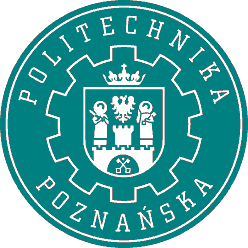
**POLITECHNIKA POZNAŃSKA**

**WYDZIAŁ INFORMATYKI**

**INSTYTUT INFORMATYKI**

Zastosowanie steganografii do wymiany informacji na forum internetowym

DANIEL KOZA  
BARTOSZ KOSTANIAK

Promotor: prof. dr hab. inż. Jerzy Brzeziński   
Instytut Informatyki   
Politechniki Poznańskiej

**KARTA PRACY DYPLOMOWEJ**

(kserokopia z podpisami)

**Spis treści**

[1. Wstęp 3](#_Toc439597651)

[1.1 Cel i zakres pracy 4](#_Toc439597652)

[1.2 Struktura pracy 4](#_Toc439597653)

[2. Wprowadzenie 5](#_Toc439597654)

[2.1 Rys historyczny 5](#_Toc439597655)

[2.2 Idea steganografii 5](#_Toc439597656)

[2.3 Nośniki informacji umożliwiające zaimplementowanie steganografii 5](#_Toc439597657)

[2.4 Narzędzia komunikacji w Internecie 5](#_Toc439597658)

[2.4.1 Czat internetowy 5](#_Toc439597659)

[2.4.2 Forum internetowe 5](#_Toc439597660)

[2.5 Steganografia w plikach graficznych 5](#_Toc439597661)

[3. Przedstawienie użytych technologii 6](#_Toc439597662)

[3.1 Git 6](#_Toc439597663)

[3.2 JavaScript 6](#_Toc439597664)

[3.3 Node.JS 6](#_Toc439597665)

[3.3.1 Platforma Node.JS 6](#_Toc439597666)

[3.3.2 Node Package Manager (NPM) 6](#_Toc439597667)

[3.4 MongoDB 7](#_Toc439597668)

[3.5 MeteorJS 7](#_Toc439597669)

[3.5.1 Ogólny opis 7](#_Toc439597670)

[3.5.2 Struktura plików 7](#_Toc439597671)

[3.5.3 Modele i metody 8](#_Toc439597672)

[3.6 AngularJS (wersja 1.x) 9](#_Toc439597673)

[3.6.1 Ogólny opis 9](#_Toc439597674)

[3.6.2 Jednostki 9](#_Toc439597675)

[3.6.3 Two-way data binding 11](#_Toc439597676)

[3.6.4 Angular-Meteor 11](#_Toc439597677)

[3.7 Preprocessory 12](#_Toc439597678)

[3.7.1 CoffeeScript 12](#_Toc439597679)

[3.7.2 Jade 13](#_Toc439597680)

[3.7.3 Less 13](#_Toc439597681)

[3.8 Narzędzia do zarządzania projektem 13](#_Toc439597682)

[3.8.1 TravisCI 13](#_Toc439597683)

[3.9 Narzędzia do testowania 13](#_Toc439597684)

[3.9.1 GulpJS 13](#_Toc439597685)

[3.9.2 Jasmine 13](#_Toc439597686)

[3.9.3 Karma 13](#_Toc439597687)

[3.9.4 Protractor 13](#_Toc439597688)

[4. Budowa aplikacji 14](#_Toc439597689)

[4.1 Struktura plików i folderów 14](#_Toc439597690)

[4.2 Modele w bazie danych 14](#_Toc439597691)

[4.3 Jednostki w aplikacji 14](#_Toc439597692)

[5. System uwierzytelniania i autoryzacji 15](#_Toc439597693)

[5.1 Rejestrowanie i logowanie użytkowników 15](#_Toc439597694)

[5.2 Role 15](#_Toc439597695)

[5.3 Struktura ról 15](#_Toc439597696)

[5.4 Panel administracyjny 15](#_Toc439597697)

[6. System ukrywania wiadomości 16](#_Toc439597698)

[6.1 16](#_Toc439597699)

[7. Testy 17](#_Toc439597700)

[7.1 Testy jednostkowe 17](#_Toc439597701)

[7.2 Testy integracyjne 17](#_Toc439597702)

[7.3 Testy funkcjonalne 17](#_Toc439597703)

[8. Zakończenie 18](#_Toc439597704)

[Literatura 19](#_Toc439597705)

# Wstęp

Wraz z wykształceniem się u człowieka umiejętności komunikacji powstał problem skutecznego przekazywania tajnych informacji, takich których zawartość mogłaby być znana tylko przez nadawcę i odbiorcę. Jak bezpiecznie przesłać list do adresata, tak aby pośrednik nie był w stanie jej odczytać? Jak wysłać do króla wiadomość, której nikt inny nie byłby w stanie odczytać? Jak przekazać polecenia dla szpiega, który znajduje się za linią frontu? Te i wiele innych pytań towarzyszy ludzkości praktycznie od powstania cywilizacji. Najczęstszym rozwiązaniem tych problemów będzie szyfrowanie wiadomości przez nadawcę i odszyfrowania jej za pomocą tajnego klucza przez odbiorcę. Dziedzinę nauki zajmującą się przekazywaniem informacji w sposób zabezpieczony przed niepowołanym dostępem nazywamy kryptologią. Dzieli się ona na kryprografię, czyli gałąź wiedzy o utajnianiu wiadomości oraz kryptoanalizę, czyli gałąź wiedzy o przełamywaniu zabezpieczeń oraz deszyfrowaniu wiadomości. Przekazywanie informacji z pomocą kryptografii ma jednak jedną, ale zasadniczą wadę: obecność tajnego komunikatu jest jawna, choć treść jest zaszyfrowana. Steganografia rozwiązuje ten problem. Jest to nauka o komunikacji w taki sposób, aby obecność komunikatu nie mogła zostać wykryta. Połączenie tych dwóch dziedzin nauki daje nam najwyższy możliwy poziom bezpieczeństwa w przekazywaniu tajnych wiadomości.

Warto też zauważyć jak bardzo na przestrzeni dziejów zmieniały się narzędzia komunikacji. Zaczynając od prostej mowy, pisma, przechodząc przez radio, telewizję, a kończąc na Internecie człowiek zyskał bardzo wiele możliwości przesyłania dowolnej wiadomości (np. w formie pliku graficznego, filmu) na dowolną odległość. Dzięki temu implementacja steganografii w dwudziestym pierwszym wieku jest teoretycznie ograniczona jedynie przez ludzką wyobraźnię.

Motywacją do podjęcia tematu była chęć stworzenia jak najbezpieczniejszego narzędzia do komunikacji w Internecie. Praktycznie każda większa korporacja, rządy państw nieustannie zbierają informację na temat każdego użytkownika przeglądającego witryny WWW. Dzięki forum steganograficznemu przeciętny człowiek, bez większych umiejętności informatycznych, będzie w stanie wymieniać wiadomości, nie obawiając się o szpiegowanie ze stron trzecich. Istnieje co prawda tzw. Darknet (pl. Ciemny Internet), czyli miejsce gdzie istnieje wolność absolutna, jednakże jest on trudny w obsłudze dla przeciętnego użytkownika.

## Cel i zakres pracy

Celem niniejszej pracy jest stworzenie w pełni funkcjonalnego forum internetowego wraz z zaimplementowanym mechanizmem do ukrywania tajnych wiadomości. Aplikacja powinna umożliwiać rejestrację i logowanie użytkowników, tworzenie, edycję, przeglądanie i usuwanie sekcji, tematów oraz postów. Posty powinny należeć do tematów, a tematy do sekcji. Należy umożliwić wgranie pliku graficznego i przypisanie go do danego postu.

## Struktura pracy

Dalsza część pracy zorganizowana jest w sposób następujący. Rozdział 2 zawiera ...

# Wprowadzenie

## Rys historyczny

## Idea steganografii

## Nośniki informacji umożliwiające zaimplementowanie steganografii

## Narzędzia komunikacji w Internecie

### Czat internetowy

### Forum internetowe

## Steganografia w plikach graficznych

# Przedstawienie użytych technologii

## Git

Git jest rozproszonym systemem kontroli wersji. Został stworzony przez Linusa Torvaldsa jako narzędzie wspomagające rozwój jądra Linux. W krótkim czasie narzędzie to zyskało dużą popularność.

## JavaScript

JavaScript jest skryptowym językiem programowania stosowanym najczęściej na stronach internetowych, jednakże dzięki różnym bibliotekom coraz częściej jest wykorzystywany także w innych środowiskach. Łączy w sobie następujące paradygmaty: obiektowy, funkcyjny oraz imperatywny. Jest to język dynamiczne i słabo typowany, czyli typy zmiennych są nadawane dynamicznie w czasie wykonywania skryptu, a konwersje na różne typy danych są wykonywane automatycznie. W języku tym występują obiekty, takie jak: Object, Array oraz typy prymitywne: Boolean, Number, String, Null, Undefined. Standardem JavaScript jest ECMAScript, czyli ustandaryzowany przez organizację ECMA skryptowy, obiektowy język programowania. Specyfikacja ta oznaczona jest jako ECMA-262 i ISO/IEC 16262.

## Node.JS

### Platforma Node.JS

Node.JS jest platformą umożliwiającą uruchamianie skryptów JavaScript (JS) w innym środowisku niż przeglądarka internetowa. Korzysta ona z otwartego silnika V8, który został stworzony i jest rozwijany przez firmę Google. Wykorzystuje go popularna przeglądarka internetowa Google Chrome. Node.JS korzysta także z biblioteki libuv, która zapewnia asynchroniczne powiadamianie o zdarzeniach, jak i również z wielu innych bibliotek, dzięki czemu Node.JS jest w środowiskiem programistycznym wykorzystywanym do tworzenia wysoce skalowalnych, sterowanych zdarzeniami i wykorzystujących asynchroniczny system wejścia-wyjścia aplikacji internetowych.

### Node Package Manager (NPM)

W platformę Node.JS wbudowany jest system modułów nazywany Node Package Manager, w skrócie NPM. Umożliwia on doinstalowywanie zewnętrznych modułów do aplikacji stworzonej w środowisku Node.JS. Za pomocą jednej komendy programista jest w stanie ściągnąć konkretną wersję danej biblioteki. Dodatkowo jest możliwość zapisania listy używanych bibliotek do pliku package.json, który znajduje się w głównym katalogu projektu, dzięki czemu inny programista będzie mógł bez problemów ściągnąć wszystkie potrzebne zależności, które nie będą się znajdowały w zdalnym repozytorium. W pliku tym możliwe jest również zapisanie podstawowych informacji o projekcie, zdefiniowanie prostych skryptów czy zarządzanie wersjami.

Listing 1. Przykładowy plik package.json

{

"name": "angular-unit-testing-helpers",

"version": "0.0.6",

"description": "A collection of helper functions for writing AngularJS unit tests.",

"scripts": {

"test": "./node\_modules/karma/bin/karma start",

"test-travis": "./node\_modules/karma/bin/karma start --single-run --browsers=Chrome\_travis\_ci,Firefox,PhantomJS"

},

"repository": {

"type": "git",

"url": "git+https://github.com/dakolech/angular-unit-testing-helpers.git"

},

"keywords": [

"angularJS",

"angular",

"unit",

"testing",

"test",

"helpers"

],

"author": "Daniel Koza",

"license": "MIT",

"bugs": {

"url": "https://github.com/dakolech/angular-unit-testing-helpers/issues"

},

"homepage": "https://github.com/dakolech/angular-unit-testing-helpers#readme",

"devDependencies": {

"angular": "1.5.0-rc.0",

"angular-mocks": "^1.4.8",

"jasmine-core": "^2.3.4",

"karma": "^0.13.15",

"karma-chrome-launcher": "^0.2.1",

"karma-firefox-launcher": "^0.1.7",

"karma-jasmine": "^0.3.6",

"karma-ng-html2js-preprocessor": "^0.2.0",

"karma-phantomjs2-launcher": "^0.4.0",

"phantomjs": "^1.9.19"

}

}

## MongoDB

MongoDB jest otwartym, nierelacyjnym systemem zarządzania danych. Charakteryzuje się dużą skalowalnością, wydajnością oraz dowolnością w strukturze baz danych. Dane są składowane w postaci dokumentów JSON (JavaScript Object Notation), dzięki czemu umożliwia to aplikacjom bardziej naturalne ich przetwarzanie, zwłaszcza przez biblioteki napisane w języku JavaScript.

## MeteorJS

### Ogólny opis

MeteorJS jest biblioteką wykorzystywaną do tworzenia aplikacji internetowych. Została napisana z użyciem platformy Node.JS. Jest zintegrowana z nierelacyjnymi bazami danych MongoDB. Działa zarówno po stronie klienta (przeglądarka internetowa), jak i serwera. Wykorzystuje wzorzec projektowy publikuj-subskrybuj do automatycznego propagowania modyfikacji danych w stronę klienta, bez konieczności pisania kodu synchronizującego przez programistę. Dzięki temu stworzenie w pełni funkcjonalnej aplikacji internetowej wymaga dużo mniej pracy niż w konkurencyjnych bibliotekach, takich jak Ruby on Rails czy Django. Jedną z głównych zalet MeteorJS jest to, że cała komunikacja między przeglądarką internetową a serwerem, jest wykonywana za pomocą WebSocket’ów. WebSocket jest to technologia zapewniająca dwukierunkowy kanał komunikacji za pośrednictwem jednego gniazda TCP.

MeteorJS posiada własny menadżer bibliotek i wtyczek, AtmosphereJS, który bardzo przypomina ten z Node.JS. Podobnie jak w NPM, programista może zainstalować zewnętrzną bibliotekę za pomocą jednego polecenia, a informacje o niej są zapisywane do pliku. Jednakże, w odróżnieniu od platformy Node.JS, nie są wymagane referencje do wtyczek w kodzie, albowiem wszystkie biblioteki są dostępne globalnie.

### Struktura plików

MeteorJS posiada z góry określoną strukturę plików i kolejność ich ładowania, co ma duże znaczenie, zwłaszcza jeżeli aplikacja korzysta z dodatkowych bibliotek, takich jak na przykład AngularJS. W uproszczeniu: wszystko co znajduje się w folderze .meteor służy do konfiguracji wewnętrznych i zewnętrznych bibliotek oraz ich wersji, zawartość folderu server jest widoczna i wykonywana tylko po stronie serwera, podobnie jak katalogu private, pliki w folderze client natomiast są wykonywane i widocznie tylko po stronie klienta, czyli w tym przypadku, przeglądarki internetowej, a skrypty w katalogu tests nie są widoczne ani przez serwer, ani przez klienta, służą tylko i wyłącznie do wykonywania testów. Kolejność ładowania plików jest następująca: wszystkie skrypty znajdujące się w katalogu lib ładowane są w pierwszej kolejności, następnie MeteorJS wczytuje resztę plików według kolejności alfabetycznej, najpierw z podkatalogu, a później z katalogu nadrzędnego, z wyjątkiem plików rozpoczynających się słowem main, które są ładowane na samym końcu.

### Modele i metody

Dane w MeteorJS są przechowywane za pomocą kolekcji. Jako że biblioteka ta korzysta z MongoDB, struktura modeli w bazie danych nie jest określona. Kolekcje tworzy się raz i są dostępne globalnie.

Listing 1. Przykład inicjalizacji kolekcji Messages dostępnej globalnie.

Messages = new Mongo.Collection("messages");

W MeteorJS używany jest po stronie serwera obiekt globalny Meteor, który posiada metodę publish oraz methods. Publish służy do wyświetlania danych bezpośrednio z MongoDB. Każda aktualizacja modelu lub kolekcji w bazie danych jest od razu widoczna w przeglądarce internetowej, bez żadnego przeładowywania strony, dzięki zastosowaniu WebSocket’ów. Oznacza, to że dane publikowane za pomocą tej metody są bezpośrednio połączone z danymi wyświetlanymi w widoku. Nie jest jednakże możliwa ich bezpośrednia modyfikacja za pomocą tej metody. Methods natomiast służy do definiowania akcji modyfikujących dane. Zarówno do metody publish, jak i methods, przesyłana zostaje funckja, która, w pierwszym wypadku musi zwrócić wektor z bazy danych, a w drugim może zwrócić dowolną wartość.

Listing 2. Przykład metody publish, która zwraca wszystkie obiekty z kolekcji Rooms

Meteor.publish("rooms", function () {

return Rooms.find({});

});

Listing 3. Przykład użycia methods. Podany kod tworzy dwie metody: foo oraz bar.

Meteor.methods({

foo: function (arg1, arg2) {

check(arg1, String);

check(arg2, [Number]);

return "some return value";

},

bar: function () {

return "baz";

}

});

## AngularJS (wersja 1.x)

### Ogólny opis

AngularJS w wersji 1.x jest jedną z najbardziej popularnych bibliotek, napisanych w języku JavaScript, do tworzenia SPA (Single Page Application), czyli stron internetowych, które bardzo przypominają aplikacje działające natywnie w środowisku Windows, Linux czy Mac OSX. Wszelkie akcje, jakie użytkownik wykonuje, dzieją się praktycznie tylko na jednej stronie, bez żadnego przeładowywania, a wszystkie dane pobierane i wysyłane do serwera są wykonywane za pomocą technologii AJAX (Asynchronous JavaScript and XML). Użytkownik ma wrażenie, że użytkuje normalną stronę, miedzy innymi z powodu zmieniających się widoków i adresów URL w przeglądarce internetowej. AngularJS korzysta z popularnego wzorca projektowego MVC (Model View Controller), delikatnie go modyfikując. AngularJS jest stosunkowo młodą biblioteką, albowiem został opublikowany w 2012 roku przez firmę Google. Dzięki wsparciu tej korporacji stał się wiodącą technologią w dziedzinie tworzenia nowoczesnych aplikacji internetowych. W 2016 roku ma zostać wydana wersja 2.0, która w bardzo dużym stopniu różni się od aktualnej edycji. Dlatego tak ważne jest podanie, której wersji programista używa.

### Jednostki

AngularJS opiera się na modułach. Są to podstawowe jednostki w tej bibliotece, dzięki którym organizujemy całą aplikację. Do każdego modułu możemy wstrzyknąć inny moduł, przez co tworzone jest drzewo zależności oraz zostaje usunięty problem globalnego definiowania obiektów. W bardzo dużym stopniu ułatwia to testowanie jednostkowe, ponieważ wstrzykiwany do testu jest tylko jeden moduł. Wszystkie jednostki opisane poniżej tworzone są właśnie w modułach.

Listing 4. Przykład inicjalizacji modułu myApp wraz ze wstrzykniętym modułem restangular

var myApp = angular.module('myApp',['restangular']);

Rolę modelu we wzorcu MVC pełnią serwisy, takie jak: service, factory, provider. Mają one dwie podstawowe cechy: AngularJS inicjalizuje je wtedy i tylko wtedy, gdy jakiś komponent od nich zależy, a także korzystają ze wzorca singleton, czyli każdy komponent otrzymuje tę samą, pojedynczą referencję do danego serwisu. Jednostki te odpowiadają głównie za logikę biznesową aplikacji oraz za komunikację z serwerem. Do serwisu można wstrzyknąć inny serwis.

Listing 5. Przykład serwisu w AngularJS

angular.

module('myServiceModule', []).

factory('notify', ['$window', function(win) {

var msgs = [];

return function(msg) {

msgs.push(msg);

if (msgs.length == 3) {

win.alert(msgs.join("\n"));

msgs = [];

}

};

}]);

Jedną z podstawowych jednostek w AngularJS są kontrolery, czyli litera C we wzorcu MVC. Odpowiadają one za zarządzanie danymi w widoku oraz za wykorzystywanie serwisów. Są łącznikiem między warstwą logiki a warstwą prezentacji aplikacji. W przeciwieństwie do serwisów, które są inicjalizowane tylko raz w trakcie używania aplikacji, kontrolery są tworzone za każdym razem, gdy dany widok od nich zależy i niszczone, gdy są już nieużywane. Ma to ogromne znaczenie w procesie projektowania aplikacji. Kontrolery powinny zawierać jak najmniej kodu i nie zarządzać logiką biznesową, ze względu na to, że nie przechowują danych (są niszczone przy każdej zmianie widoku) i nie da się ich wstrzyknąć do innego komponentu.

Listing 6. Przykład kontrolera ze wstrzykniętym serwisem $scope w AngularJS

myApp.controller('DoubleController', ['$scope', function($scope) {

$scope.double = function(value) {

return value \* 2;

};

}]);

Esencją AngularJS są dyrektywy. Dodają one dodatkowe elementy lub atrybuty do widoku, rozszerzając dość ograniczone możliwości dokumentów HTML. AngularJS posiada 69 wbudowanych dyrektyw (w wersji 1.4), które zaczynają się od liter ng, na przykład: ng-click=”funkcja()”, ng-if=”false” czy ng-controller=”DoubleController”. Pierwsza służy do przypisania danej funkcji do zdarzenia kliknięcia na daną część strony, druga usuwa element z DOM’u, gdy wyrażenie jest nieprawdziwe, a ostatnia przypisuje podany kontroler do widoku. Biblioteka umożliwia także tworzenie nowych dyrektyw. Najprostszy przykład znajduje się w listingach 7-9. Oczywiście jest możliwość tworzenia bardziej skomplikowanych elementów, które będą przyjmować różne dane wejściowe, posiadać swój własny kontroler, czy korzystać z serwisów.

Listing 7. Przykład dyrektywy w AngularJS

myApp.directive('redName', function() {

return {

restrict: 'AE',

template: '<div class=”red”><span> My name </span></div>'

};

};);

Listing 8. Przykład użycia dyrektywy z listingu 7. w dokumencie HTML

<red-name></red-name>

<div red-name></div>

Listing 9. Wygenerowany dokument HTML z listingu 8. w przeglądarce internetowej

<div class=”red”>

<span>

My name

</span>

</div>

<div class=”red”>

<span>

My name

</span>

</div>

### Two-way data binding

Jedną z najważniejszych cech, jeśli nie najważniejszą, jest tak zwane dwukierunkowe wiązanie danych (ang. two-way data binding). Wszystkie dane przypisane do serwisu $scope, są bezpośrednio powiązane z danymi występującymi w dokumencie HTML. Oznacza to, że każda zmiana modelu, jest od razu widoczna w widoku, jak i każda aktualizacja danej, na przykład w formularzu, jest od razu dostępna w serwisie $scope. AngularJS wyłapuje zmiany w modelach przez porównanie wartości z wartościami zgromadzonymi we wcześniejszym procesie, co jest nazywane pętlą digest (ang. digest loop). Cała aplikacja jest podzielona na wiele zakresów (scope): główny $rootScope, dostępny na całej stronie, oraz jego potomki, nowy dla każdego kontrolera. Pętla digest rozpoczyna się od najmłodszego potomka i aktualizuje wszystkie wartości z poszczególnych zakresów, przechodząc przez rodziców, a kończąc na $rootScope.

### Angular-Meteor

Angular-Meteor jest to moduł do biblioteki AngularJS umożliwiający jego współpracę z platformą MeteorJS. Dodaje on między innymi serwis $meteor, który ułatwia komunikację z serwerem. Dzięki niemu w aplikacji zostaje wykorzystane nawet trójkierunkowe wiązanie danych (ang. three-way data binding), dane z bazy danych są bezpośrednio połączone z danymi w serwisach AngularJS, jak i w dokumencie HTML, przez co każda aktualizacja obiektów na serwerze jest od razu widoczna w przeglądarce. Moduł ten dodaje także wiele innych funkcjonalności, takich jak narzędzia uwierzytelniające współpracujące z wtyczką Accounts do MeteorJS, czy obsługę kolekcji z MongoDB i metod z obiektu globalnego Meteor.

## Bower

Bower jest narzędziem podobnym do NPM, ale jest znacznie prostszy. Służy głównie do zarządzania bibliotekami używanymi po stronie klienta. Pliki źródłowe ściągane są z zdalnego repozytorium i zapisywane do katalogu bower\_components. Konfiguracja narzędzia Bower znajduje się w dokumencie bower.json.

## Preprocessory

### CoffeeScript

CoffeeScript jest jednym z pierwszych preprocesorów jaki powstał do języka JavaScript. Dodaje on tak zwany lukier składniowy, aby zwiększyć czytelność i skrócić objętość kodu. Jego twórcy w dużym stopni inspirowali się językiem Ruby oraz Python. W CoffeScript słowo kluczowe function zostało zastąpione -> lub =>, które dodatkowo wstrzykuje do funkcji obiekt this z wyższego bloku kodu. Nie są wymagane nawiasy okrągłe, ani nawiasy klamrowe. Podobnie jak w Python’ie, definicja funkcji jest wyznaczana przez wcięcie w kodzie. Nie trzeba pisać także słowa kluczowego return, gdyż, podobnie jak w Ruby, funkcja zwraca ostatnie wyrażenie w jej ciele. Program napisany w CoffeeScript jest kompilowany do czystego JavaScript, także bez problemowo działa w każdej przeglądarce.

Listing 10. Przykład kodu w JavaScript

function listen (el, event, handler) {

if (el.addEventListener) {

return el.addEventListener(event, handler);

} else {

return el.attachEvent("on" + event, function() {

return handler.call(el);

});

}

}

Listing 11. Kod z listingu 10 napisany w CoffeeScript

listen = (el, event, handler) ->

if el.addEventListener

el.addEventListener event, handler

else

el.attachEvent 'on' + event, ->

handler.call el

### Jade

Jade jest preprocesorem do dokumentów HTML. Jego składnia cechuje się dużą prostotą w stosunku do popularnego języka znaczników używanego głównie przez przeglądarki internetowe. Klasy oraz identyfikatory przypisuje się tak jak w CSS (zaczynając od kropki lub znaku #), wcięcia oznaczają potomne elementy, atrybuty znajdują się w nawiasach okrągłych, a tagi nie posiadają znaków < oraz >. Szablon Jade jest kompilowany do czystego HTML.

Listing 12. Przykład kodu w HTML

<!doctype html>

<html>

<head>

<link type="text/css" rel="stylesheet" href="/site.css" />

<title>Hello</title>

</head>

<body>

<h1>Hello world!</h1>

</body>

</html>

### 

Listing 13. Kod z listingu 12 napisany w Jade

doctype html

html

head

link(type='text/css', rel='stylesheet', href='/site.css')

title Hello

body

h1 Hello world!

### Less

Less jest preprocesorem do kaskadowych arkuszy stylów. Stara się on wprowadzić elementy z normalnych języków programowania, takie jak na przykład zmienne, funkcje czy pętle. Jedną z jego najważniejszych cech jest zagnieżdżanie reguł CSS. Drastycznie poprawia ono czytelność kodu, jak i jego objętość. Less umożliwia także importowanie styli z różnych plików, dzięki czemu programista może zdefiniować kolejność wczytywania arkuszy. Less jest kompilowany do czystego CSS.

Listing 14. Przykład kodu w CSS

.box {

color: #fe33ac;

border-color: #fdcdea;

}

.box div {

-webkit-box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);

}

Listing 15. Kod z listingu 12 napisany w Less

@base: #f938ab;

.box-shadow(@style, @c) when (iscolor(@c)) {

-webkit-box-shadow: @style @c;

box-shadow: @style @c;

}

.box-shadow(@style, @alpha: 50%) when (isnumber(@alpha)) {

.box-shadow(@style, rgba(0, 0, 0, @alpha));

}

.box {

color: saturate(@base, 5%);

border-color: lighten(@base, 30%);

div { .box-shadow(0 0 5px, 30%) }

}

## 

## Narzędzia do zarządzania projektem

### TravisCI

TravisCI jest narzędziem wykorzystywanym do ciągłej integracji. Każdy projekt dostępny na platformie Github.com ma możliwość dodania pliku .travis.yml, który odpowiada za konfigurację. Podstawową funkcjonalnością TravisCI jest możliwość budowania oraz testowania aplikacji w zamkniętym środowisku i przesyłania informacji o statusie na skrzynkę pocztową właściciela projektu. Takie zadania mogą być wykonywane dla wybranych gałęzi oraz Pull Request’ów. Przy odpowiedniej konfiguracji narzędzie to jest w stanie wysyłać kod źródłowy na serwer produkcyjny w przypadku pozytywnie zakończonych wszystkich poleceń. TravisCI jest wykorzystywany przez wiele projektów z otwartym kodem źródłowym, ponieważ jest płatny tylko dla prywatnych repozytoriów.

Listing 16. Przykładowy plik .travis.yml

language: node\_js

node\_js:

- '5'

script:

- npm run-script test-travis

before\_script:

- npm install

## 

## Narzędzia do testowania

### GulpJS

GulpJS jest systemem automatyzacji pracy dostępny na platformie Node.JS. Za pomocą tego narzędzia programista jest w stanie zdefiniować często wykonywane przez niego zadania na plikach, takie jak kompilacja preprocesorów, kopiowanie, usuwanie, wykonywanie testów czy sprawdzanie składni w kodzie. Cała konfiguracja znajduje się w pliku gulpfile.js.

Listing 17. Przykładowy plik gulpfile.js

'use strict';

var gulp = require('gulp');

var sass = require('node-sass');

var rename = require('gulp-rename');

var replace = require('gulp-replace');

var rimraf = require('gulp-rimraf');

gulp.task('build', ['images'], function() {

var css = sass.renderSync({

file: 'template/scss/styles.scss'

});

gulp.src('template/screenshot.jpg').pipe(gulp.src('build'));

return gulp.src('template/responsive.html')

.pipe(replace('app.css', css))

.pipe(rename('index.html'))

.pipe(gulp.dest('build'));

});

gulp.task('clean', function() {

return gulp.src('build', {

read: false

})

.pipe(rimraf());

});

### Jasmine

Jsmine jest biblioteką wspomagającą testowanie, dostępną na platformie Node.JS. Testy są wykonywane w funckjach it, które można grupować w blokach describe. Istnieją także funkcje beforeEach oraz afterEach, wykonywane odpowiednio przed i po każdym teście. Walidacja poprawności wykonania danego testu odbywa się w funkcji expect, która posiada takie atrybuty jak toBe, toEqual, oznaczające, że wyrażenie podane jako argument w expect ma być takie samo jak podany obiekt. W Jasmine istnieją także specjalne obiekty spyOn (pl. szpieguj na), które sprawdzają czy dana metoda została wywołana i z jakimi argumentami. Jest możliwość także nadpisania istniejących funkcji, tak aby zachowywały się tak, jak program tego oczekuje.

Listing 18. Przykładowe test w Jasmine

describe("Hello world", function() {

it("says world", function() {

expect(helloWorld()).toContain("world");

});

});

describe("Person", function() {

it("calls the sayHello() function", function() {

var fakePerson = new Person();

spyOn(fakePerson, "sayHello");

fakePerson.helloSomeone("world");

expect(fakePerson.sayHello).toHaveBeenCalled();

});

});

### Karma

Karma umożliwia automatyczne wykonywanie testów na różnych przeglądarkach internetowych, takich jak Firefox, Chrome, czy PhantomJS (narzędzie zastępujące prawdziwą przeglądarkę). Biblioteka ta została stworzona przez twórców AngularJS. Posiada wiele wtyczek, dzięki którym można na przykład sprawdzić pokrycie kodu testami w aplikacji lub wskazać używaną bibliotekę. Plikiem odpowiadającym za konfigurację narzędzia jest najczęściej karma.config.js.

Listing 19. Przykładowy plik karma.config.js

module.exports = function(config) {

config.set({

basePath: '',

frameworks: ['jasmine'],

files: [

'node\_modules/angular/angular.js',

'node\_modules/angular-mocks/angular-mocks.js',

'test-helpers.js',

'test/\*\*/\*.js',

'examples/\*\*/\*.js',

'examples/\*\*/\*.html'

],

preprocessors: {

'examples/\*\*/\*.html': ['ng-html2js']

},

reporters: ['progress'],

port: 9876,

logLevel: config.LOG\_INFO,

autoWatch: true,

browsers: ['PhantomJS2'],

singleRun: false

}

### Protractor

Protractor jest narzędziem wykorzystywanym do wykonywania testów end-to-end (funkcjonalnych) dla aplikacji napisanej w AngularJS. Biblioteka ta wykonuje swoje zadania na praktycznie już gotowej stronie internetowej, naśladując zachowanie użytkownika w przeglądarce. Wykorzystuje do tego oprogramowanie Selenium. Wszelka konfiguracja znajduje się najczęściej w pliku protractor.config.js.Protractor, podobnie jak Karma może używać biblioteki Jasmine do tworzenia testów.

Listing 20. Przykładowy test w protractor

'use strict';

describe('Main scenario', function() {

it('should load the main page', function() {

browser.get('/');

expect(browser.isElementPresent(By.css('body'))).toBe(true);

});

});

Listing 21. Przykładowy plik protractor.config.js

exports.config = {

jasmineNodeOpts: {

showColors: true,

defaultTimeoutInterval: 300000

},

specs: ['\*\*/\*.scenario.js'],

seleniumArgs: ['-browserTimeout=60'],

seleniumAddress: 'http://localhost:4444/wd/hub',

multiCapabilities: [{

'browserName': 'chrome'

}]

};

# Architektura aplikacji

## Struktura plików i folderów

### .gulp

Niestety MeteorJS nie współpracuje z NPM (w wersji 1.2.1), dlatego żeby użyć GulpJS, potrzebny jest ukryty katalog .gulp (każdy plik lub katalog zaczynający się kropką jest niewidoczny dla aplikacji). Znajdują się w nim .bowerrc, bower.json oraz packages.json do zarządzania zewnętrznymi bibliotekami. Za pomocą gulpfile.coffee skonfigurowane zostało narzędzie GulpJS, używane do generowania algorytmu seganograficznego (rozdział 6.), jak i do startowania testów jednostkowych oraz funkcjonalnych (włączenie Selenium oraz Protractor).

### .meteor

Folder .meteor jest generowany automatycznie podczas tworzenia nowego projektu w MeteorJS. Wszystkie pliki znajdujące się w tym katalogu powstają i są aktualizowane automatycznie, bez ingerencji programisty. W .id znajduje się unikalny identyfikator dla aplikacji. Wszystkie używane wtyczki lub biblioteki zapisywane są do dokumentu packages, a numery ich wersji oraz zależności znajdują się w versions. Platforms odpowiada za platformy docelowe aplikacji, w tym przypadku jest to browser (przeglądarka internetowa) oraz server (serwer). W pliku release znajduje się numer wersji samej platformy MeteorJS.

### .stegano

### client

W katalogu client znajduje się cała aplikacja klienta napisana w AngularJS. Każda nazwa pliku .coffee, opisuje jego zawartość, i tak skrypty z końcówką module.coffee oznaczają definicję modułu, directive.coffee – dyrektywy, service.coffee – serwisu, a controller.coffee – kontrolera. Pliki z widokami posiadają końcówkę ng.jade. Jest to spowodowane wymaganiami wtyczki do zarządzania szablonami Jade dla AngularJS. Główny dokument index.jade nie potrzebuje ng w nazwie. Ze względu na brak możliwości ustawienia kolejności ładowania skryptów w MeteorJS, wszystkie definicje modułów muszą się znajdować w folderach lib. Jako, że w tych katalogach może się znajdować więcej plików (na przykład skrypty konfiguracyjne), to nazwy dokumentu z modułem zaczynają się literą a (lub an w przypadku gdy nazwa rozpoczyna się samogłoską). Dzięki temu zabiegowi, AngularJS bez problemów wczytuje wszystkie moduły. Architektura aplikacji klienckiej jest zorganizowana w sposób modułowy, to znaczy, że każdy moduł posiada osobny katalog. Ułatwia to w znacznym stopniu zidentyfikowanie zastosowania danego pliku.

### lib

W folderze tym znajdują się między innymi definicje kolekcji, dzięki czemu są one dostępne globalnie. Plik filters.coffee zawiera inicjalizację globalnego obiektu Filter, który służy do filtrowania danych pobieranych z MongoDB.

### server

Katalog ten odpowiada za część serwerową aplikacji i podobnie jak część kliencka, jest podzielona na moduły. W każdym module możemy znaleźć pliki z końcówką methods.coffee oznaczające skrypty z metodami obiektu Meteor, publish.coffee – funkcje publikujące dane z MongoDB, seed.coffee – kod do tworzenia przykładowych obiektów w bazie danych oraz functions.coffee – funkcje używane w innych częściach części serwerowej. Plik app.coffee jest głównym skryptem inicjalizującym. Ze względu na to, że MeteorJS w wersji 1.2.1 nie wspiera eksportowania i importowania obiektów z innych plików (wszystko jest globalne), to zaimplementowany został prosty system require/module.exports, podobny do tego używanego na platformie Node.JS. Dzięki temu uniknięto globalnego definiowania większości funkcji.

### tests

Folder ten zawiera pliki wykonujące testy. W katalogu unit znajdują się skrypty do wykonywania testów jednostkowych aplikacji klienckiej, posiadają one końcówkę spec.coffee. W e2e mieszczą się scenariusze do testów funkcjonalnych, posiadają w nazwie słowo kluczowe scenario. Folder jasmine/server zawiera skrypty wykonujące testy integracyjne serwera. Jasmie/client/integration powstał tylko w celu uruchomienia bliźniaczej aplikacji do testów funkcjonalnych.

### Pliki w katalogu głównym

.codeclimate.yml – plik konfiguracyjny dla platformy CodeClimate.com, która sprawdza między innymi pokrycie aplikacji testami oraz jakość kodu.

.gitingore – dokument, dzięki któremu możliwe jest ignorowanie wybranych folderów/plików przez repozytorium Git .

.travis.yml – plik konfiguracyjny dla platformy TravisCI.

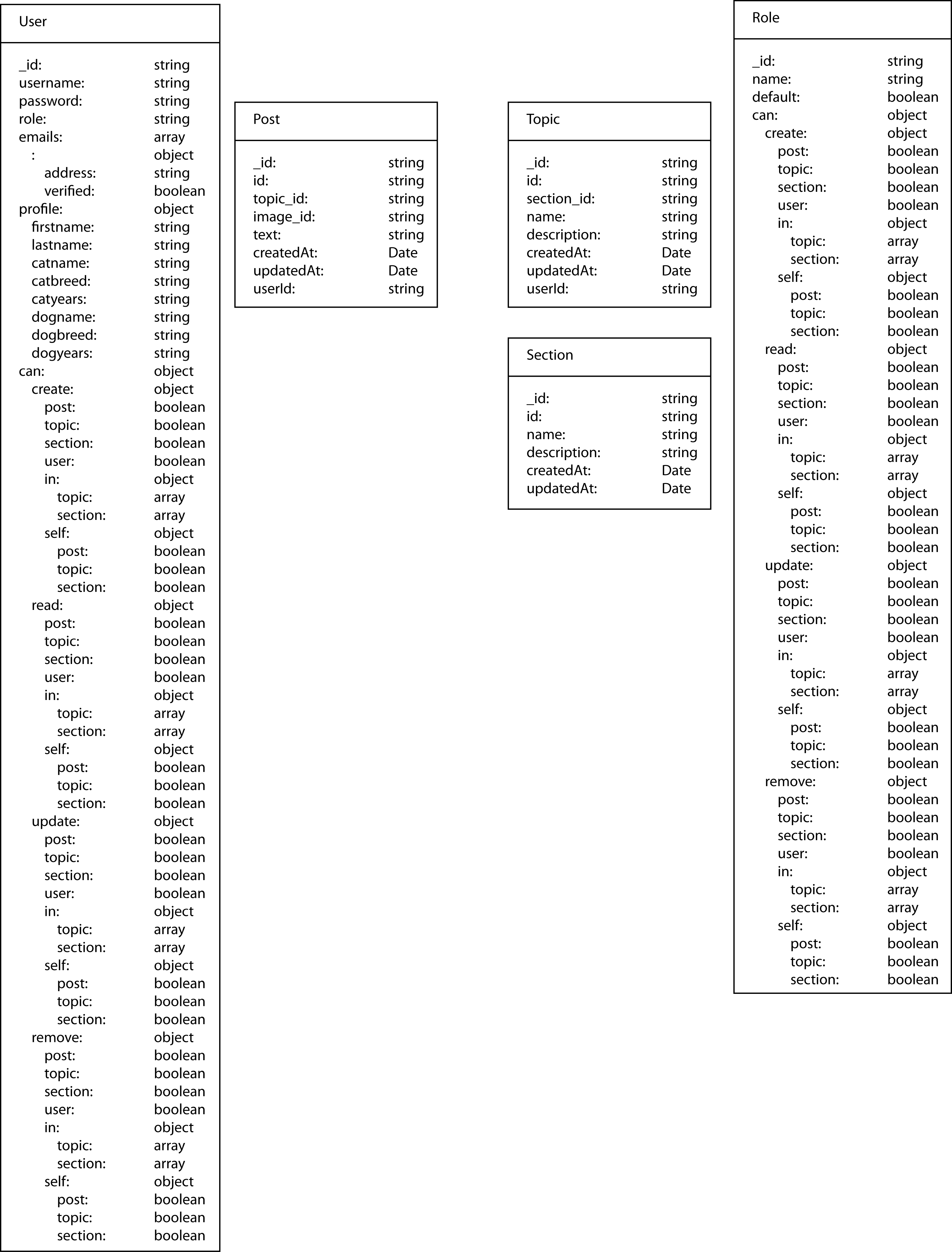
README.md – plik zawierający podstawowe informacje o projekcie.

build.variables.sh – prosty skrypt pomocny przy opisywaniu wersji aplikacji dostępnej na serwerze produkcyjnym/testowym.

deploy.exp – skrypt umożliwiający automatyczne wgranie aplikacji na serwer produkcyjny/testowy po pomyślnym przejściu testów na platformie TravisCI.

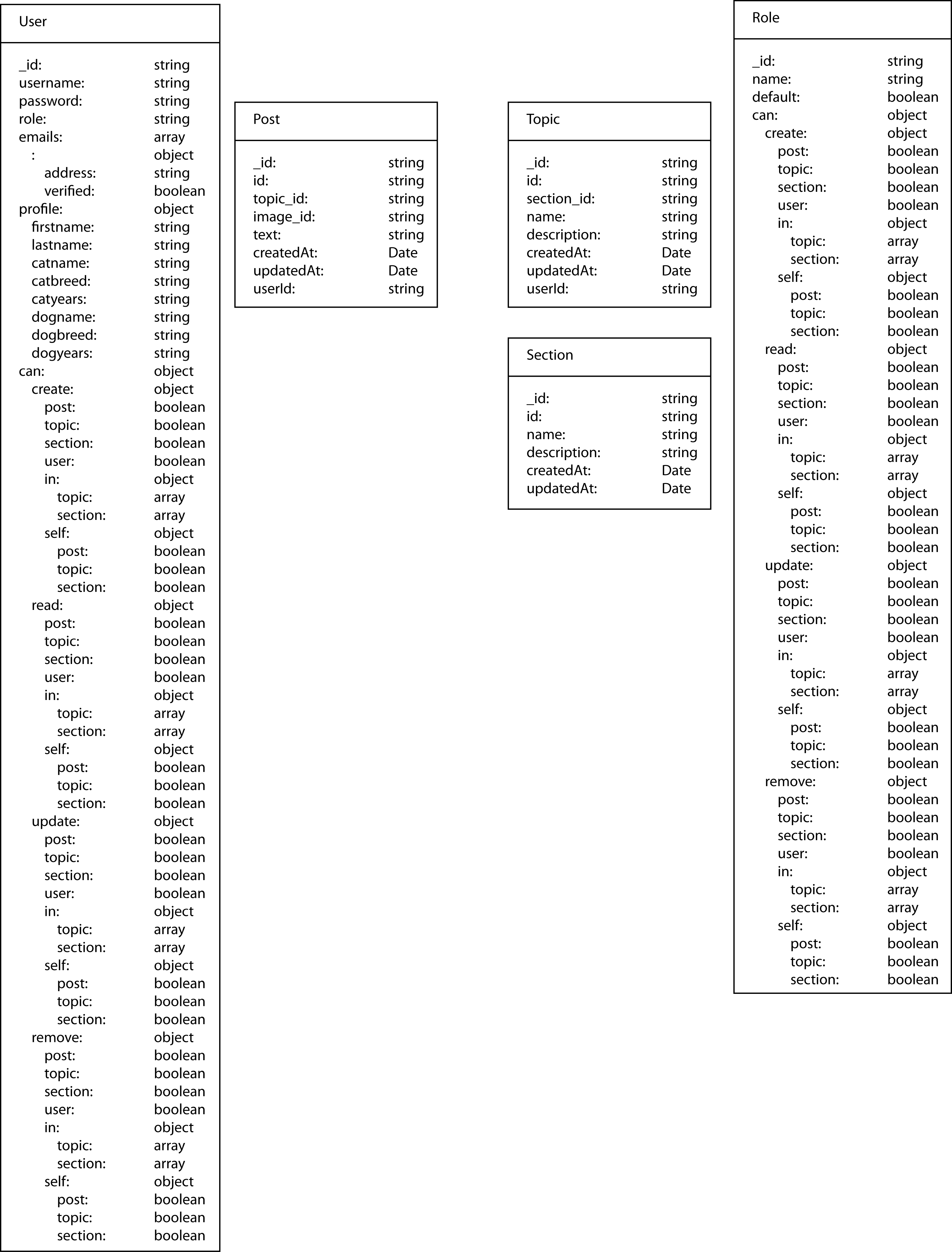
## Modele w bazie danych

### Post



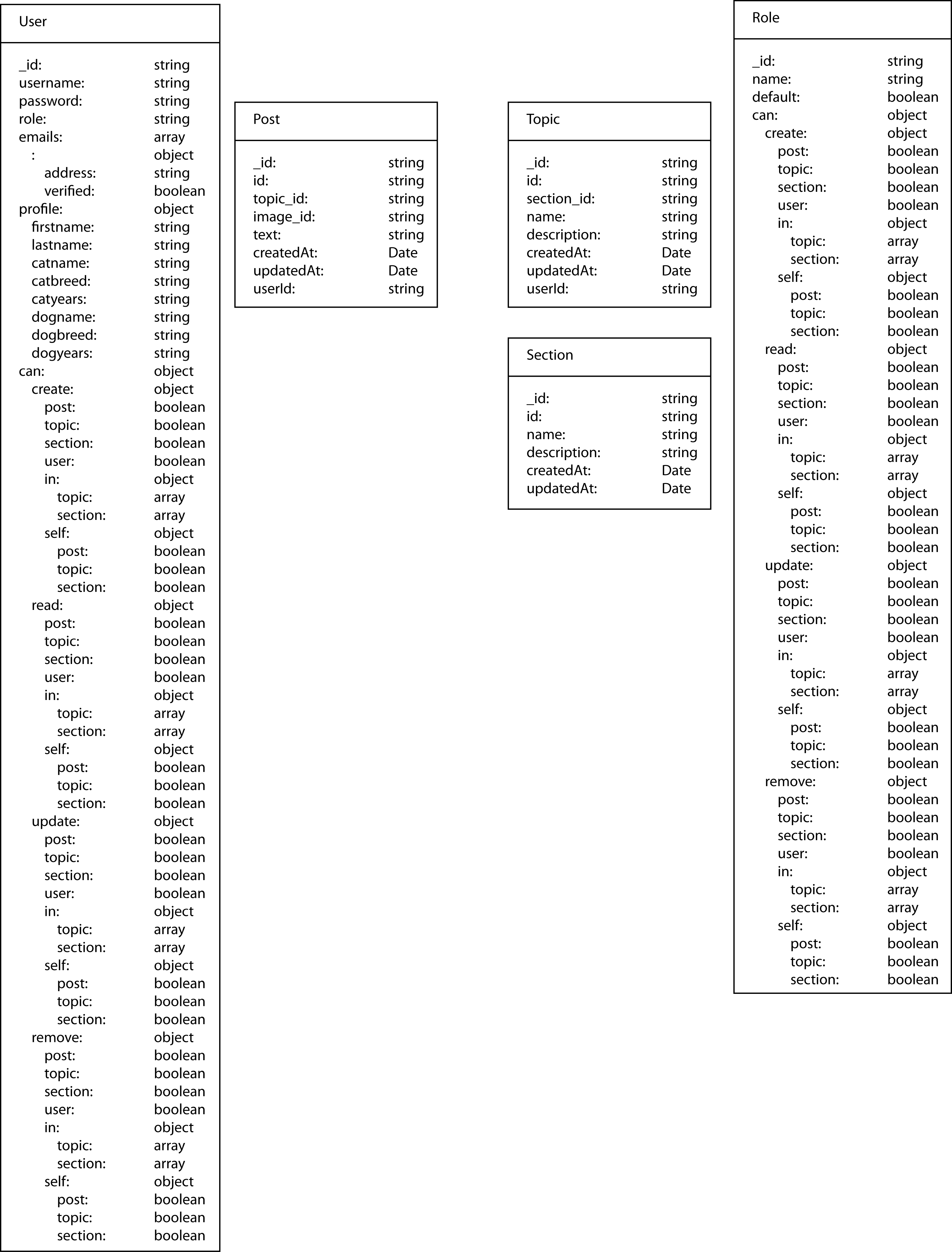
Rysunek 1. Schemat modelu postu w bazie danych

### Temat



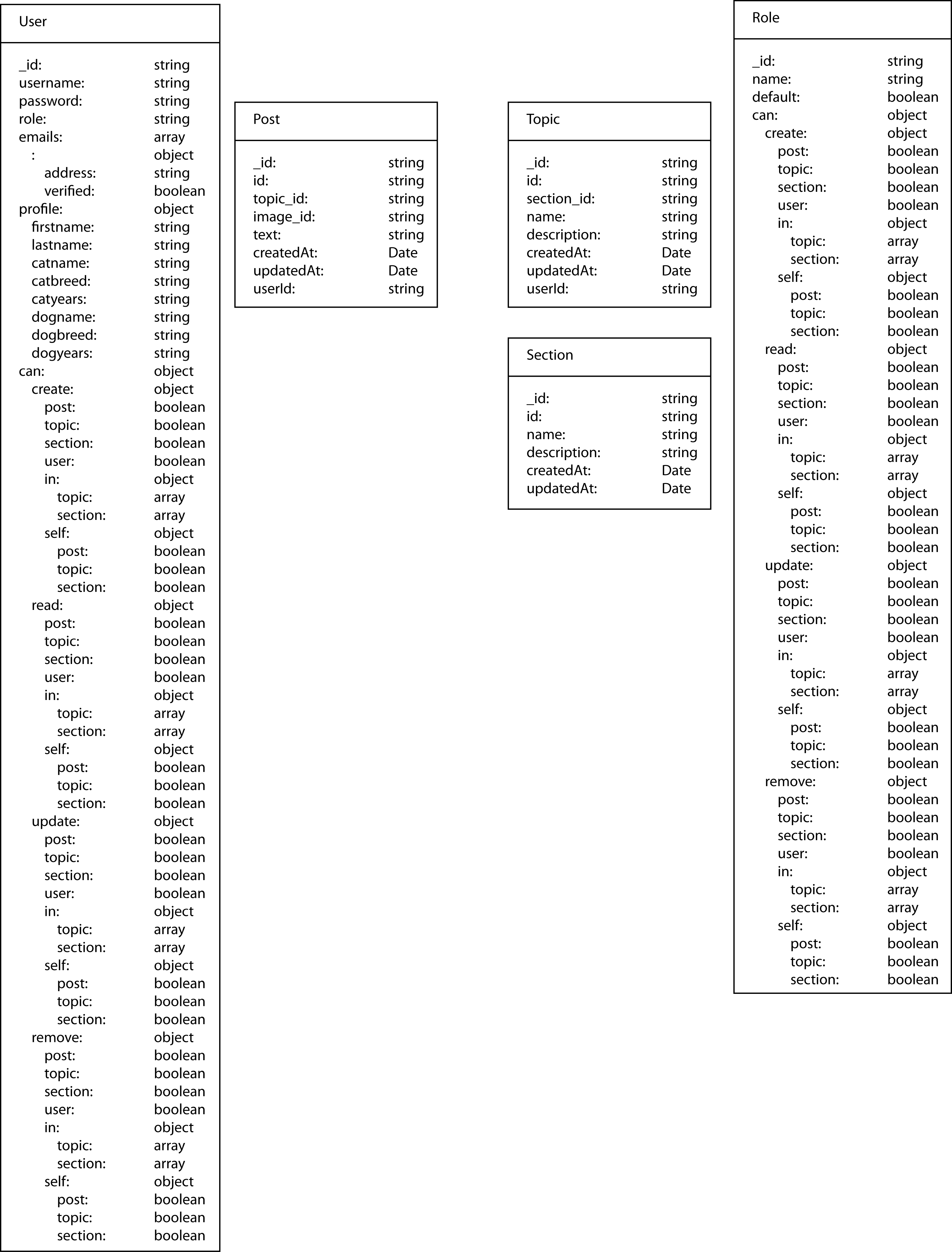
Rysunek 2. Schemat modelu tematu w bazie danych

### Sekcja



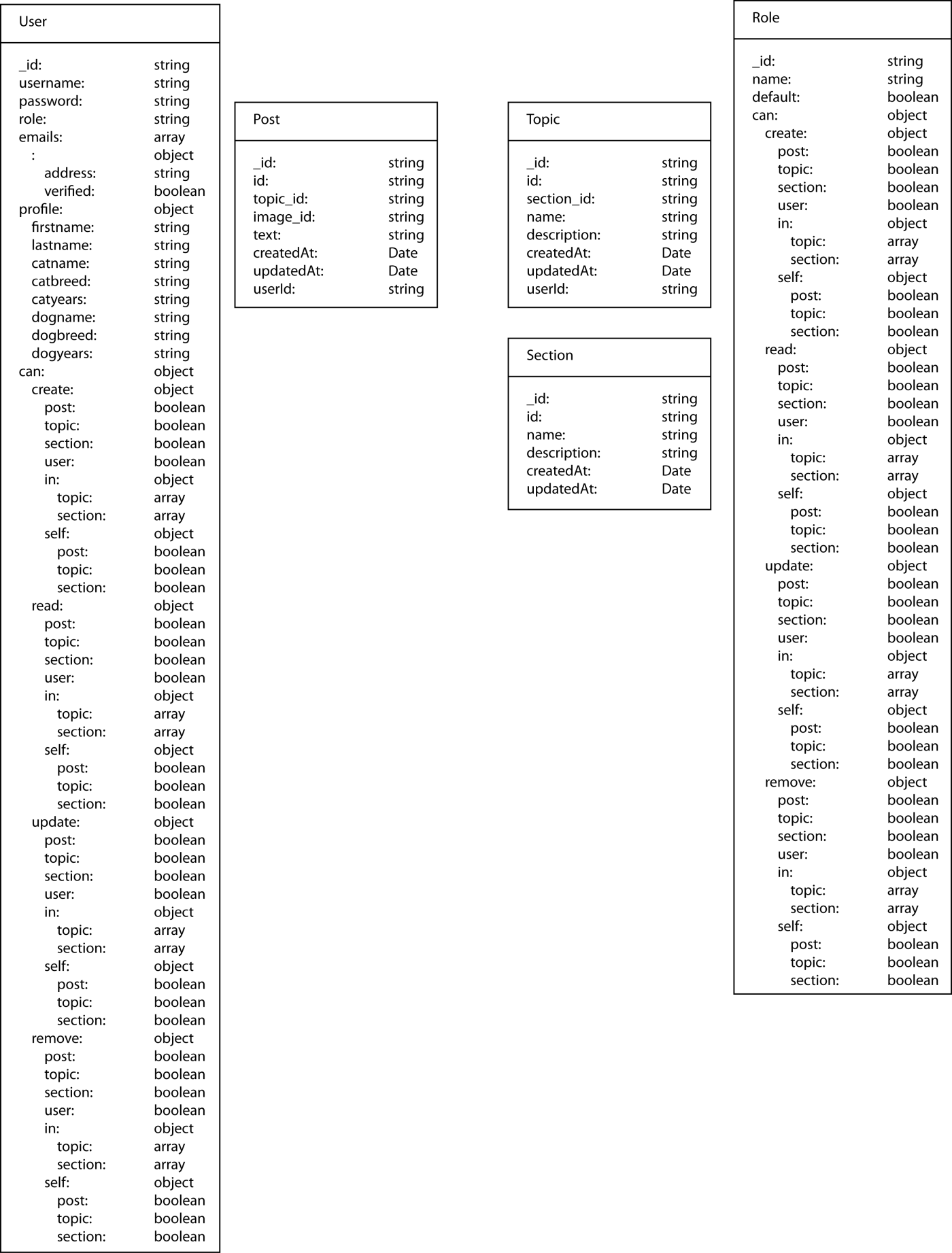
Rysunek 3. Schemat modelu sekcji w bazie danych

### Rola



Rysunek 5. Schemat modelu roli w bazie danych

### Użytkownik



Rysunek 4. Schemat modelu użytkownika w bazie danych

## Zasada działania aplikacji

# System uwierzytelniania i autoryzacji

## Rejestrowanie i logowanie użytkowników

## Role

## Struktura ról

## Panel administracyjny

# System ukrywania wiadomości

## 

# Testy

## Testy jednostkowe

## Testy integracyjne

## Testy funkcjonalne

# Zakończenie

# Literatura

1. Autorzy: Tytuł, Informacje wydawnicze, data
2. Autorzy: Tytuł, Informacje wydawnicze, data