REST and JSON

General part

- Explain the term REST and the architectural Requirements that relates to REST
- Elaborate on how JSON or XML supports communication between subsystems, even when the subsystems are implemented on different platforms.

Practical part

1) Often, when we create web applications, we have the need for large amounts of test data.

Implement a class, which can provide random test data as sketched below[1]:

```
String data = dataGenerator.getData(100,"fname,lname,street,city ");
```

The first argument indicates the amount of test persons to create, the second which properties to include.

So the example above should return a JSON array with 100 test data on the form:

```
[{"fname": "Bo", "lname":"Hansen", "street": "Lyngbyvej 26", "city":
"Lyngby"},..]
```

Hint: See next page for a hint of how to create the JSON

And used like this: String `data = dataGenerator.getData(25,"fname, lname");`

it should return 25 test data as sketched below (only firstName and lastName are included):

```
[{"fname": "Bo", "lname":"Hansen"},..]
```

2) Create a JAX-RS application, which should implement a REST service, that when called like this:

```
GET: http:/……/api/addresses/100/fname,lname,city
```
[2]

Should return a JSON array as sketched below:

```
[{"fname": "Bo", "lname" : "Hansen","city": "Lyngby"}, {..}, …]
```

3)

Create a simple web page which should, using whatever technology you prefer, render a table with test data fetched via the REST method implemented in step 2.

4) If you have time. Add a response header to the REST service that will solve the same origin policy problem, so client pages hosted on other servers can access the service.

---

[1] *Hint: Include some hard coded arrays in your module like ["Ib", "Bo", "Lars", "Henrik"] and pick a random value for each name you generate*

[2] *Hint: Use the @PathParam annotation and threat the last two parts (/100/fname,lname,city) as arguments*

Note: this part is NOT a part of the exercise, it's meant as FYI.

The following Code will (try) generate a JSON array with **one** person, with all data. So the task boils down to something like. Parse the inputs string to see how many fields to include. Loop around the required number of times, and generate a new person each time including only the required fields.

```
JsonArray names = new JsonArray();
  JsonObject person = new JsonObject();
  person.addProperty("fName", "Lars");
  person.addProperty("lName", "Mortensen");
  person.addProperty("street", "Lyngbyvej 23");
  person.addProperty("city", "Lyngby");
names.add(person);

Gson gson = new GsonBuilder().setPrettyPrinting().create();
String jsonStr = gson.toJson(names); //The JSON string is ready
System.out.println(jsonStr);
```

General advices:

Spend you 80 minutes well. Don't (initially) waste your time in trying to come up with "sensible" data.

As a start ["aa","bb","cc","dd"] is just as good as ["Lyngbyvej 25", "Vesterbrogade 23","Sofie vej 23"]

Spend you 80 minutes well. It is better to have a solution that can only generate *firstname* and *lastname*, but also includes the REST part, and perhaps a bit of the client side part, compared to a solution that only covers step-1. The final fields can be added if you have time.