Christos Papastamos csd4569
HY452 Services Technologies
Assignment 2

Task A

The desired XML schema can be seen below

```
<element name="email" type="emailType"/>
    <complexType name="emailType">
        <sequence>
            <element name="head" type="headType"/>
            <element name="body" type="bodyType"/>
        </sequence>
    </complexType>
    <complexType name="headType">
       <element name="email" type="emailType"/>
<complexType name="emailType">
    <sequence>
        <element name="head" type="headType"/>
        <element name="body" type="bodyType"/>
    </sequence>
</complexType>
<complexType name="headType">
    <sequence>
        <element name="from" type="nameAddress"/>
        <element name="to" type="nameAddress" minOccurs="1" maxOccurs="unbounded"/>
        <element name="cc" type="nameAddress" minOccurs="0" maxOccurs="unbounded"/>
        <element name="subject" type="string"/>
    </sequence>
</complexType>
<complexType name="nameAddress">
    <attribute name="name" type="string" use="optional"/>
    <attribute name="address" type="string" use="required"/>
</complexType>
<complexType name="bodyType">
    <all>
        <element name="text" minOccurs="0" maxOccurs="unbounded" type="string"/>
        <element name="attachment" minOccurs="0" maxOccurs="unbounded">
        <complexType>
            <attribute name="encoding" use="default" value="mime">
                <simpleType>
                    <restriction base="string">
                        <enumeration value="mime"/>
                        <enumeration value="binhex"/>
                    </restriction>
                </simpleType>
            </attribute>
            <attribute name="file" type="string" use="required"/>
        </complexType>
        </element>
    </all>
</complexType>
```

To achieve the desired schema I had to change the 4rth complex type from sequence to all.

I also had to change the min and max Occurs of the element “text” to 0 and unbounded to allow an arbitrary amount of text entries.

Task B

For this task we were asked to create a Dynamic Web Project like the one we implemented on the course’s lectures.

i. For the cm_to_inches and inces_to_cm service I created two files which can be seen below.

```java
package hy452.ws.rest;

import javax.ws.rs.GET;

@Path("inch_to_cm") //set your service url path to <base_url>/hello
// the <base_url> is based on your application name, the servlet
// and the URL pattern from the web.xml configuration file
public class cm_to_inch{
    // This method is called if TEXT_PLAIN is requested
    @GET
    @Produces(MediaType.TEXT_PLAIN) //defines which MIME type is delivered by a method annotated with @GET
    public double convertInch(@QueryParam("inch") double inch) {
        return inch * 2.54;
    }

}
```

```java
package hy452.ws.rest;

import javax.ws.rs.GET;

@Path("cm_to_inch") //set your service url path to <base_url>/hello
// the <base_url> is based on your application name, the servlet
// and the URL pattern from the web.xml configuration file
public class inch_to_cm{
    // This method is called if TEXT_PLAIN is requested
    @GET
    @Produces(MediaType.TEXT_PLAIN) //defines which MIME type is delivered by a method annotated with @GET
    public double convertInch(@QueryParam("cm") double cm) {
        return cm / 2.54;
    }
}
```

I created a function in each file converting inches to cm and vice versa, which take as argument a double from the Query part of the request.

ii. For this question I created a new Dynamic Web Project containing a class with a function which gets weather info from the OpenWeatherMap API (as an XML) for a specific city specified in the query part. The code can be seen in the screenshot below:

```java
package hy452.ws.rest;

import java.io.BufferedReader;

@Path("CitySearch") //set your service url path to <base_url>/hello
// the <base_url> is based on your application name, the servlet
// and the URL pattern from the web.xml configuration file
public class CitySearch{
    // This method is called if TEXT_PLAIN is requested
    @GET
    @Produces(MediaType.TEXT_PLAIN) //defines which MIME type is delivered by a method annotated with @GET
    public String convertInch(@QueryParam("city") String city) throws IOException {

        if(city==null){return null;}

        URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q="+city+"&mode=xml&appid=2144527af17f383e91ad2a7efcd8c87b");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();

        return content.toString();
    }
}
```

iii. For the bonus part I just had to create a string using the values found in the json API output. The code used can be seen below:

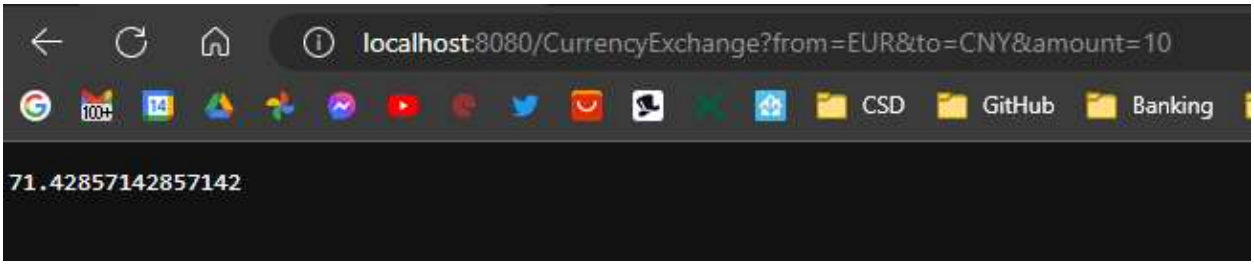
```java
package hy452.ws.rest;

import java.io.BufferedReader;


@Path("PrettyCitySearch") //set your service url path to <base_url>/hello
// the <base_url> is based on your application name, the servlet
// and the URL pattern from the web.xml configuration file
public class PrettyCitySearch{
    // This method is called if TEXT_PLAIN is requested
    @GET
    @Produces(MediaType.TEXT_PLAIN) //defines which MIME type is delivered by a method annotated with @GET
    public String convertInch(@QueryParam("city") String city) throws IOException {

        if(city==null){return null;}

        URL url = new URL("https://api.openweathermap.org/data/2.5/weather?q="+city+"&mode=json&appid=2144527af17f383e91ad2a7efcd8c87b");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");

        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer content = new StringBuffer();
        while ((inputLine = in.readLine()) != null) {
            content.append(inputLine);
        }
        in.close();

        JSONObject ret_obj = new JSONObject(content.toString());

        String printstr = "In " + ret_obj.get("name") + " it has "+ ret_obj.getJSONObject("main").get("humidity") + "% humidity";

        return printstr;
    }
}
```

Task C

i. For the first question I created a REST-style web service using Spring Boot following the method we discussed in the lectures. The Application code can be seen below

```java
package com.example.TaskC_hy452;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
@RestController
@OpenAPIDefinition(info = @Info(title="Currency Conversion API", description="Given an amount and a currency, get back the amount in any other supported currency" ))
public class TaskCHy452Application {

    public static void main(String[] args) {
        SpringApplication.run(TaskCHy452Application.class, args);
    }

    @Operation(summary = "Return the given amount in the desired currency")
    @GetMapping("/CurrencyExchange")
    public double hello(@RequestParam(value= "from", defaultValue= "EUR")String from, @RequestParam(value= "to", defaultValue="USD") String to, @RequestParam(value= "amount", defaultValue="0") double amount) {
        if(amount==0) {return 0;}
        double amountinEUR  = 0;
        double outputamount = 0;

        switch (from) {
        case "EUR":
            amountinEUR=amount;
            break;
        case "USD":
            amountinEUR=amount * 0.97;
            break;
        case "GBP":
            amountinEUR=amount * 1.15;
            break;
        case "HRK":
            amountinEUR=amount * 0.13;
            break;
        case "CNY":
            amountinEUR=amount * 0.14;
            break;
        default:
            return -1;
        }

        switch (to) {
        case "EUR":
            outputamount=amountinEUR;
            break;
        case "USD":
            outputamount=amountinEUR / 0.97;
            break;
        case "GBP":
            outputamount=amountinEUR / 1.15;
            break;
        case "HRK":
            outputamount=amountinEUR / 0.13;
            break;
        case "CNY":
            outputamount=amountinEUR / 0.14;
            break;
        default:
            return -1;
        }
```

I deployed the maven application and the result was the following



ii. For the API documentation I installed Swagger and added some documentation lines in the code (line 15,22 in the screenshot above). The final Swagger UI can be seen in the screenshot below