Główny plik szablonu

TODO

/\*

Bootstrap w najnowszej wersji v3.2.0.

\*/

### Sekcja administratora

Główną funkcjonalnością administratora jest zarządzanie słownikiem elementów oraz przypisywaniem kategorii do elementów. Ponieważ kategorie elementów zgodnie z założeniem należy sporządzić w formie struktury drzewiastej oraz aby aplikacja nie wymagała przeładowania strony wykorzystano jsTree jest to plugin jQuery, dzięki któremu można dynamicznie tworzyć strukturę kategorii z użyciem technologii Ajax.

TODO

### Sekcja użytkownika

## Komunikacja

## Aplikacja mobilna

### Konfiguracja środowiska

# Testowanie

# Wnioski

# Bibliografia

|  |  |
| --- | --- |
| [1] | S. Maple, „Frameworks Comparison,” 30 7 2013. [Online]. Available: http://zeroturnaround.com/rebellabs/the-curious-coders-java-web-frameworks-comparison-spring-mvc-grails-vaadin-gwt-wicket-play-struts-and-jsf/. |
| [2] | „Play 2.3.x documentation,” [Online]. Available: https://www.playframework.com/documentation/2.3.x/Home. |
| [3] | „Tworzenie nazwy źródła danych w systemie,” 10 08 2014. [Online]. Available: http://support.microsoft.com/kb/305599/pl. |
| [4] | „Ebean documentation,” [Online]. Available: http://www.avaje.org/ebean/documentation.html. |
| [5] | „Deadbolt documentation,” [Online]. Available: http://deadbolt.ws/#/java-docs. |
| [6] | „Introduction to the Java Persistence API,” [Online]. Available: http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html. |
| [7] | „jsTree documentation,” [Online]. Available: http://www.jstree.com/docs/. |

1. Strona internetowa projektu Akka: http://akka.io/ [↑](#footnote-ref-1)
2. https://typesafe.com/ [↑](#footnote-ref-2)