POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: Informatyka (INF)
SPECJALNOŚĆ: Inżynieria systemów informatycznych (INS)

PRACA DYPLOMOWA INŻYNIERSKA

System zarządzania inteligentnym domem z wykorzystaniem Raspberry Pi oraz technologii internetowych.

Smart house management system using Raspberry Pi and Web technologies.

AUTOR:
Marcin Mantke

PROWADZĄCY PRACĘ:
dr inż. Marek Piasecki

OCENA PRACY:

WROCŁAW 2015

Spis treści

Rozdział 1

Wstęp

1.1 Ogólny opis projektu

1.2 Cel projektu

1.3 Wymagania

1.4 Ograniczenia

1.5 Przegląd istniejących rozwiązań

1.5.1 Domoticz.com

Od czasu popularyzacji rozwiązań pokroju Arduino i Raspberry Pi, hobbystyczne projekty inteligentnych domów są coraz częściej realizowane. Sprzyja temu fakt, że ceny podzespołów wymaganych do realizacji projektu są coraz niższe, a osoby zainteresowane mają coraz więcej literatury dostępnej w Internecie. Takimi właśnie hobbystami byli twórcy platformy *Domoticz*. Jest to zagraniczny serwis udostępniający multiplatformowe rozwiązania dla inteligentnych domów. Jest on skierowany głównie do hobbystów. Jak można przeczytać na stronie domowej projektu (`http://www.domoticz.com/`), *Domoticz* jest systemem automatyki domowej, który pozwala na monitorowanie i konfigurację urządzeń, takich jak: światła, przełączniki, różnego rodzaju sensory i mierniki, jak np temperatury, deszczu, wiatru, UV, prądu, gazu i wody.

Serwis ten udostępnia biblioteki umożliwiające podłączenie sensorów oraz oprogramowanie jednostki bazowej systemu (zwykle Raspberry Pi). Jako, że udostępniane są biblioteki, a nie tylko gotowe moduły sprzętowe, całość jest bardziej elastyczna. Oczywiście są tu ograniczenia, zarówno hardware'owe, jak i software'owe, lecz są one mniejsze niż w przypadku gotowych rozwiązań.

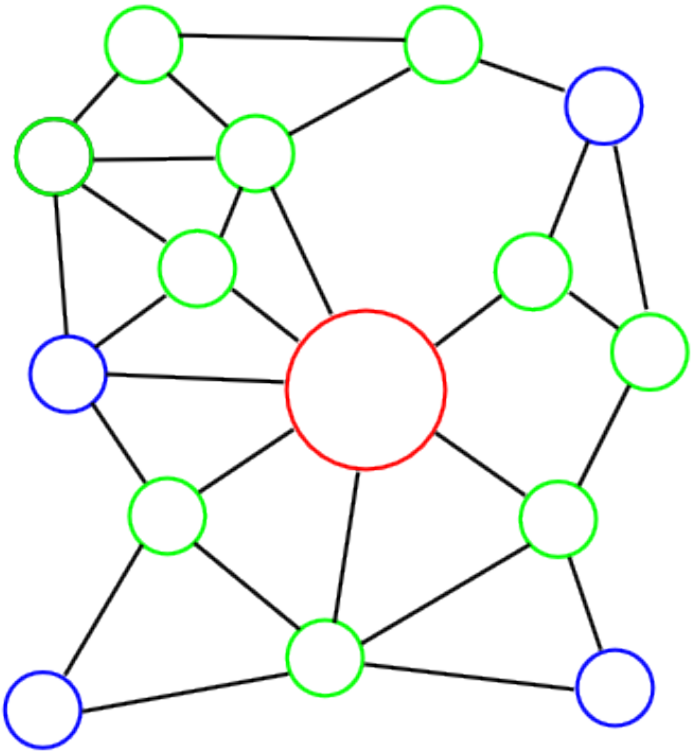
Samo oprogramowanie jest darmowe (Licencja GNU), ze strony producenta możliwy jest zakup urządzeń współpracujących z jego systemem, jednak możliwe jest również tworzenie sensorów we własnym zakresie, przy użyciu posiadanych podzespołów oraz udostępnionych bibliotek.

1.5.2 Fibaro

Rozwiązania oferowane przez firmę *Fibaro* mają odmienną filozofię od firmy *Domoticz*. Firma ta jest ukierunkowana na rozwiązania komercyjne. Oferuje ona systemy automatyki budynkowej, składające się z gotowych podzespołów, które trzeba jedynie zainstalować w budynku oraz skonfigurować.

System umożliwia dodawanie własnych scenariuszy działania. Ich definiowanie zbudowane jest na wzór programowania *Scratch* (`https://scratch.mit.edu/`)

System Fibaro opiera się na topologii sieci typu *mesh*. Urządzenia łączą się ze sobą za pośrednictwem protokołu *Z-wave*.



Rysunek 1.1: Schemat topoligii typu mesh.

1.6 Zarys koncepcji

Jednostka centralna Czujniki Aplikacja webowa Akcja - reakcja

Rozdział 2

Wybrane technologie

2.1 Hardware

2.2 Software

2.2.1 Aplikacja internetowa

Serwis internetowy wykonany został w oparciu o następujące technologie:

- **Ruby on Rails** - wnętrze (back-end) portalu webowego, które odpowiedzialne jest za wykonywanie określonych zadań na podstawie danych otrzymanych z fasady (front-end),
- **AngularJS** - fasada (front-end) - jej zadaniem jest komunikacja z użytkownikiem (odbieranie od niego danych oraz przekazywanie ich do back-end'u).

Ponadto wykorzystane zostały następujące elementy:

- **CoffeeScript** - język programowania kompilowany do JavaScriptu. Ponieważ CoffeeScript kompiluje się do JavaScriptu, programy mogą być krótsze o około $\frac{1}{3}$ bez strat dla szybkości działania,
- **Slim** – język oparty na szablonach (ang. template language), którego celem jest maksymalna redukcja składni HTML'owej poprzez usunięcie np. tagów zamykających. W efekcie otrzymujemy bardzo czytelny kod, oparty na wcięciach.

2.2.2 Baza danych

Jako system bazodanowy wykorzystany został MySQL w wersji 5.6. Niewątpliwymi zaletami tego systemu jest to, że posiada on wysoki stopień niezawodności, jest darmowy oraz powszechnie dostępny i wspierany. Dodatkowym atutem jest dostępność graficznego narzędzia usprawniającego korzystanie z bazy danych, jakim jest MySQL Workbench. Jest to aplikacja, dzięki której można połączyć się z kilkoma bazami danych (np. na serwerze produkcyjnym oraz lokalnej kopii roboczej), co w przy pracy z aplikacją webową, w której zmiany w bazie danych muszą być na bieżąco śledzone, a dane często wymagają bezpośredniej zmiany/aktualizacji, jest niezwykle pożyteczną funkcjonalnością.

Rozdział 3

Architektura systemu

3.1 Topologia systemu

https://pl.wikipedia.org/wiki/Topologia_gwiazdy

Rozdział 4

Specyfikacja systemu

Rozdział 5

Implementacja

Rozdział 6

Testy

6.1 Aplikacja internetowa

6.1.1 Testy jednostkowe

Do testów jednostkowych dla aplikacji webowej użyliśmy biblioteki RSpec, służącej do testowania kodu Rubiego, ze wsparciem dla frameworka Rails. Wybraliśmy tą bibliotekę, ponieważ wspiera ona idee Model-View-Controller, w oparciu o którą powstała nasza aplikacja. Dużą wygodą jest również automatyczna konfiguracja środowiska testowego, która polega m. in. na osobnej, testowej bazie danych, która jest tworzona na początku wykonywania testów i czyszczona po zakończeniu każdego przypadku testowege.

RSpec posiada bardzo intuicyjną składnie, przyjazna użytkownikowi naszym zdaniem. Testy podzielone są na 3 poziomową strukturę, każdy poziom zaczyna się słowem kluczowym:

1. „describe" – tu definiujemy jaką funkcje/klasę/moduł ma sprawdzać dany zbiór testów,
2. „context" – to miejsce służy do określenia warunków testu,
3. „it" – mówi nam jak powinien się zachować testowany moduł.

Przykładowy pojedynczy przypadek testowy:

Listing 6.1: RSpec test jednostkowy.

```
RSpec.describe TripsController, :controller do #
Describe'POST #create' do
context 'with valid attributes' do
it 'creates the trip' do
login_user
post :create, trip: attributes_for(:trip_with_path), format: :json
expect(Trip.count).to eq(1)
end
end
end
end
```

Powyższy test ma za zadanie sprawdzić czy dane wysłane w odpowiednim formacie zapytaniem http post, przekazane do funkcji „create" klasy „TripsController" stworzą w bazie danych nowy rekord w tabeli Trip.

Dla ułatwienia wprowadzania atrybutów do testowanych metod lub dodawania rekordów do bazy danych nie wykorzystując napisanych przez siebie metod posłużyliśmy się biblioteką „FactoryGirl", która implementuje wzorzec projektowy fabryki. Definicja obiektu jest bardzo prosta:

Listing 6.2: Definicja FactoryGirl.

```
FactoryGirl.define do
factory :full_trip, class: Trip do
distance 100
avg_rpm 2500
avg_speed 94
avg_fuel 8.9
date '2015-05-14T21:23:11.510Z'
mark 5.0
user_id 1
beginning 'Start'
finish 'Finish'
challenge_id nil
engine_type_id 1
engine_displacement_id 1
end
end
```

Poniżej znajduje się użycie metody "FactoryGirl.create", która tworzy obiekt w bazie danych. Ten przykład ilustruje również strukturę zbioru testów dla pojedynczej metody:

Listing 6.3: Użycie FactoryGirl.

```
describe 'GET #mytrips' do
context 'with valid attributes' do
let!(:login) { login_user }
let(:current_user_id) { login.id }
it 'renders all user\'s trips as json' do
FactoryGirl.create(:engine_displacement)
FactoryGirl.create(:engine_type)
FactoryGirl.create(:fuel_consumption)
FactoryGirl.create(:full_trip, user_id: current_user_id)
get :mytrips, format: :json
expect(response).to be_success
json = JSON.parse(response.body)
expect(json).not_to be_empty
expect(json[0].length).to eq(20)
end
it 'renders empty array when user has no trips' do
get :mytrips, format: :json
expect(response).to be_success
json = JSON.parse(response.body)
expect(json).to be_empty
end
end

context 'with user not logged in' do
it 'redirects to login page' do
FactoryGirl.create(:engine_displacement)
FactoryGirl.create(:engine_type)
FactoryGirl.create(:full_trip)
get :mytrips, format: :json
expect(response).to have_http_status(401)
end
```

```
end
end
```

Stworzyliśmy 19 testów jednostkowych. Badaliśmy pokrycie kodu aplikacji testami narzędziem Rcov. Tą analizę uruchamialiśmy po każdym commicie do repozytorium, dlatego wiemy jak się to kształtowało od moment skonfigurowania Rcov w Jenkinsie:

Rysunek 6.1: Wykres pokrycia kodu.

Ostatecznie pokrycie kodu źródłowego testami jednostkowymi wynosi około 75%. Rcov pokazuje również dokładnie, które linie kodu są testowane:

Rysunek 6.2: Raport Rcov dla pliku trips_controller.rb.

Rozdział 7

Podsumowanie

Bibliografia

[1] *Ruby on Rails Guides*, dostępne pod adresem: `http://guides.rubyonrails.org/`,
aktualne na dzień 14.06.2015r.

[2] *AngularJS Developer Guide*, dostępne pod adresem: `https://docs.angularjs.org/guide`, aktualne na dzień 14.06.2015r.