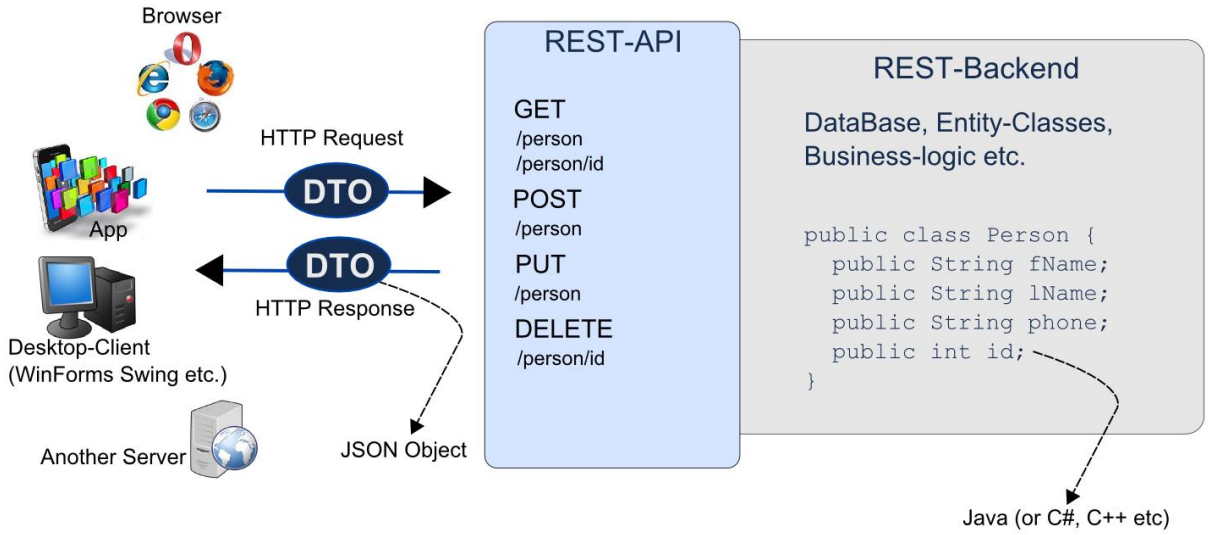
REST with JAX RS and Client with Javascript and AJAX

In this exercise we will go through most of the steps necessary to create a REST driven application as sketched below, using a very simple one-class model, to simplify matters.



In the backend we will implement a Java Person class and a façade implementing the following interface:

```
public interface IPersonFacade {
  Public void addEntityManagerFactory(EntityManagerFactory emf);
  public Person addPerson(Person p);
  public Person deletePerson(int id);
  public Person getPerson(int id);
  public List<Person> getPersons();
  public Person editPerson(Person p);
}
```

By now you should know that an important REST constraint is to have a layered system, with Resources Decoupled from their Representation. For this exercise we will expose data as JSON using the URIs given in the figure above.

For the REST-URIs that either return or consumes an existing Person, the following JSON must be used:

```
{"fName":"Lars","lName":"Mortensen","phone":"12345678","id":0}
```

For the REST URI that creates a Person, use the JSON above, without the id property:

For the GET method that returns all Persons, the JSON must have this format:

```
[{"fName":"Lars","lName":"Mortensen","phone":"12345678","id":0},
{"fName":"Peter","lName":"Olsen","phone":"12345678","id":1}]
```

To help with the conversion between your Java backend and the JSON-based frontend you should design a utility class as sketched below:

```
public class JSONConverter {
  public static Person getPersonFromJson(String js){..}
  public static String getJSONFromPerson(Person p) {..}
  public static String getJSONFromPerson(List<Person> persons) {..}
}
```

Tasks

Server side:

1) Create a new NetBeans Maven Web Project
2) Create an Entity class (with a corresponding database) to implement the Person from the figure above
3) Create a script to setup some sample data and "call" the script from your persistence.xml file (HINT)
4) Implement a Façade class from IPersonFacade and use JUnit to test the Façade. Se hint-1 for help.
5) Implement and test the JSONConverter class introduced above
6) Implement the GET methods from the REST-API and test via a browser
7) Implement the POST method and test using Postman
8) Implement the PUT method and test using Postman
9) Implement the DELETE method and test using Postman

Client side:

1. Implement a read-only page to show all Persons in a table. The table must be built in the browser using plain JavaScript and data fetched via a REST call.
2. Add a refresh button that should refresh the page designed in the previous step. Use Postman to add a new Person to verify that we actually get an updated list, without having to create a new page on the server.
3. Add an option to create new Persons (inspired by the figure below) on the same page as the one with the table (see hint-2). Use the REST API to create the new person on the server.
4. Add an option to delete a Person (row) as sketched on this figure (*see Hint-5*)

ID	First Name	Last name	Phone	
1	Peter	Olsen	1234	delete / edit
2	Hanne	Olsen	1234	delete / edit
3	Kurt	Wonnegut	7985476	delete / edit
4	Heidi	Petersen	1234	delete / edit

Reload Data | Add New Person

5. Add an option to edit a Person (row) as sketched on the figure.

Hints

Hint-1 Testing a JPA Design (mocking the database)

Don't do this until we have had the next test-lesson, related to mocking database and how to test a REST-API

Use this link for info about how to mock away the database:
https://docs.google.com/document/d/1txCv5GpCqsIGNn-PSq5_Qgd5_GQaixNNSQy_C6AuY8Q/edit?usp=sharing

Hint-2: *For this task, you could create a Form next to the table (visible only when you press a "add Person" button or you could use Bootstrap to bring up a modal with the Form*
(http://www.w3schools.com/bootstrap/bootstrap_modal.asp)

Hint-3:

In netbeans add a new folder “scripts” to "src/main/resources" under “other sources” and add a file data.sql

to the folder.
Now add this line to the persistence unit:
<property name="javax.persistence.sql-load-script-source" value="scripts/data.sql"/>
And in data.sql write sql statements to insert sample data into the tables