Instant Messaging service made as a RESTful Web Service

Juan Luis Gorricho

RESTful web services

- Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP and WSDL based Web services deploying services on the Internet, due to its lightweight nature and the ability to transmit data directly over HTTP.
- The RESTful architecture is focused on managing resources instead of the usual RPC implementation.
- The HTTP commands commonly used are: GET, POST, PUT and DELETE to materialize the usual CRUD actions on resources.

RESTful web services

- Representational State Transfer means that the state of the user is transferred along the same HTTP request as a representational form.
- Consequently, statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all information necessary for the server to fulfill that request.

RESTful web services

- By stateless it means that the **web server** does not store any state about the client.
- That is where the *ST* in *REST* comes from, *State Transfer*. You transfer the state around, instead of having the server store it. **This is the only way to scale to millions of users.**
- The load of session management is amortized across all the clients, the clients store their session state and the servers can service many orders of magnitude or more clients in a stateless fashion.

JSON

- JSON or JavaScript Object Notation, is an open standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs.
- It is used primarily to transmit data between a server and a web client application, as an alternative to XML.
- Although originally derived from the JavaScript scripting language, JSON is a language-independent data format.
- Code for parsing and generating JSON data is readily available in many programming languages.

GSON

- Gson (also known as Google Gson) is an open source Java library to serialize and deserialize Java objects to (and from) JSON data structures.

RESTful web services and NetBeans

- In order to understand the programming of RESTful web services with NetBeans we will create a new web app.
- Define, as usual, an "entity" package.
- From that package, right-click on the mouse and select the entry: "Restful Web Services from Database". Create a new Data Source using the "ecommerce" ddbb connection.
- Try to understand all the code that has been generated by the NetBeans assistant.

Programming with WebSockets

- Snippet of code for servers:

```
@ServerEndpoint("/ws")
public class WebSocketServer {

  private static Set<Session> clients = new HashSet<Session>();

  @OnOpen
  public void onOpen(Session session) {...}

  @OnClose
  public void onClose(Session session) {...}

  @OnMessage
  public void onMessage(String message, Session session) {...}

  session.getBasicRemote().sendText(json);                    }
```

Programming with WebSockets

- Snippet of code for clients:

```
@ClientEndpoint
public class WebSocketClient {

static Session session;

public static void newInstance() {
    WebSocketContainer container = ContainerProvider.getWebSocketContainer();
    session = container.connectToServer(WebSocketClient.class,
      URI.create(Cons.SERVER_WEBSOCKET)); }

@OnMessage
public void onMessage(String message) {...}

session.getBasicRemote().sendText(json);                    }
```

Second approach

- This is a second approach implementing an instant messaging service, a remote approach without persistence.
- This approach focuses on using a web application service and accessing the service using a REST API.
- The approach uses Swing clients instead of Internet browsers, and the **asynchronous** behavior will be implemented with a pushing strategy using a WebSockets library. The client has no persistence either.

Exercise

- Open the **InstantMessagingRemote_server** web application project. Import the GSON library from the libs folder if necessary.
- Add the code from the previous exercise implementing: **PublisherImpl** and **TopicManagerImpl**, be careful not to delete some extra methods.
- Run the web application project.

Exercise

- Try to understand the meaning of all the code contained in the web app. project.
- In particular, review:
  - The two FacadeREST included in the facadeREST package. Review the syntax specifying the URL paths to be used to execute the different methods. Review the use of an instance of Global, with application scope, to retrieve the instance of TopicManager.

Exercise

- In particular, review:
  - The programming of WebSocketServer where the server only accepts subscription requests from its clients to subscribe or unsubscribe that client to a topic.
  - The programming of SubscriberImpl where we use a WebSocket session to asynchronously send messages to the client.

Exercise

- Open the InstantMessagingRemote_client_exercise.
- Try to understand the coding of the Java classes included in apiREST as an example of the usual programming of a REST API. Notice all methods are programmed as **static** methods.
- Copy and paste the code of the SwingClient Java class from previous version.
- Complete the code of PublisherStub and TopicManagerStub, basically, to execute remote calls using the apiREST; except for the subscribe() and unsubscribe() methods where you have to use the WebSocketClient methods, why?

Exercise

- Complete the code of the WebSocketClient in accordance with the programming of the WebSocketServer and the SubscriberImpl at the server side.
- Review the schema of classes of the following slide for a better understanding of the exercise.
- Run separately two clients with different user names to test the code.

Schema of classes for a remote service