

**Ingeniería en Sistemas**

**Computacionales**

**Raúl Tostado Blázquez I.D. 145901**

**Materia: Sistemas Distribuidos**

**Nombre del Trabajo: Práctica Play-WebServices**

**Profesora: Ofelia D. Cervantes Villagómez**

**San Andrés Cholula, Puebla a**

**08 de abril de 2015**

Reporte práctica Play-WebServices

**Introducción a la práctica**

Play es un framework de código abierto para el desarrollo de aplicaciones Web bajo el uso estricto del patrón modelo-vista-controlador (MVC). Play es muy competente para el desarrollo de servicios Web debido a su uso por defecto de servicios RESTful, completa integración con Json y la forzosa implementación de MVC. Debido a esto Play permite que bajo su arquitectura los servicios Web sean muy fáciles de implementar.

**Antes de comenzar:**

El primer paso fue dar una lectura detallada a la práctica para saber cuál es el contenido de la misma, posteriormente se descargaron como Zip los programas con los que se trabajó y otros complementos necesarios para trabajar.

**Actividad 1 - Tutorial de Servicios Web**

**Activator** es una plataforma que incluye diversas herramientas de desarrollo como **Play** y **Sbt**, otorga soporte para **Scala** y **Java** además de ofrecer un ambiente de pruebas unitarias bastante flexible.

* Para crear un nuevo proyecto basta con ejecutar en la consola el comando **new**:

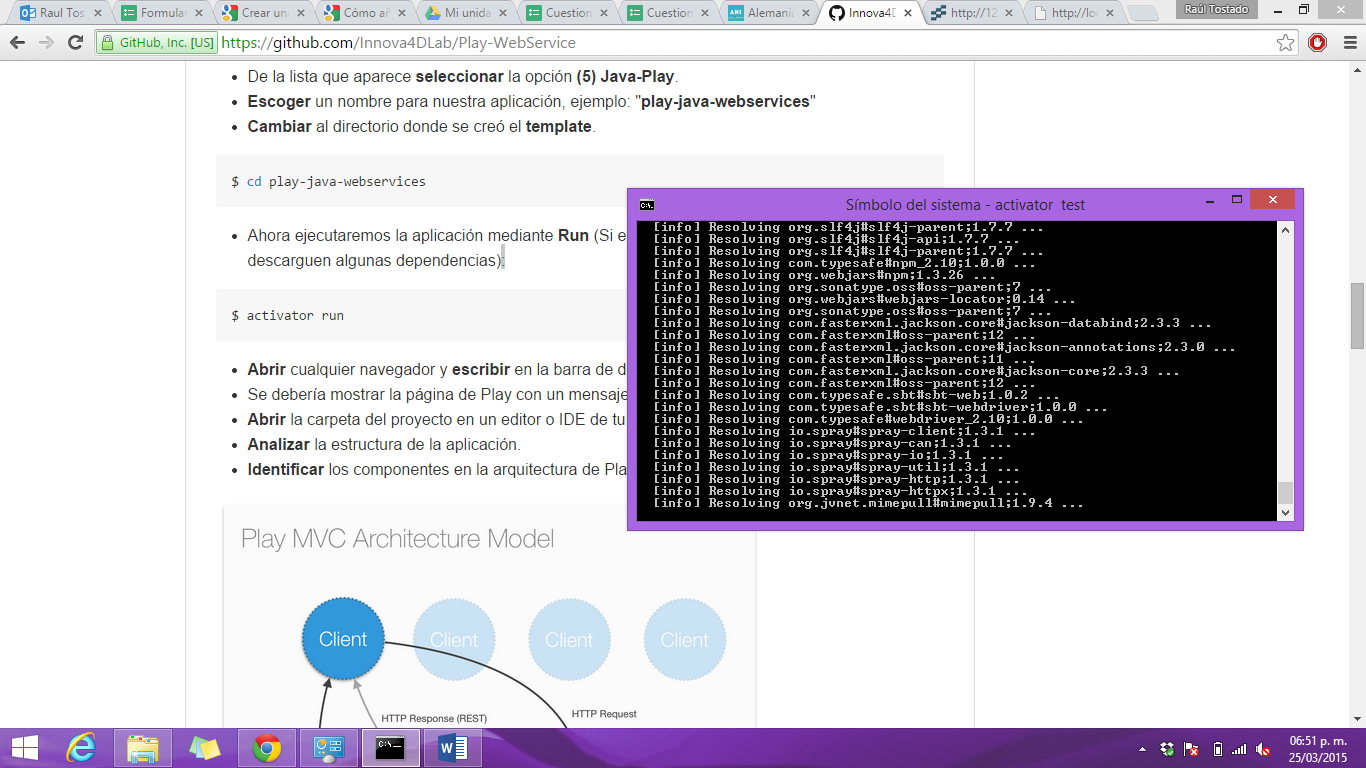
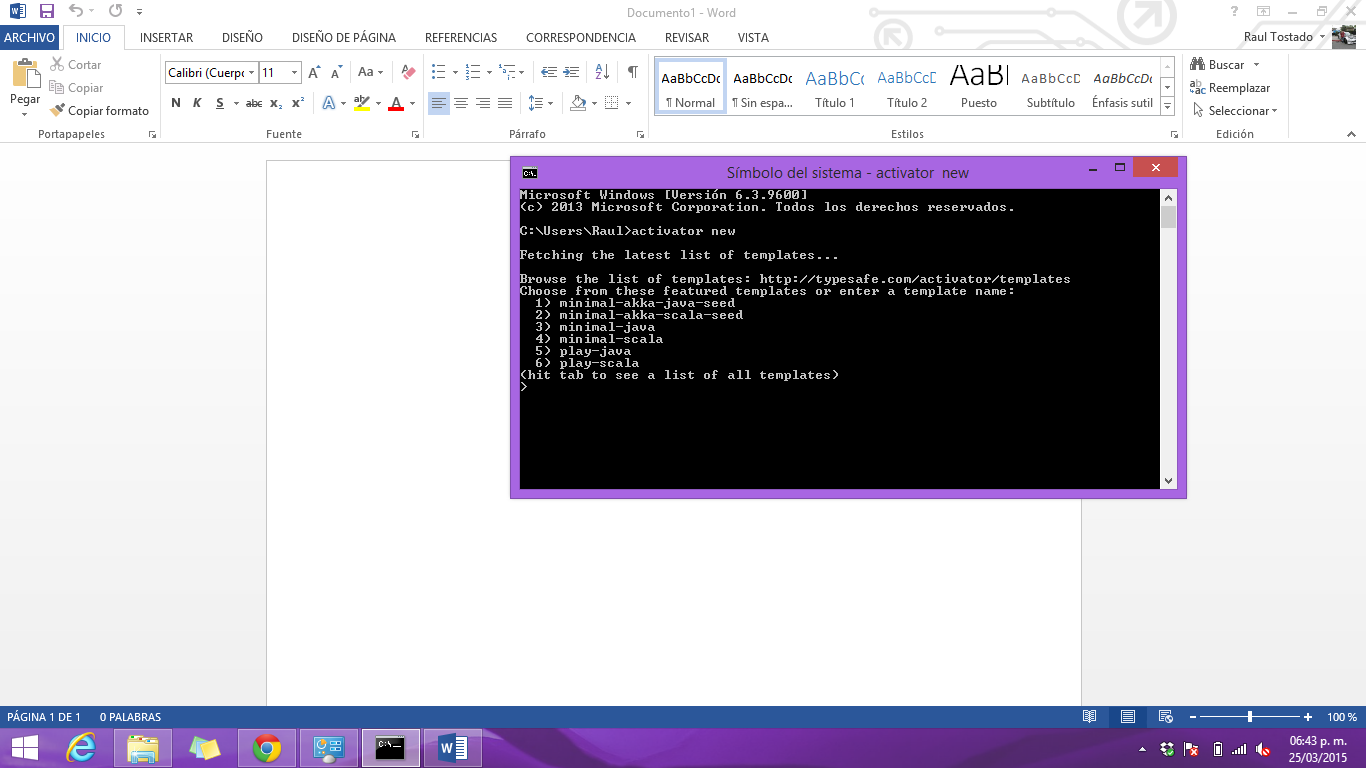
$ activator new

* De la lista que aparece **seleccionar** la opción **(5) Java-Play**.
* **Escoger** un nombre para nuestra aplicación, ejemplo: "**play-java-webservices**"
* **Cambiar** al directorio donde se creó el **template**.

$ cd play-java-webservices

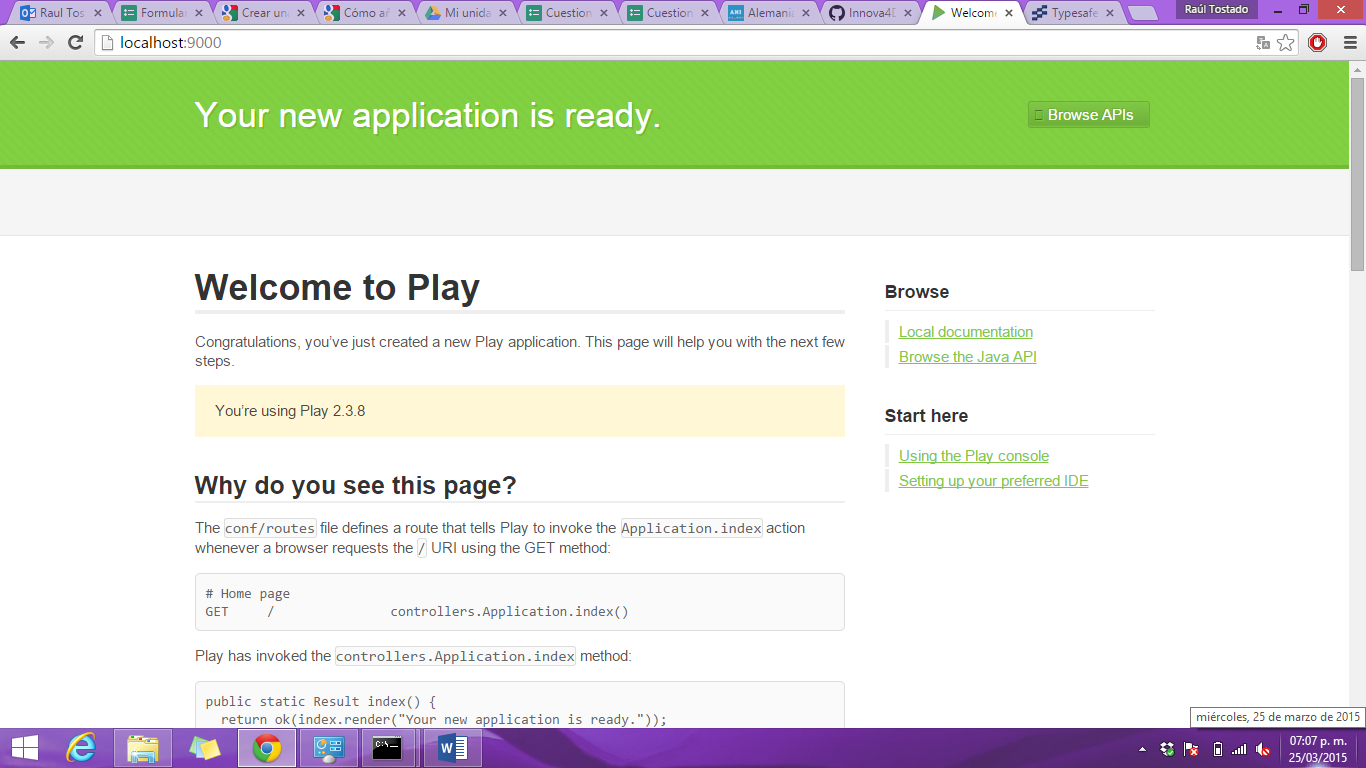
* Ahora ejecutaremos la aplicación mediante **Run** (Si es la primera vez, es posible que se descarguen algunas dependencias):

$ activator run

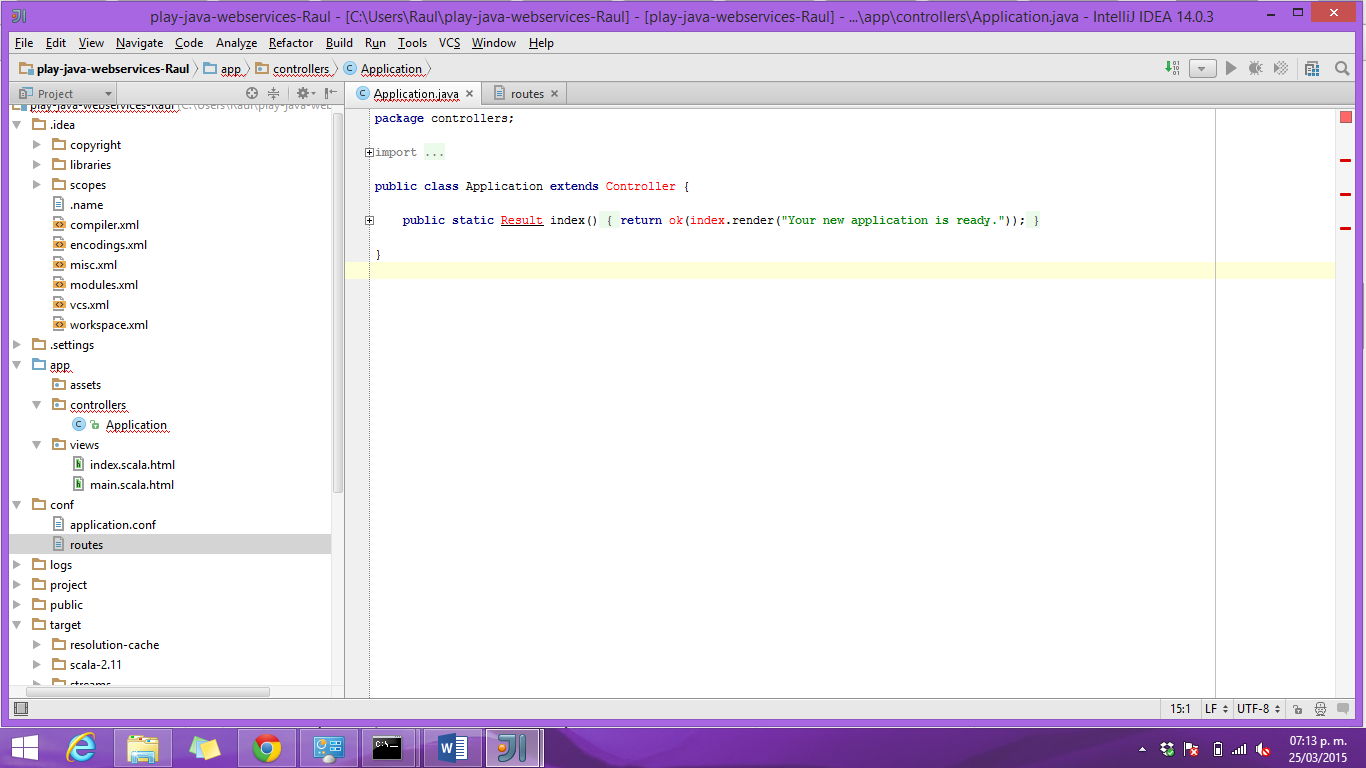


R= Primero se descargaron todos los complementos y se inició el procedimiento para poder levantar el servidor

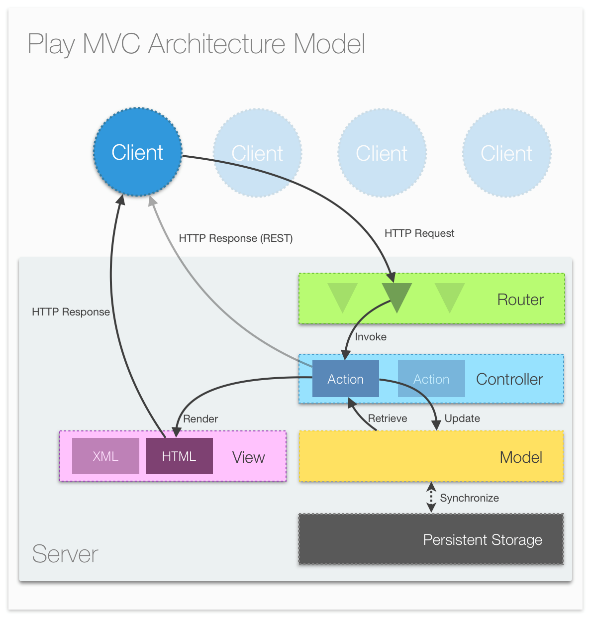
* **Abrir** cualquier navegador y **escribir** en la barra de direcciones **"localhost:9000"**
* Se debería mostrar la página de Play con un mensaje. **"La aplicación está lista"**.



* **Abrir** la carpeta del proyecto en un editor o IDE de tu preferencia.



* **Analizar** la estructura de la aplicación.
* **Identificar** los componentes en la arquitectura de Play:



* Para continuar es importante identificar las carpetas del proyecto **"App -> Controllers"** y **"conf"**, ambas carpetas aparecen en root del proyecto.
* Dentro de **"conf"** existe un archivo llamado **routes**, dada la arquitectura de Play, analiza su contenido y explícalo en tu reporte.
* Dentro de **"App -> Controllers"** existe una clase llamada **Application**, la cual es una clase en donde podemos definir la lógica **HTTP** de la aplicación, aquí es donde se van a definir los **Servicios Web**.
* Dentro de éste archivo edita la siguiente línea:

/\*

\* El siguiente método solamente despliega

\* un mensaje dentro del template index.

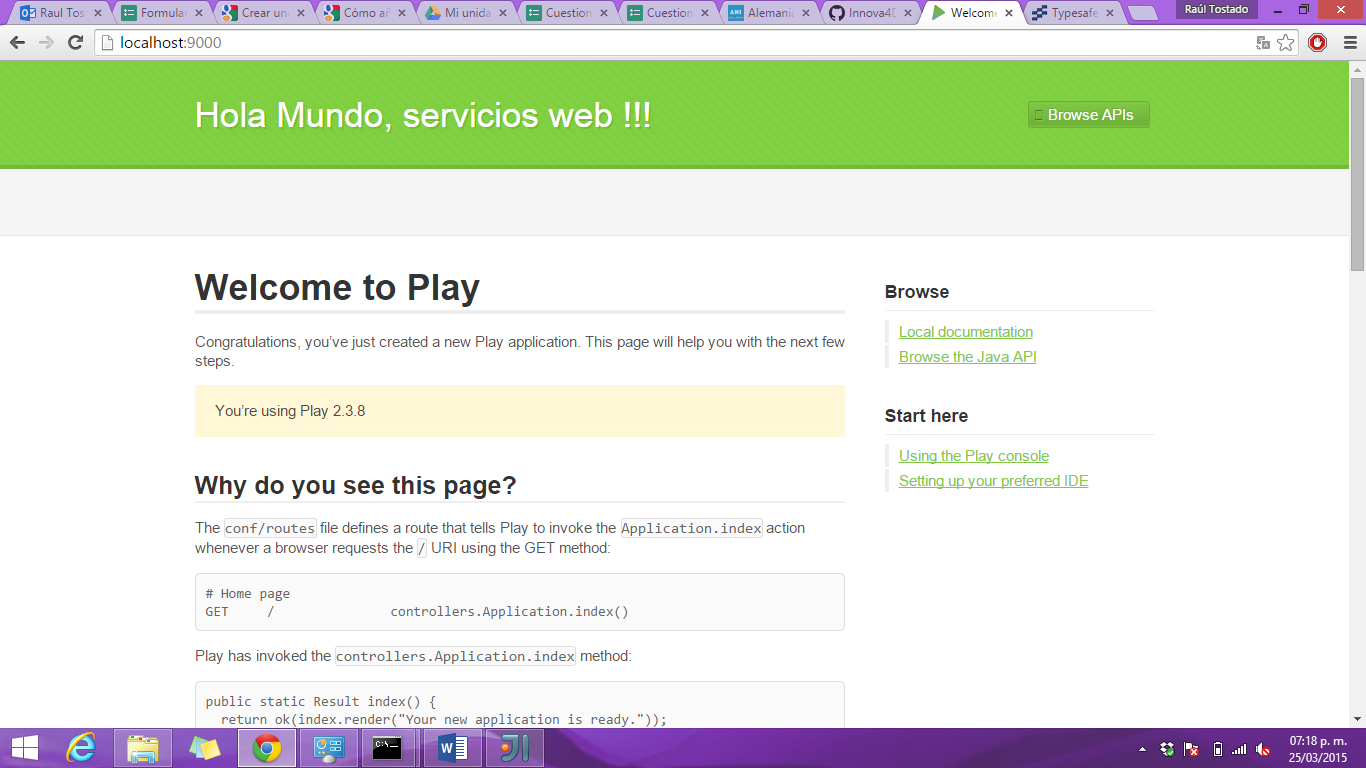
\*/

public static Result index() {

return ok(index.render("Hola mundo, servicios web!!"));

}

* Guarda el archivo y actualiza **localhost:9000** en el navegador.



* Ahora procedemos a realizar nuestro **servicio Web**, dentro de la clase **Application** necesitamos importar algunas librerías:

import play.libs.Json;

import com.fasterxml.jackson.databind.JsonNode;

import com.fasterxml.jackson.databind.node.ObjectNode;

import play.api.libs.json.\*;

* Agregamos un método dentro de la clase **Application**:

/\*

\* El siguiente método hace una multiplicación de enteros.

\* @param a el primer entero.

\* @param b el segundo entero.

\* @return Resultado en formato Json.

\*/

public static Result getMultiplicacion(int a, int b){

ObjectNode result = Json.newObject();

result.put("resultado", a\*b );

return ok(result);

}

* Ahora ya tenemos la lógica de nuestro servicio **multiplicación**, pero para ponerlo en funcionamiento debemos ir a **routes** y agregar el método.
* **# Mis Servicios Web** es una línea de comentario para indicar el bloque donde inician las declaraciones de los servicios Web.
* El método multiplicación es de tipo **GET** por lo que definimos una ruta con nombre**/multiplicación** y dos variables, **:a** y **:b**
* Por último, le indicamos la ubicación del método: **controllers.Application.getMultiplicación**pasamos las variables al método mediante **(a:Int, b:Int)**. El código se ve así:

# Routes

# This file defines all application routes (Higher priority routes first)

# ~~~~

# Home page

GET / controllers.Application.index()

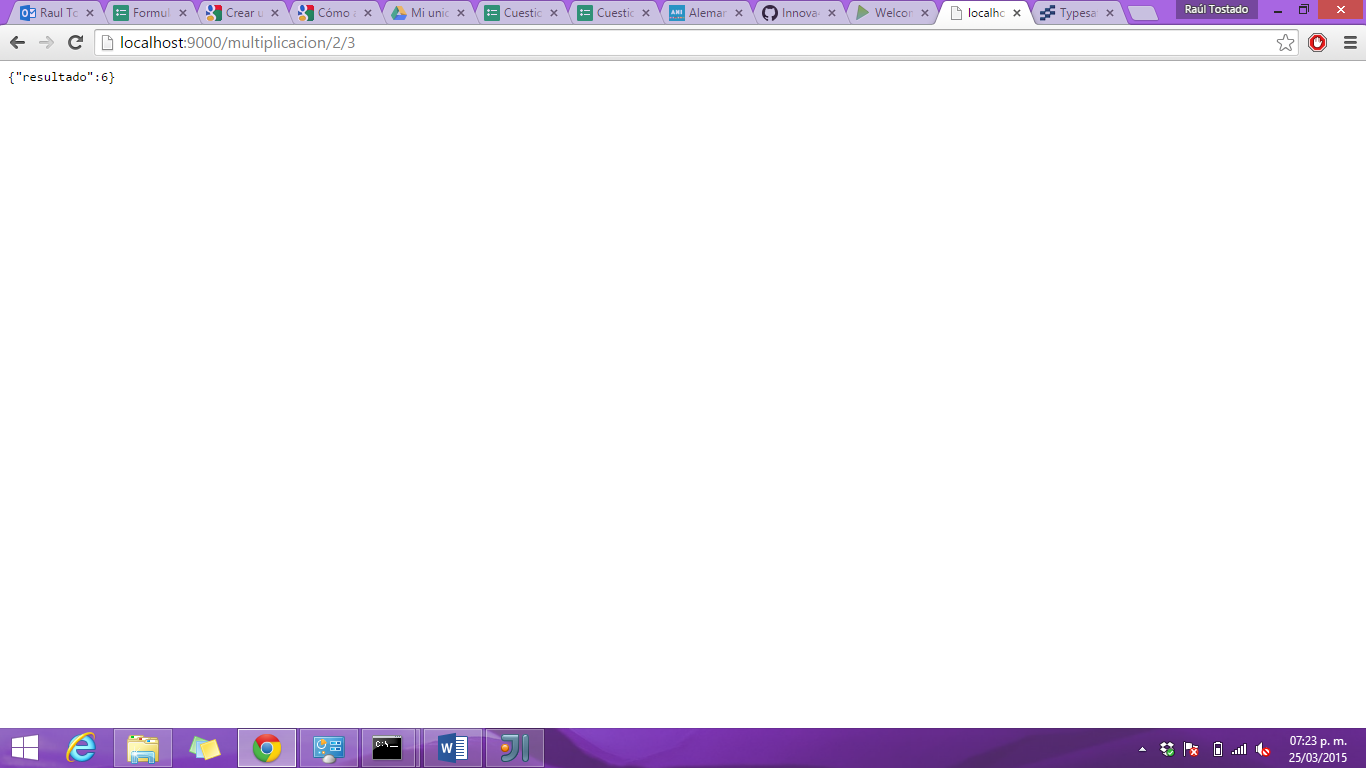
# Mis Servicios Web

GET /multiplicacion/:a/:b controllers.Application.getMultiplicacion(a: Int,b: Int)

# Map static resources from the /public folder to the /assets URL path

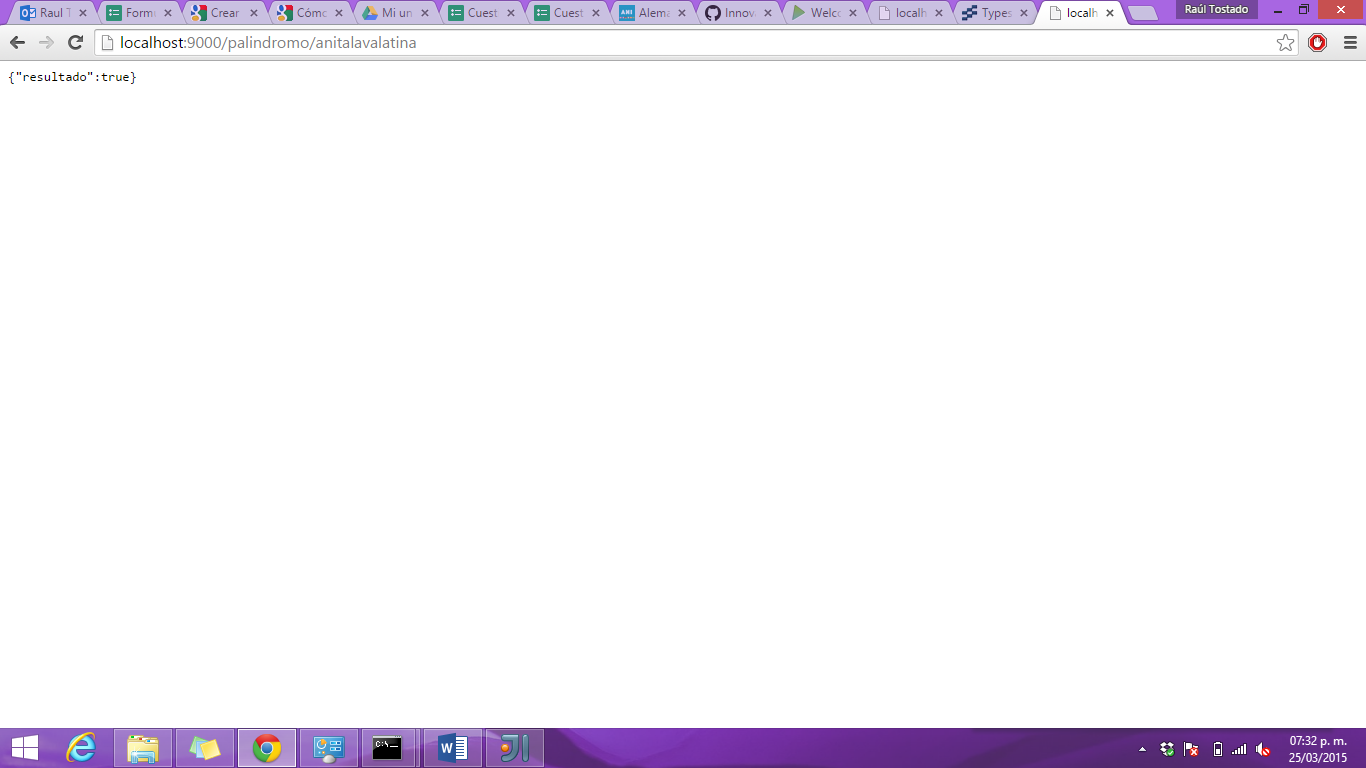
GET /assets/\*file controllers.Assets.at(path="/public", file)

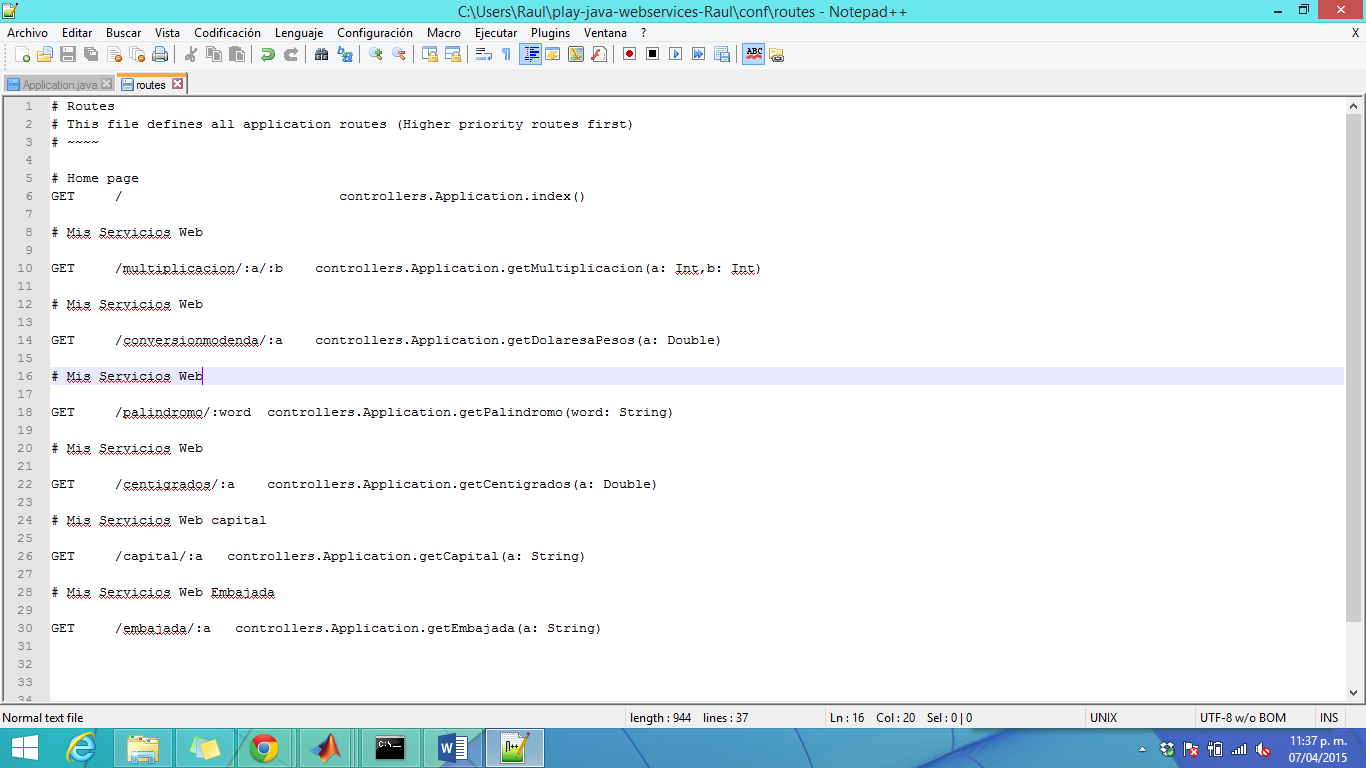
* El servicio Web multiplicación está listo y es accesible a través de**localhost:9000/multiplicacion/2/3**:



* Dado el método **palíndromo** define el **enrutamiento** adecuado para crear un **servicio Web**.
* **Analizar** y **describir** el procedimiento en el **reporte**.

**R=** Teniendo el código para generar el palíndromo, es necesario agregarlo en la aplicación para posteriormente enrutarlo dando la dirección con la cual se va a acceder, en este caso /palíndromo para posteriormente añadir el elemento que recibe el cual será nuestro palíndromo a analizar; para terminar le damos la indicación que llame al método que nos devolverá el palíndromo.

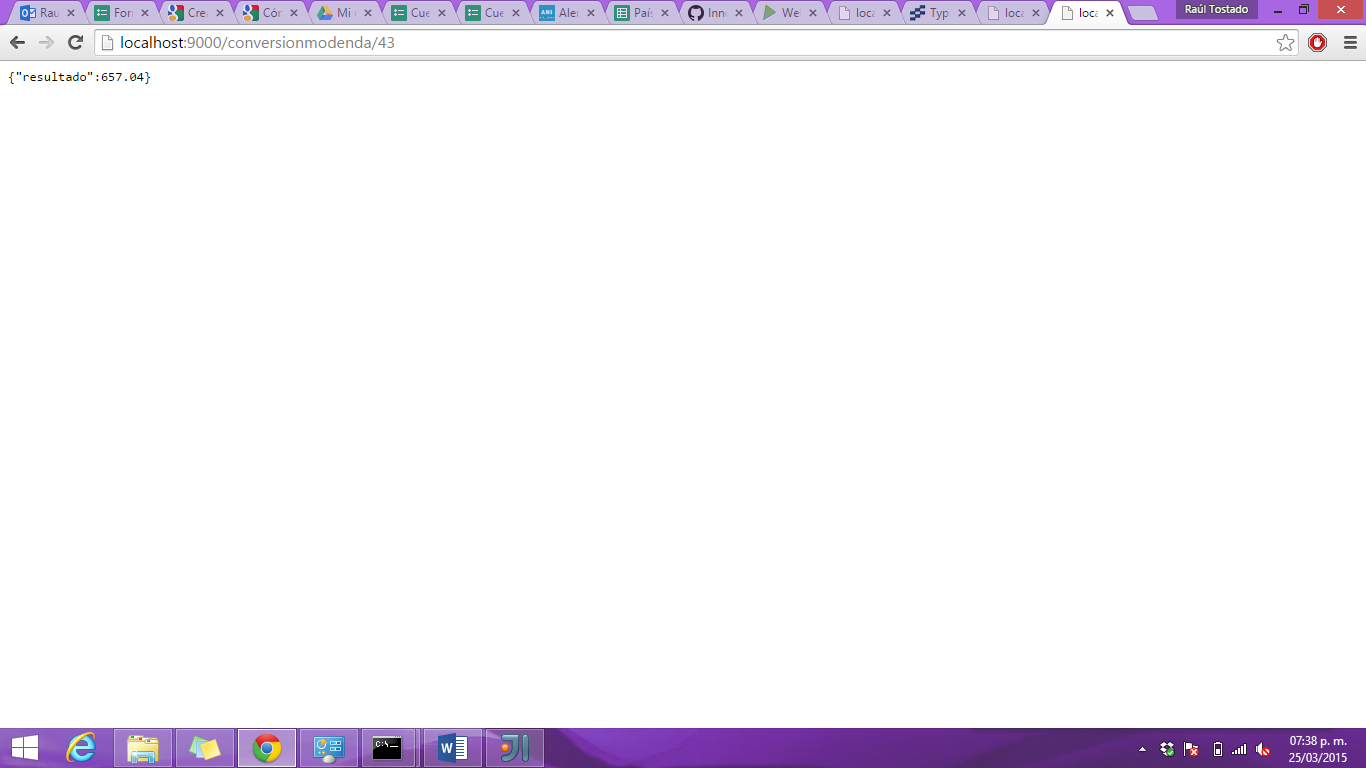
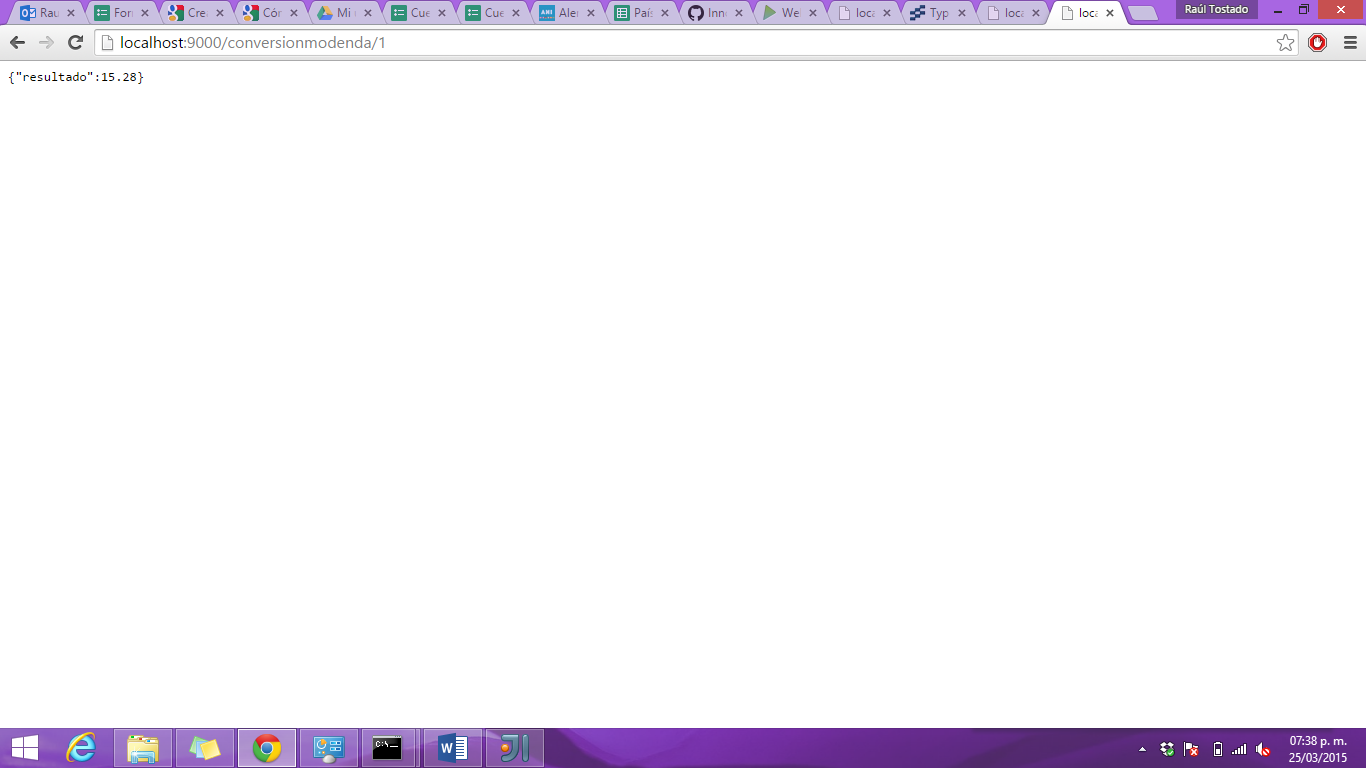




# Actividad 2 - Implementación de Servicios Web

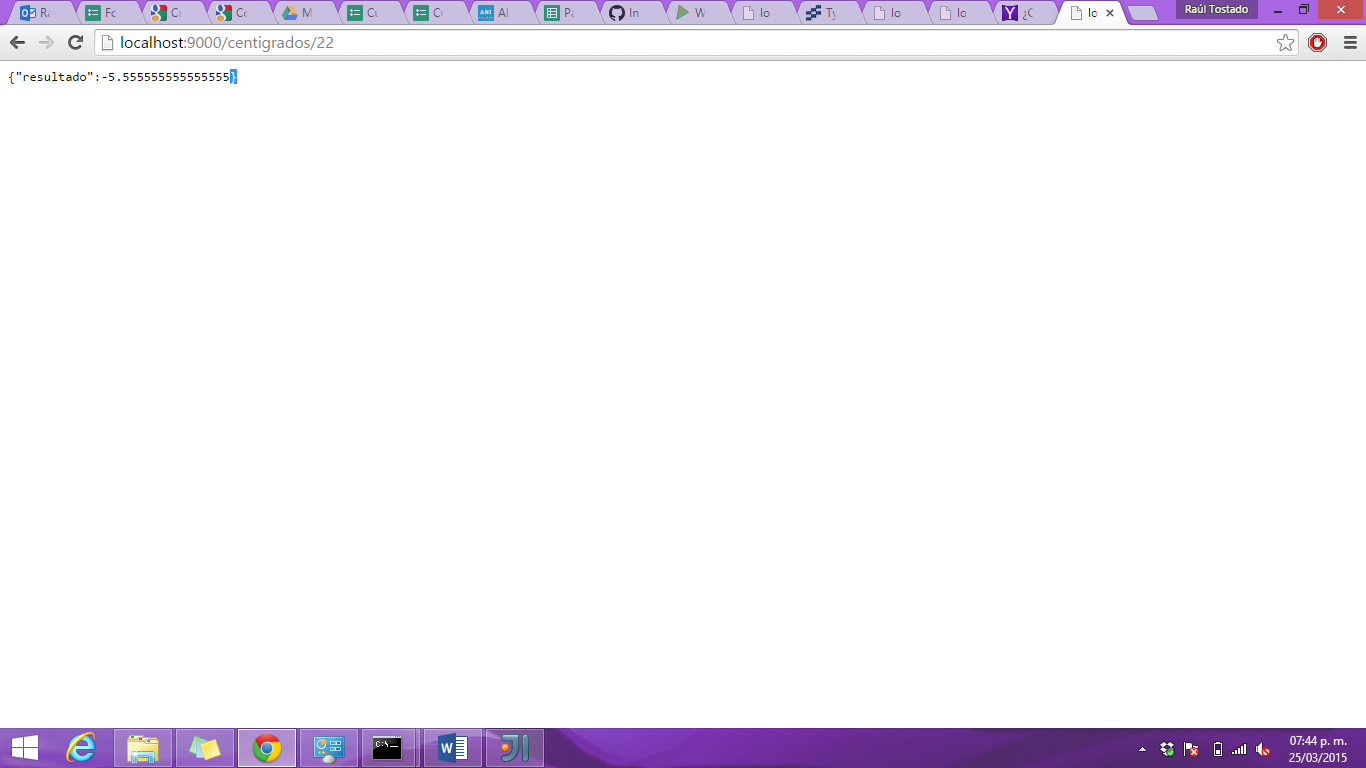
Una vez completado el tutorial es momento de poner en práctica el aprendizaje, como objetivo deberás realizar la implementación de algunos servicios **RESTful**. Cada servicio contiene una entrada y una salida, la salida debe ser en formato **JSON** como se realizó en el tutorial.

* Desarrollar un **Servicio Web** que convierta de **Dólares** a **pesos**.

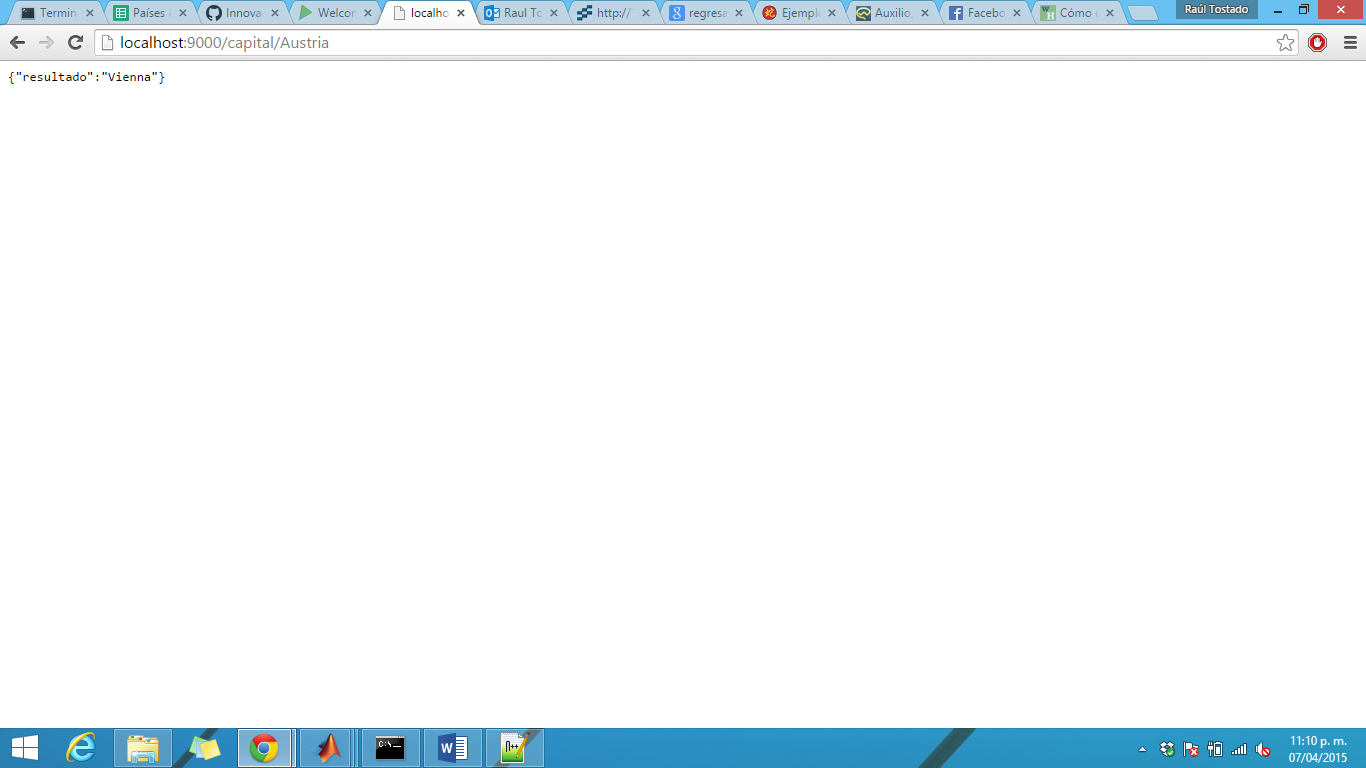
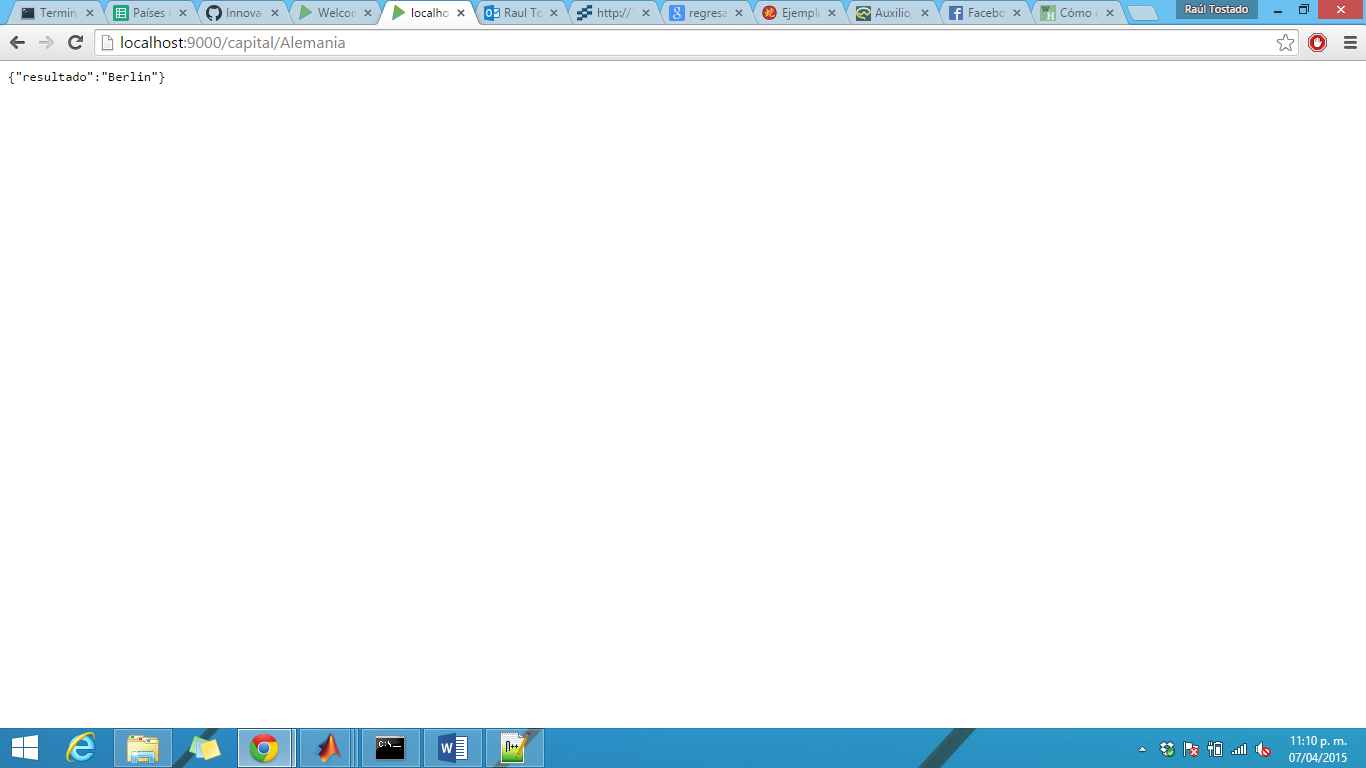
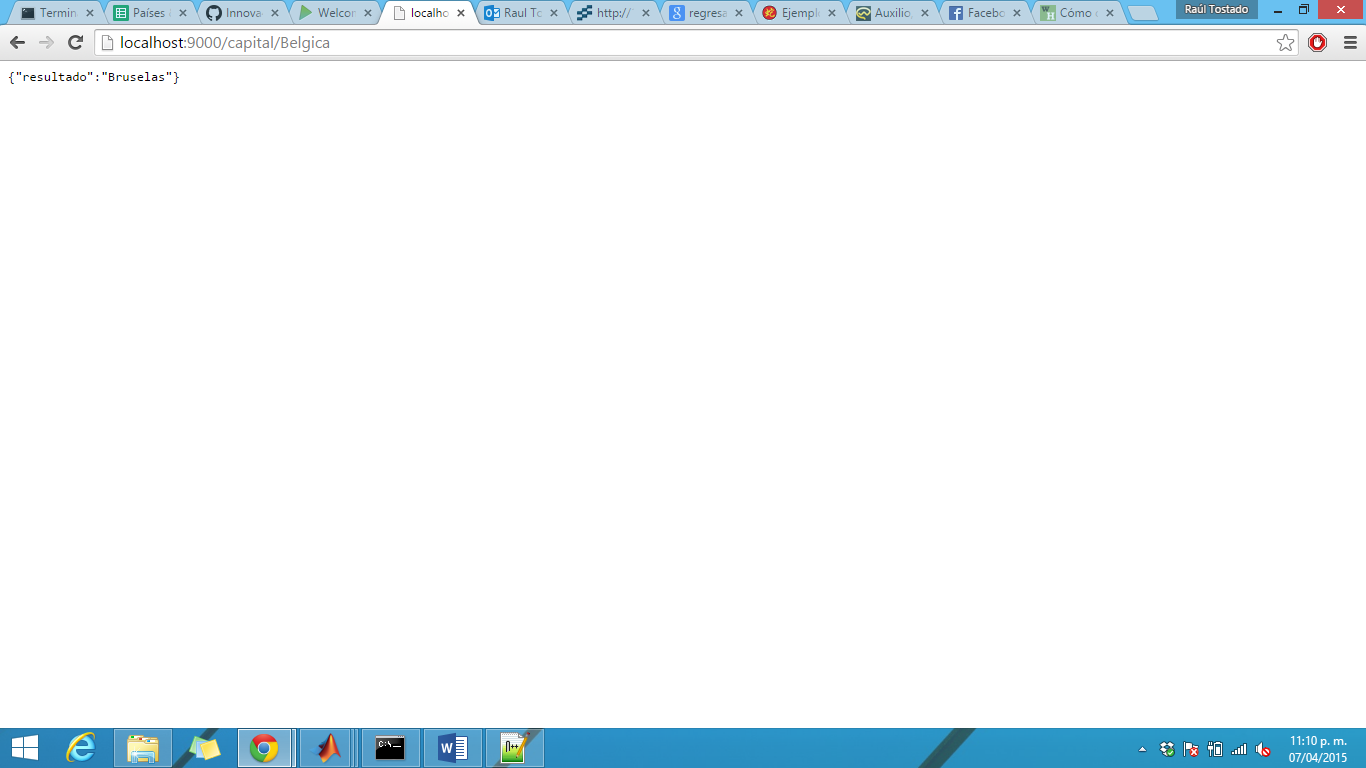


R= Dólares convertidos a pesos mexicanos.

* Desarrollar un **Servicio Web** que convierta de **Farenheit** a **Centígrados**.

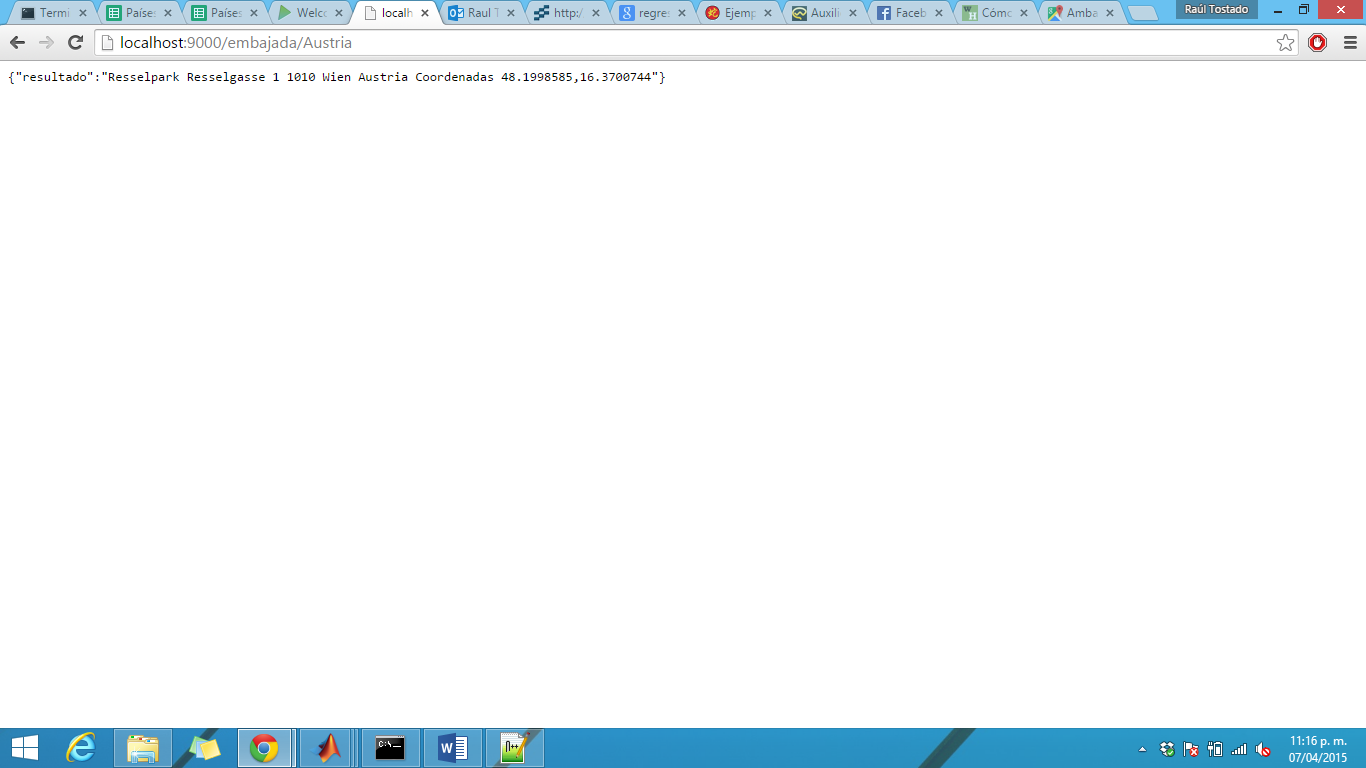


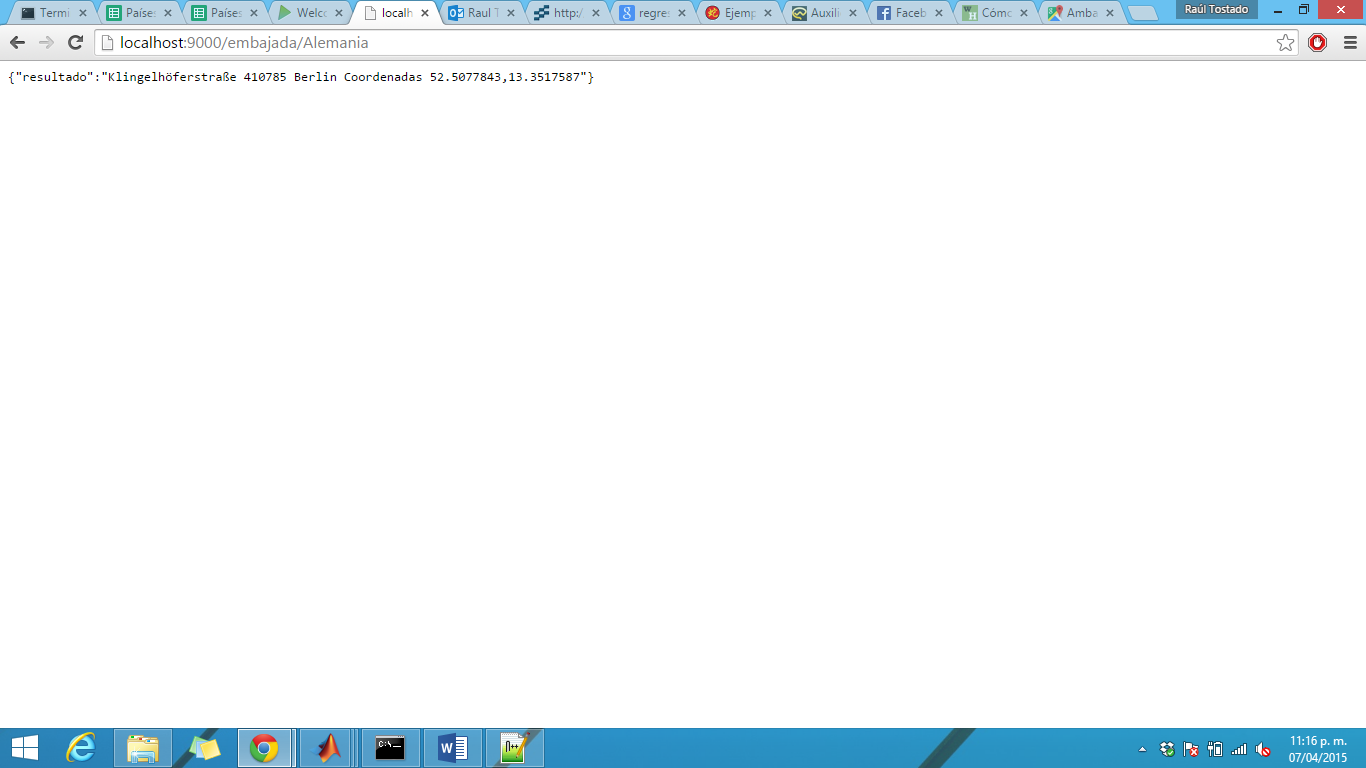
* Desarrollar un **Servicio Web** que dado un **País Europeo**, regrese la **capital**.

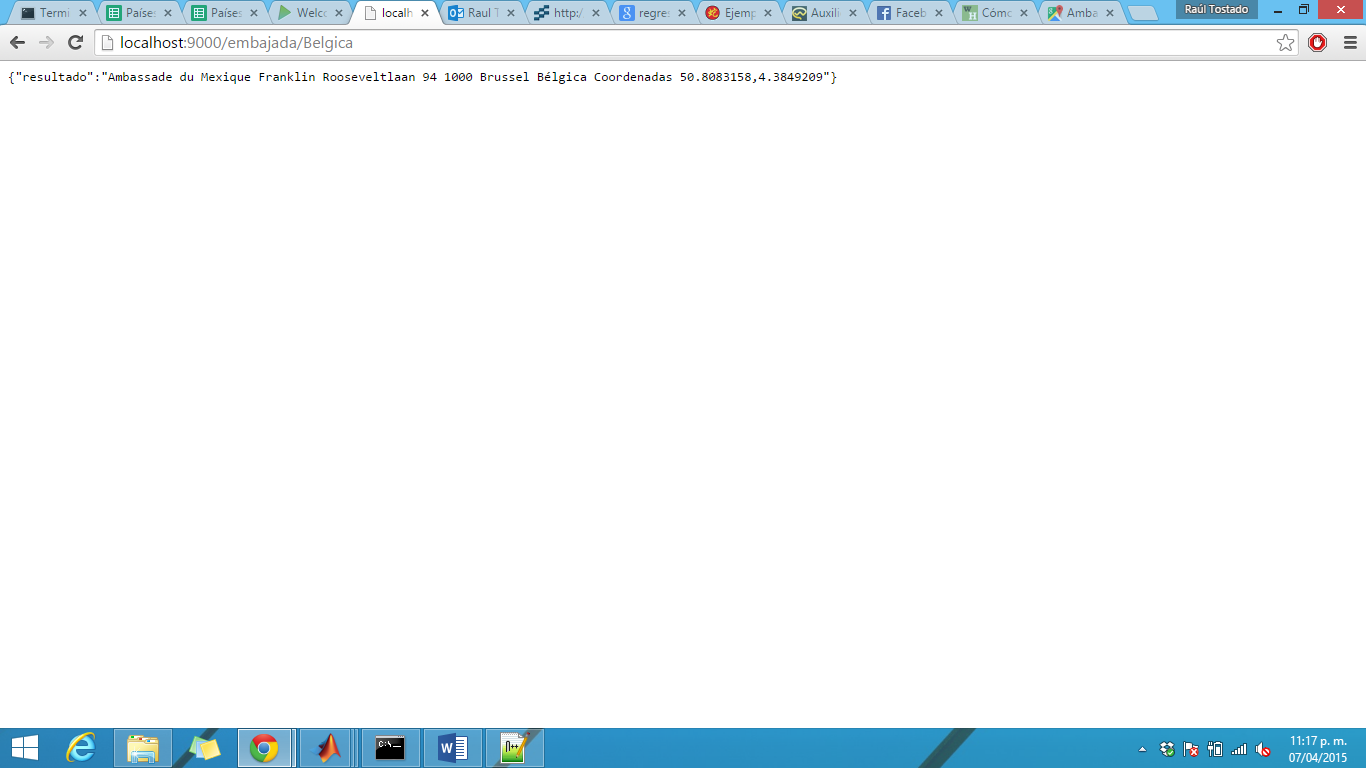


R= En este caso como el posterior, solo se usaron tres ciudades para poder demostrar cómo funcionan los métodos.

* Desarrollar un **Servicio Web** que dado un **País Europeo**, regrese la **ubicación de la embajada de México**.
  + [Lista](https://docs.google.com/spreadsheets/d/15z65g2wvjrj6Q-7S4oRhCNBbJhZuaT2SaVBtb2w822M/edit#gid=0) con ubicaciones de las **embajadas mexicanas** en **Europa**.







R= Además de las coordenadas de la embajada, también se agregó la dirección en la ciudad para encontrar la embajada de una manera más sencilla.

# Actividad 3 - Limpieza

Para comprender mejor las actividades recuerda **limpiar** y **documentar** correctamente el código. Por ejemplo, es importante afinar algunos detalles como escribir los nombres de variables y métodos de manera clara, es decir, nombres que representen e indiquen explícitamente su **funcionalidad**. **Play Framework** utiliza Java, por lo tanto es muy fácil realizar la documentación mediante [Javadocs](http://en.wikipedia.org/wiki/Javadoc).

* Recordando que la documentación debe ser siguiendo las guías de [Javadocs](http://en.wikipedia.org/wiki/Javadoc).

R= Elementos agregados en el código.

# Actividad 4 - Preguntas

En el reporte de la práctica, además de mostrar y explicar el desarrollo de las actividades deberás responder a las siguientes preguntas:

* Explicar el funcionamiento del patrón arquitectónico **MVC**.

R= El MVC consta de tres partes, el model es la representación de la información de la información, es decir, nuestras clases que tendrán los métodos que se van a ejecutar; el view no hace absolutamente nada, es únicamente la vista en la cual podemos ver la información obtenida como resultado de llamar los métodos del model y finalmente el controller, el cual responde a los eventos, en oras palabras sirve como un intermediario entre el model y el view y le dice a cada uno que tarea realizar.

* Explicar los pasos para la creación de un servicio **REST** en **Play**.

R= Primero se tienen que instalar los elementos necesarios en la computadora para poder trabajar, se usa el comando activator new para crearlo en una carpeta específica y posteriormente activator run para poder ejecutarlo. Al estar levantando el servidor, necesitamos un editor para poder agregar nuevos métodos y rutas al servidor, el cuál en este caso Play, nos sirve como un esqueleto para que se levante de manera más eficaz. Cabe señalar que Rest nos devolverá el texto en formato XML (formato más utilizado).

* En la arquitectura de **Play** ¿Cuál es la función de **Router**?

R= Que cualquier cliente pueda conectarse con nuestro servidor, es decir, independientemente de la máquina o sistema operativo que utilicemos, la información va a trabajarse con el formato que le dará el Router para poder ser leída por el Servidor. En el caso de routes damos la forma en la que se va a llamar el método en el navegador.

* Explicar la funcionalidad del **controlador**.

R= Todas las solicitudes pasan por el controlador, el cual las va a mandar al lugar que le corresponde, ya sea al model para solicitar información o al view para mostrarla.

* Explicar la diferencia entre **POST** y **GET**.

R= Get se una en caso de que necesitemos realizar alguna operación, en todos los casos nuestro servicio web recibe un elemento el cuál será con el que trabajará el método.

El protocolo HTTP-GET crea una cadena de consulta de los pares de nombre y valor y, a continuación, anexa la cadena de consulta a la dirección URL de la secuencia de comandos en el servidor que controla la solicitud. Por lo tanto, puede marcar la solicitud.

El protocolo HTTP-POST pasa los pares de nombre y valor en el cuerpo del mensaje de solicitud HTTP. (Microsoft, 2015)

GET lleva los datos de forma "visible" al cliente (navegador web). El medio de envío es la URL. Los datos los puede ver cualquiera. Los datos son visibles por la URL, por ejemplo:www.aprenderaprogramar.com/action.php?nombre=pedro&apellidos1= gomez

POST consiste en datos "ocultos" (porque el cliente no los ve) enviados por un formulario cuyo método de envío es post. Es adecuado para formularios. Los datos no son visibles. La ventaja de usar POST es que estos datos no son visibles al usuario de la web. En el caso de usar get, el propio usuario podría modificar la URL escribiendo diferentes parámetros a los reales en su navegador, dando lugar a que la información tratada no sea la prevista. (González, 2015)

* Describir el procedimiento para crear un **servicio Web**.

R= Mismas acciones que en el punto de Rest, lo necesario es levantarlo primero para posteriormente crear todos los métodos con los cuales se va a trabajar. Finalmente en routes vamos a indicar como se va a llamar cada método.

**Notas:** Explica ampliamente y justifica tus respuestas.

# Actividad 5 - Conclusiones

En esta práctica aprendimos a levantar un servicio web con la ayuda de Play; la práctica fue muy interesante, ya que el ver los componentes del servidor, se puede aprender mucho sobre cómo crear cualquier tipo de tarea, además de que al solicitar un valor, nos devuelve el resultado como una cadena de tipo Json.

# Referencias

González, E. (Abril de 2015). *Aprenderaprogramar.com*. Obtenido de http://www.aprenderaprogramar.com/index.php?option=com\_content&view=article&id=527:get-y-post-html-method-formas-de-envio-de-datos-en-formulario-diferencias-y-ventajas-ejemplos-cu00721b&catid=69:tutorial-basico-programador-web-html-desde-cero&Itemid=192

Microsoft. (Abril de 2015). *Microsoft Soporte*. Obtenido de https://support.microsoft.com/es-es/kb/833383/es