Desarrollo de Servicios Web 1

Tema Nº5:SERVICIOS REST - I

Indicador de logro Nº5:Aplica los servicios RESTful a través del API JAX-RS y Java Interfaces.

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº5:**

SERVICIOS REST - I

**Subtema 5.1:**

REST (REpresentational State Transfer)

**MARCO TEÓRICO**

Define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

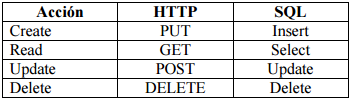
En la mayoría de casos, REST se basa en los siguientes estándares para su implementación:

* Formato: Tipos MIME tales como text/html, application/xml, etc.
* Identificación de recursos: URI ( Uniform Resource Identifier)
* Transporte: HTTP
* Manipulación de recursos:

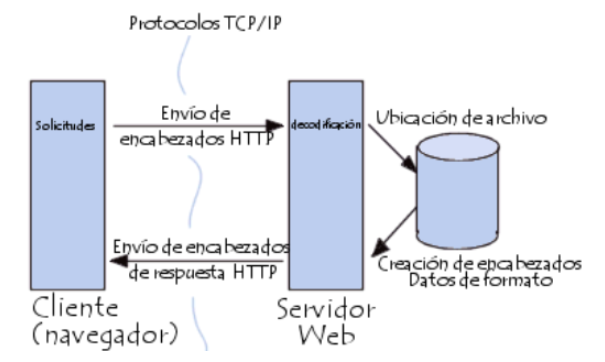
Basado en métodos del protocolo HTTP tales como GET, PUT, POST, DELETE

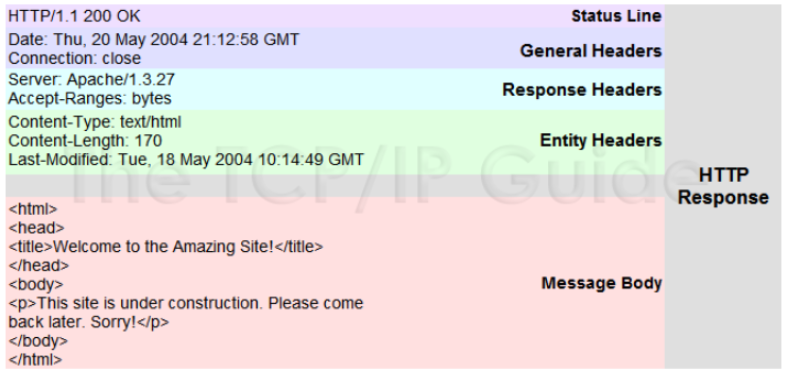
**Principios de la Arquitectura REST**

* **Recurso:** Representados por un formato (jpeg, mpeg, html, xml, etc.) Estos recursos se pueden ver en internet.
* **Identificador Único:** Cada recurso debe ser identificable por un identificador único (URI única).
* **Métodos HTTP estándar:** Acciones o verbos como GET, PUT, POST, DELETE, HEAD, OPTIONS, TRACE, CONNECT.
* **GET:** Para obtener un recurso.
* **PUT:** Para crear un recurso en el servidor.
* **POST:** Para cambiar un recurso o actualizarlo.
* **DELETE:** Para eliminar un recurso.

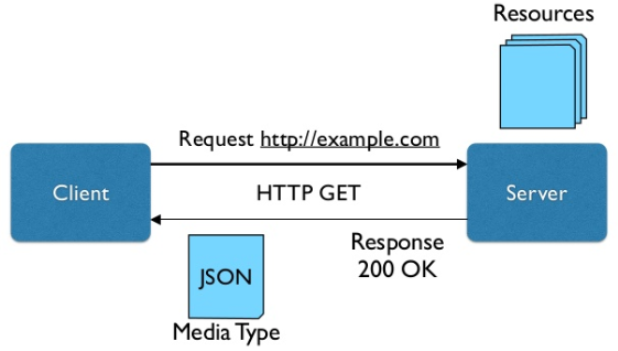


Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.





El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente [HTML](https://www.google.com/url?q=http://es.wikipedia.org/wiki/HTML&sa=D&ust=1503593632213000&usg=AFQjCNEbNliCRhzwPhkOVS5H5mi422b9Yw) o [XML](https://www.google.com/url?q=http://es.wikipedia.org/wiki/XML&sa=D&ust=1503593632214000&usg=AFQjCNHijWG7NT3GLsi_hAkKmwVLWVHDFA). Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.



**Subtema 5.2:**

REST vs SOAP

**MARCO TEÓRICO**

|  |  |
| --- | --- |
| **REST** | **SOAP** |
| • Expone “recursos” que simbolizan información.  •Utiliza métodos HTTP: (GET / PUT / POST / DELETE)  • Se enfatiza en la comunicación punto a punto sobre HTTP.  • Se enfatiza en no mantener estados en la comunicación para promover bajo acoplamiento. | • Expone operaciones que representan lógica de negocio.  • Utiliza únicamente HTTP POST para la transferencia de información.  • Utiliza HTTP GET para la descripción del servicio o WSDL.  • Soporta únicamente XML y datos adjuntos.  • Soporta el manejo de la comunicación con y sin estado |
| • Puede optimizar el consumo de recursos y ancho de banda ya que los mensajes son más ligeros (no necesariamente utiliza SOAP y XML para el intercambio de información).  • El cliente no necesita información de enrutamiento a partir de la URI inicial. | • Ha sido utilizado ampliamente para el desarrollo de web services y cuenta con mayor cantidad de tecnologías asociadas.  • Se puede asociar con servicios empresariales como - Alta disponibilidad con WS-Reliable Messaging (WS-RM) - Transacciones con WS-Atomic Transactions (WS-AT) - Seguridad con WS-Security. |

**Jersey Java Client**

La implementación Jersey de Jax-RS nos brinda una API de clases para generar clientes de un web service REST desde clases Java, a continuación se observan ejemplos básico del uso de esta API, la documentación completa de la API se puede obtener en:

[https://jersey.java.net/documentation/latest/client.html](https://www.google.com/url?q=https://jersey.java.net/documentation/latest/client.html&sa=D&ust=1503593632215000&usg=AFQjCNHlD9D8Auh2fPbiTpCZfH8GgYzh0Q" \t "_blank)

En conjunto con las clases generadas por la API de Jersey se puede utilizar la herramienta XJC como para generar las clases de negocio a partir del XSD que se observa en el WADL.

**1. MATERIALES**

Para la experiencia a realizar se requiere lo siguiente:

1. EQUIPO

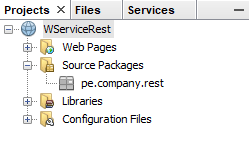
Se requiere tener un equipo de cómputo (PC o laptop).

1. HERRAMIENTAS

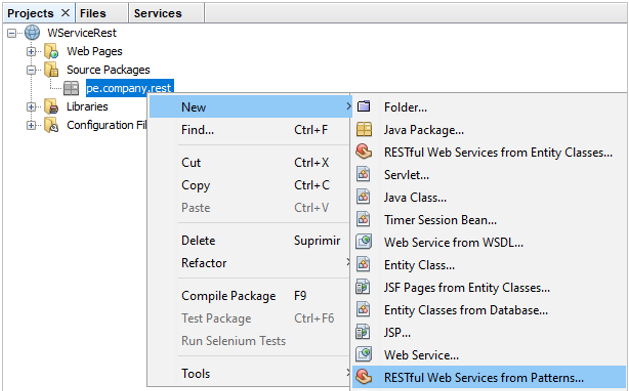
* Java Developer Kit (JDK) version 8.x
* NetBeans 8.2
* Apache Tomcat 9.0.12
* Jersey 2.5 (JAX-RS RI)
* SoapUI 5.5

**2. PROCEDIMIENTO**

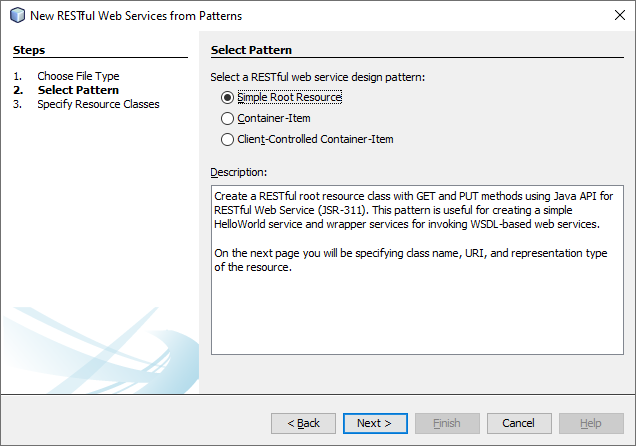
Se realizará un Proyecto Web donde se pueda consumir un mensaje en diferentes tipos de textos. Se tiene el proyecto, con el siguiente paquete:



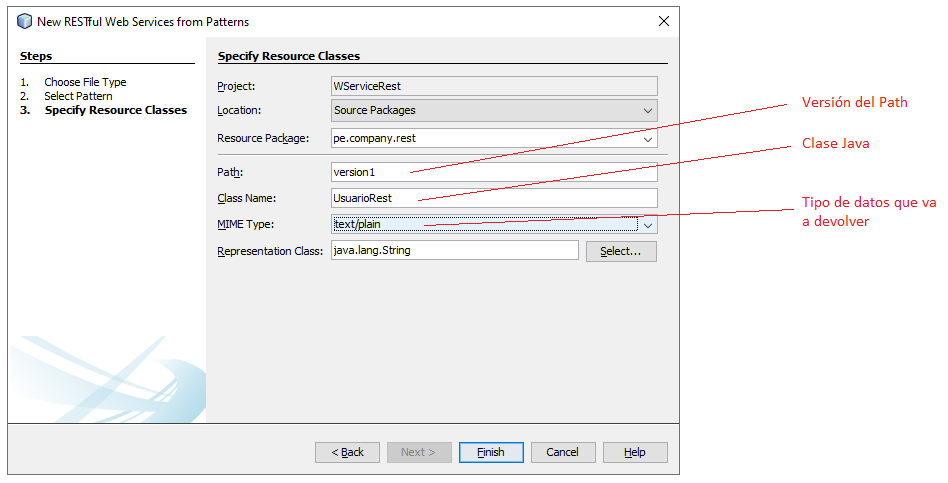
Se pasa a crear el Servicio (Web Service desde patrones):



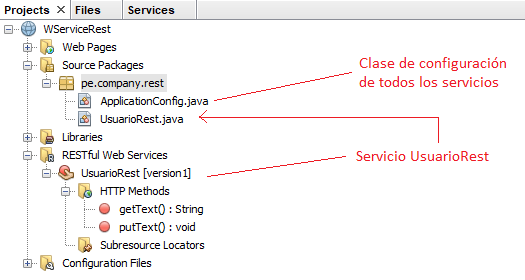
Nos aparece, en este caso se marca Simple Root Resource que trabajará con la presentación JSR-311



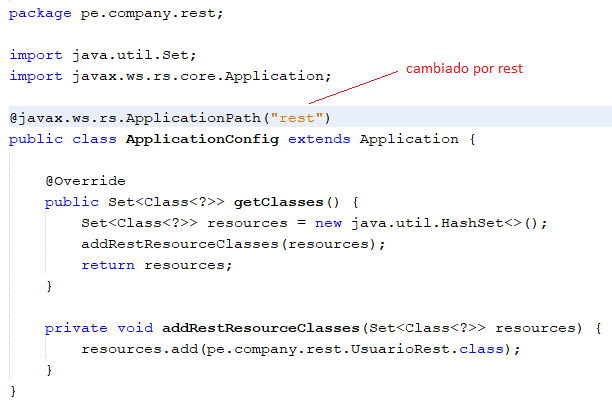
Especificamos lo siguiente:



Se Observa:



Nos dirigimos al ApplicationConfig.java, por comodidad cambiamos en ApplicationPath:



Ahora nos dirigimos al servicio Rest (UsuarioRest.java):

Hacemos:

package pe.company.rest;

import javax.ws.rs.core.\*;

import javax.ws.rs.\*;

@Path("version1")

public class UsuarioRest

{

@Context

private UriInfo context;

public UsuarioRest() {}

@Path("mensaje") //ruta URL -> /rest/version1/mensaje

@GET //indica que la acción es lectura

@Produces(MediaType.TEXT\_PLAIN) //indica que va a producir información en texto plano

public String getText() {

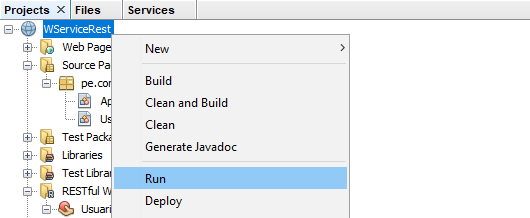
String response="Helo World with Web Service Rest"; //información para el texto plano

return response;

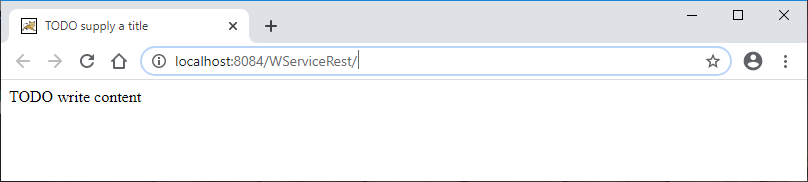
}

}

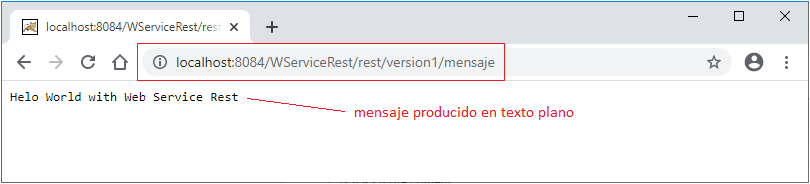
Ejecutamos el proyecto:



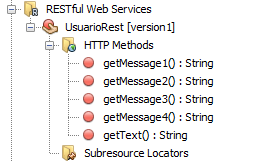
Nos muestra:



Para acceder al servicio colocamos la URL y se tendrá:



Se tiene las operaciones con los diferentes tipos de Texto:



package pe.company.rest;

import javax.ws.rs.core.\*;

import javax.ws.rs.\*;

@Path("version1")

public class UsuarioRest

{

@Context

private UriInfo context;

public UsuarioRest() {}

@Path("mensaje") //ruta URL -> /rest/version1/mensaje

@GET //indica que la acción es lectura

@Produces(MediaType.TEXT\_PLAIN) //indica que va a producir información en texto plano

public String getText() {

String response="Helo World with Web Service Rest"; //información para el texto plano

return response;

}

@Path("message1/{username}") //ruta URL -> /rest/version1/message1/{username}

@GET

@Produces(MediaType.TEXT\_PLAIN) //indica que va a producir información en texto plano

public String getMessage1(@PathParam("username") String username) {

String response="Welcome "+username+"!";

return response;

}

@Path("message2") //ruta URL -> /rest/version1/message2?username={username}

@GET

@Produces(MediaType.TEXT\_HTML) //indica que va a producir información en texto HTML

public String getMessage2(@QueryParam("username") String username) {

String response="<h3>Welcome "+username+"!</h3>";

return response;

}

@Path("message3/{username}") //ruta URL -> /rest/version1/message3/{username}

@GET

@Produces(MediaType.TEXT\_XML) //indica que va a producir información en texto XML

public String getMessage3(@PathParam("username") String username) {

String response="<msg>Welcome "+username+"!</msg>";

return response;

}

@Path("message4") //ruta URL -> /rest/version1/message4?username={username}

@GET

@Produces(MediaType.APPLICATION\_JSON) //indica que va a producir información en texto JSON

public String getMessage4(@QueryParam("username") String username) {

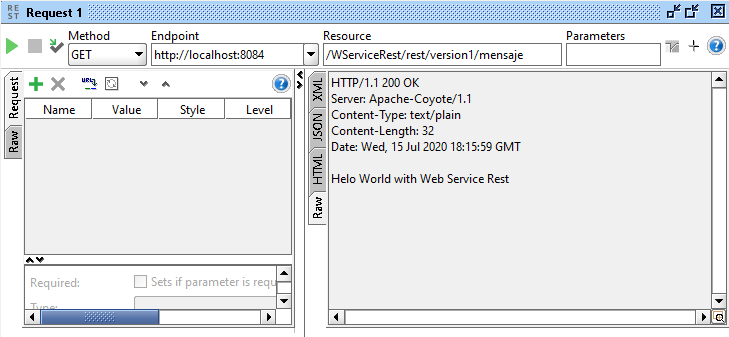
String response="{Welcome "+username+"!}";

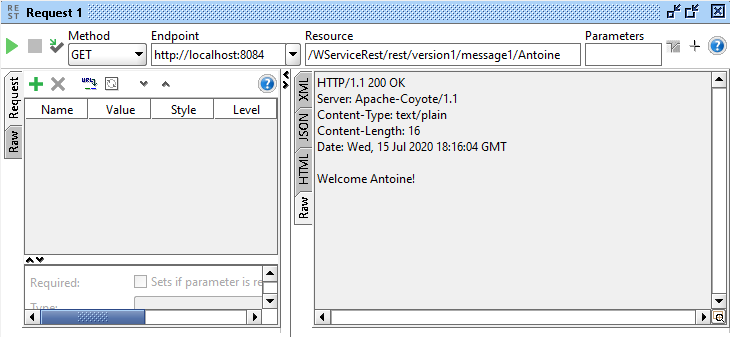
return response;

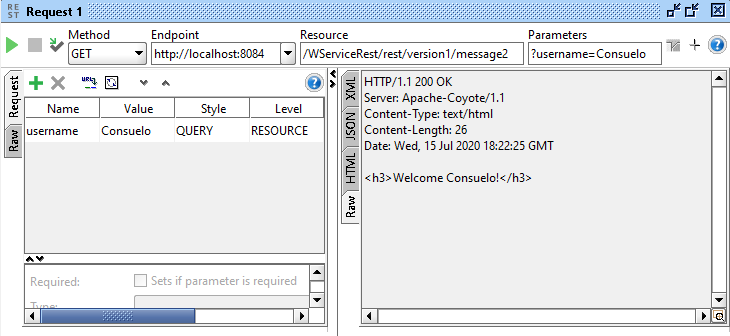
}

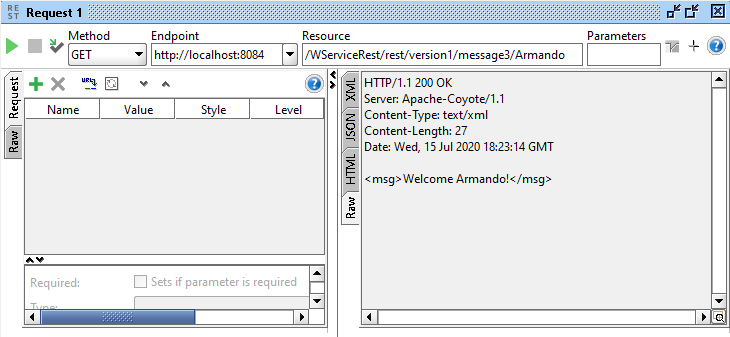
}

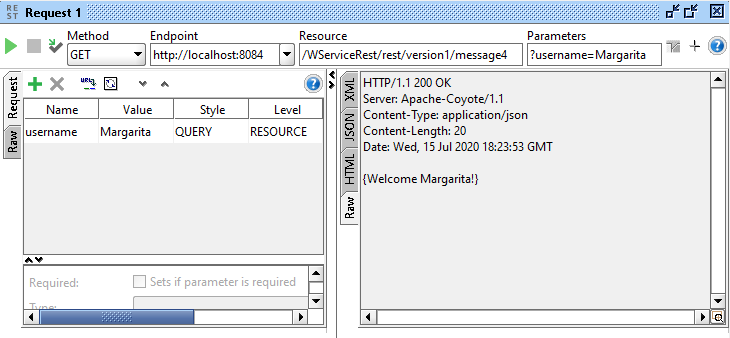
Pruebas unitarias con SoapUI:











**ACTIVIDAD VIRTUAL:**

1. **CUESTIONARIO TÉCNICO**

Revisa y analiza el tema desarrollado en la presente sesión, luego responde las siguientes preguntas propuestas:

* ¿Por qué Rest y no SOAP?
* ¿Qué función tienen los métodos HTTP: HEAD, OPTIONS, TRACE, CONNECT?
* ¿La anotación @PathParam y @QueryParam en qué se diferencian?
* ¿Qué se entiende al poner la anotación @Produces?
* Crear un servicio REST mediante NetBeans, Tomcat y Jersey. Iniciar en la máquina local y probar accesos de clientes.

1. **ENLACES DE REFERENCIA**

* https://dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful
* https://developers.sw.com.mx/knowledge-base/soap-o-rest/
* http://www.jtech.ua.es/j2ee/2010-2011/restringido/servc-web/sesion04-apuntes.html

1. **CONCLUSIONES DE LA EXPERIENCIA**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_