Zaawansowane technologie webowe

Dokumentacja projektu

Internetowy Chat

Filip Dziedzic
Informatyka
rok V 2013/2014
środa godz. 11.00

Spis treści

Założenia

Celem projektu było zaimplementowanie internetowego czatu z wykorzystaniem różnych technologii webowych. Głównym założeniem autora było maksymalne wykorzystanie technologii pozwalających na bezstanową oraz asynchroniczną obsługę klientów łączących się z serwerem aplikacji.

Zgodnie z obecnie obowiązującymi trendami, całość „logiki" warstwy prezentacji została zaimplementowana po stronie klienckiej (javascript). Takie rozwiązanie pozwala na odciążenie serwera aplikacyjnego kosztem przeglądarek internetowych klientów.

Wymagania funkcjonalne

Aplikacja powinna umożliwiać komunikację pomiędzy jej użytkownikami na zasadach czatu. Każdy podłączony do aplikacji użytkownik powinien posiadać swój własny identyfikator. W ramach aplikacji powinny być dostępne wydzielone miejsca zwane pokojami, w obrębie których prowadzona powinna być komunikacja pomiędzy użytkownikami. Wiadomość wysłana przez jednego z użytkowników powinna trafić do niego samego oraz innych użytkowników będących podłączonym do tego samego pokoju.

Aplikacja powinna pozwalać na przesyłanie „emot ikon", które powinny być wyświetlane w sposób graficzny.

Częścią czatu powinien być również inteligentny czat bot, który powinien potrafić interpretować przesyłane przez użytkowników wiadomości i odpowiadać na nie. W idealnym przypadku, użytkownik powinien nie być w stanie odróżnić odpowiedzi automatycznych udzielanych przez bota, od odpowiedzi, które przesłałby człowiek.

Architektura

Aplikacja została zaimplementowana w architekturze klient-serwer, pozwalającej na podłączanie się dowolnej[1] liczby klientów (użytkowników – przeglądarek internetowych) do jednego serwera.

Jak już zostało to wspomniane, całość logiki odpowiadającej za odpowiednie renderowanie widoku aplikacji zostało przeniesione do strony klienckiej, która komunikowała się z serwerem przy pomocy różnych metod protokołu HTTP, tj. operacji GET oraz POST. Żeby to było możliwe, serwer aplikacji musiał został zaimplementowany zgodnie z wzorcem architektury REST [1], tj. musiał udostępniać wszystkie operacje poprzez protokół HTTP (serwer został zaprojektowany zgodnie z architekturą RESTfull).

Warstwa logiki biznesowej

Aplikacja została zbudowana w oparciu o nowoczesny framework aplikacyjny Play framework [2] zaimplementowany w języku Scala [3], pozwalający na wykorzystanie go w aplikacjach pisanych w językach Java oraz Scala. Zastosowany framework pozwala na szybką budowę, wielce skalowalnych aplikacji. Pomimo, że framework poza stroną serwerową pozwala również na tworzenie warstwy prezentacji (poprzez szablony wykorzystujące elementy HTMLa oraz język Scala), ta funkcjonalność nie została wykorzystana. Bazą frameworka jest wzorzec projektowy MVC.

Konfiguracja API RESTowego odbywa się przy pomocy specjalnego pliku *routes*. Przykładowe

1 Liczba użytkownik możliwa do obsłużenia w rozsądnym czasie zależy oczywiście od mocy fizycznego serwera, na którym uruchomiony został serwer aplikacji. Pobieżne testy wydajnościowe wykazały, że serwer aplikacji był w stanie obsłużyć około 600 zapytań na sekundę na maszynie z jednym procesorem wirtualnym oraz 1GB pamięci RAM.

wpisy wykorzystane w aplikacji wyglądają następująco:

```
# Home page
GET     /                           controllers.ChatApplication.index
GET     /chatFeed/:room             controllers.ChatApplication.chatFeed(room: String)
POST    /chat                       controllers.ChatApplication.postMessage
```

Akcje kontrolera są to funkcje przetwarzające obiekty typu *Request* w obiekty typu *Response* (wykorzystanie możliwości języka Scala, gdzie funkcje są typami pierwszoklasowymi – first class citizen). Przykładowa akcja wygląda następująco:

```
/** Controller action serving chat page */
def index = Action {
  Ok(views.html.index("Chat with Server Sent Events - Advanced Web Programming"))
}
```

Do rozsyłania przesyłanych wiadomości została wykorzystana biblioteka *Iteratee* [4] wchodząca w skład frameworka. Pozwala ona na asynchroniczne przetwarzanie porcji (ang. chunk) danych. Przykładowo przy przesyłaniu dużego pliku, możliwe zapisywanie kolejnych porcji pliku, tak aby w przypadku zerwania połączenia, klient mógł rozpocząć wysyłanie pliku w momencie, w którym nastąpiło zerwanie połączenia. W przypadku czatu, porcją danych jest pojedyncza wiadomość, która jest rozsyłana do wszystkich podłączonych użytkowników. Rozsyłanie wiadomości do klientów zaimplementowane zostało przy pomocy technologii Server-Sent Events [5], wykorzystującej protokół HTTP (w przeciwieństwie do WebSocketów), pozwalającej na przesyłanie komunikatów od serwera do klienta.

Utworzenie obiektów potrzebnych do dodawania oraz rozsyłania wiadomości wygląda następująco:

```
/** Central hub for distributing chat messages */
val (chatOut, chatChannel) = Concurrent.broadcast[JsValue]
```

W momencie otrzymania nowej wiadomości jest ona dostarczana do obiektu *chatChannel* będącego swoistym kanałem przekazującym wiadomości dalej do obiektu *chatOut*. Obiekt *chatOut* można przyrównać do Enumeratora, który iteruje po wszystkich wiadomościach, które zawiera i przesyła je dalej do za-subskrybowanych klientów.

„Wkładanie” wiadomości do kanału czatu:

```
chatChannel push Json.toJson(escaped)
chatChannel push ChatBot.talk(msg.room, msg.msg)
```

Przesyłanie wiadomości w odpowiedzi na zapytanie pod adresem zbindowanym do akcji *chatFeed* kontrolera. Symbol &> jest to metoda pozwalająca na przesłanie porcji danych (wiadomości) z jednego kanału do drugiego, w tym wypadku z kanału czat do *EventSource* (poprzez filtr, który pozostawia wiadomości przeznaczone dla konkretnego pokoju czata).

```
/** Controller action serving activity based on room */
def chatFeed(room: String) = Action {
  Ok.stream(chatOut &> filter(room) &> EventSource()).as("text/event-stream")
}
```

Chat bot

Bot został zaimplementowany z wykorzystanie meta języka AIML [6] zaprojektowanego właśnie z myślą o czat botach oraz innych aplikacjach wykorzystujących sztuczną inteligencję.

W aplikacji został wykorzystany interpreter meta języka AIML udostępniony publicznie pod adresem http://www.pandorabots.com/. Serwis ten pozwala na wgranie własnej definicji bota oraz udostępnienie go poprzez usługi sieciowe.

Przykładowa kategoria wiedzy wykorzystana w czat bocie wymuszająca odpowiedź *„It's my native planet”* na wiadomość rozpoczynającą się od słowa *„EARTH”* wygląda następująco:

```
<category>
    <pattern>EARTH *</pattern>
    <template>It's my native planet.
    </template>
</category>
```

Warstwa prezentacji

Początkowo klient (warstwa prezentacji) została zaimplementowana z wykorzystaniem meta języka HTML, szablonów CSS oraz JavaScriptu (poprzez użycie biblioteki Vaquery). Podstawowymi funkcjami, które zostały zrealizowane przy pomocy JavaScriptu były kolejno:

- wysyłanie komunikatów POST do serwera w celu dodania nowej wiadomości

```
function sendMsg(msg, time) {
    room = $('#roomSelect').val();
    if (validRoom(room)) {
        json = {
            room: room,
            user: $('#user').val(),
            msg: msg,
            time: dtFormat(time)
        }
        console.log(json);
        $.ajax({
            url: '/chat',
            type: "POST",
            data: JSON.stringify(json),
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: function (data) {
                $('#msg').val('');
                console.log('success');
            }
        })
    }
}
```

- otworzenie źródła zdarzeń (EventSource) w celu pobierania kolejnych wiadomości z serwera

```
/** start listening on messages from selected room */
function listen(room) {
    console.log(room);
    chatFeed = new EventSource("/chatFeed/" + room);
    chatFeed.addEventListener("message", addMsg, false);
};
```

- obsługa funkcjonalności drag and drop

```
function emotStart(e) {
    e.dataTransfer.effectAllowed = 'copy';
    e.dataTransfer.setData('emot', this.id);
};

function chatOver(e) {
    if (e.preventDefault) e.preventDefault();
    e.dataTransfer.dropEffect = 'copy';
}

function chatDrop(e) {
    if (e.stopPropagation) e.stopPropagation();
    ...
}
```

W przypadku tej implementacji klienta, dodawanie kolejnych wiadomości polegało na bezpośrednim manipulowaniu elementami drzewa DOM dokumentu HTMLowego. Ostatecznie aplikacja kliencka została zmodyfikowana, tj. do jej budowy został wykorzystany framework AngularJS [7]. Pozwoliło to na odejście od manipulacji drzewem DOM, w przypadku zmiany wartości zmiennych w kontrolerze aplikacji, odpowiednie widoki zostają automatycznie zaktualizowane. Ta sama sytuacja ma miejsce w przypadku aktualizacji widoku – automatycznie aktualizowany jest również kontroler. Sam framework oparty jest o wzorzec MVC, co niejako wymusza pisanie przejrzystych aplikacji (a przynajmniej bardzo to ułatwia).

Formularz do wysyłania nowych wiadomości oparty o AngularJS wygląda następująco:

```
<div id="footer">
    <input ng-model="msg" type="text" name="chat" id="msg" placeholder="Say something" class="input-block-level" required="" />
    <input type="submit" class="btn btn-primary" id="sendMessage" value="Send"/>
</div>
```

W przypadku każdej zmiany przechowywanej wartości w elemencie od *id=”msg”* automatycznie aktualizowana jest zmienna *msg* kontrolera. Wynika to z dodatkowego atrybutu, nieistniejącego w czystym HTMLu, *ng-model=”msg”*.

Interfejs aplikacji

Interfejs aplikacji zawiera 4 główne elementy:

1. menu wyboru pokoju oraz imienia aktualnego użytkownika,
2. główną część czatu wyświetlającą wiadomości,
3. panel emot ikon, które można „przeciągnąć” i upuścić na część wiadomości – spowoduje to wysłanie „buźki” do innych użytkowników oraz
4. pole tekstowe z przyciskiem „*Send”*, służące do wysyłania nowych wiadomości.

Całość została przedstawiona na zrzucie ekranu poniżej.

Your name: Joe | Room 1

Chat bot: Hi there!

---

at: 2014-01-22 09:29:25
Joe: How are you today?

---

at: 2014-01-22 09:29:26
Chat bot: Ah. I am functioning within normal parameters.

---

at: 2014-01-22 09:29:34
Joe: Great!

---

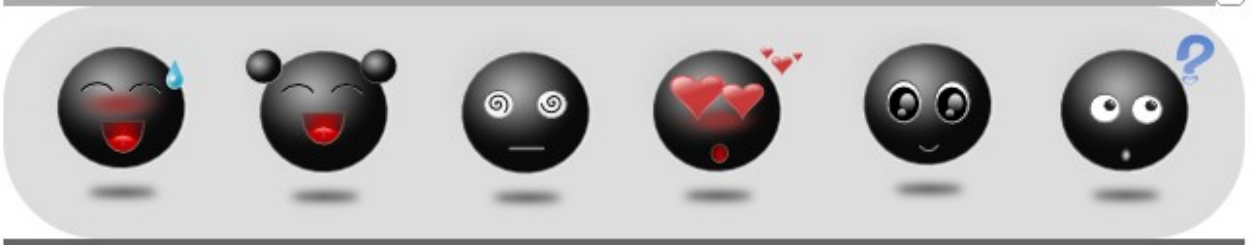
at: 2014-01-22 09:29:35
Chat bot: I'm glad you liked it .

---

at: 2014-01-22 09:29:41
Joe: 

---

at: 2014-01-22 09:29:42
Chat bot: I am so glad you find this amusing, .



Say something | Send

Lista wykorzystanych technologii

Z wykładu

- HTML5: drag and drop, server-sent events
- XML
- Jquery
- CSS: rounded corners
- DOM tree

Inne technologie

- Język Scala
- Framework Play framework
- JSON
- REST
- Framework AngularJS

Źródła

[1] http://en.wikipedia.org/wiki/Representational_state_transfer

[2] http://www.playframework.com/

[3] http://www.scala-lang.org/

[4] http://www.playframework.com/documentation/2.2.x/Iteratees

[5] http://en.wikipedia.org/wiki/Server-sent_events

[6] http://pl.wikipedia.org/wiki/AIML

[7] http://angularjs.org/