1 HelloWorldGAE [10 minutes]

1.1 Eclipse

1. Run /packages/mobserv/eclipse/eclipse (and **not the default Eclipse**)
2. Set the android sdk location at Windows-> preferences-> Android -> SDK Location to "/packages/mobserv/android-studio/sdk"
3. Set the java build path at Windows -> Preferences -> Java -> Build path to "Project"
4. Restart eclipse

1.2 Creating a project

5. Click on the Google button on the toolbar and select New Web Application Project. The equivalent command in Eclipse is File → New → Web Application Project;
6. Put "Helloworldgae" in ProjectName, and "eurecom.fr.helloworldgae" as package name;
7. Uncheck "Use Google Web Toolkit." and verify that "Use Google App Engine" is checked, and the "Generate project sample code", too.
8. Click Finish.

1.3 Run the project

1. Click on Run → Debug As→ Web Application.
2. Check on the Console output the line "INFO: Server default is running at http://localhost:8888/":
3. Open http://localhost:8888/helloworldgae in a browser to see the message "Hello, world";
4. When finished, click on the red button in the Eclipse console to stop the server.

2 Hellomoongae [40 minutes]

Of course one can just change the message, but actually we're going to show how the users requests are mapped to a defined URL (dispatching).

2.1 Create a new servlet

1. Right click on the package eurecom.gae.helloworldgae located in the src/ folder, New → Class;
2. Add it to the existing package name

3. Write HellomoongaeServlet in the Name field and javax.servlet.http.HttpServlet in the Superclass field;

4. Add the following code in the class

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
resp.setContentType("text/plain");
resp.getWriter().println("Hello, moon");
}
```

Note that there are two commonly used methods for a request-response between a client and server:

- GET - Requests data from a specified resource
- POST - Submits data to be processed to a specified resource

2.2 Make the servlet available to the users

When the web server receives a request, it determines which servlet class to call using a configuration file known as the "web application deployment descriptor." This file is named web.xml, and resides in the war/WEB-INF/ directory. WEB-INF/ and web.xml are part of the servlet specification.

1. Open the web.xml file;

2. Just after the HelloGAEServlet </servlet> tag, add the following code:

```
<!-- HelloMoon Servlet -->
<servlet>
<servlet-name>Hellomoongae</servlet-name>
<servlet-class>eurecom.fr.helloworldgae.HellomoongaeServlet</servlet-class>
</servlet>
```

3. And just after the </servlet-mapping> tag already present, add the following code

```
<!-- HelloMoon pattern/servlet -->
<servlet-mapping>
    <servlet-name>Hellomoongae</servlet-name>
    <url-pattern>/hellomoongae</url-pattern>
</servlet-mapping>
```

2.3 Run the project

Run the project as in the previous section, but this time request in the browser http://localhost:8888/hellomoongae . You should see a “Hello, moon” message.

2.4 Modify the doGet

We'll see simply how to handle a parameter given by the user and change the output accordingly. Open again HelloMoonServlet and change the method doGet like this:

```
resp.setContentType("text/plain");
String msg = "Hello, moon";
// Check for an argument
if (req.getParameter("name") != null) {
```

```
    msg += " from " + req.getParameter("name");
}
resp.getWriter().println(msg);
```

2.5 Run the project

Run the project and try to make requests changing the name parameter in the request.

For example: http://localhost:8888/hellomoon?name=navid

2.6 Using JSP (Check the code, but skip this part)

We can achieve the same result using Java Server Pages. The JSP is a technology for inserting dynamic content into a HTML or XML page using a Java servlet container. In the example we give as output a complete HTML file.

In the directory war/ create a file hellomoon.jsp with the following content:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Hello Moon</title>
</head>
<body>
<h1>Hello!</h1>
<% String msg = "hello, moon";
// Check if we have an argument
if (request.getParameter("name") != null) { %>
<p><strong><% out.print(request.getParameter("name")); %></strong> says:
<% } %>
Hello, moon</p>
</body>
</html>
```

2.7 Run the project

Restart your server as usual and request from the browser something like:
http://localhost:8888/hellomoon.jsp?name=navid

2.8 Publish the code

2.8.1 Register the application

To register and to manage your Google App Engine application, use the Administration Console: https://appengine.google.com/:

1. Sign in to App Engine using your Google account;
2. Click the "Create an Application" button. Follow the instructions to register an application ID, a name unique to this application. In the rest of the tutorial we will be using “hellomoongae, you have to choose a different name that is unique, for instance hellojupitergae, hellomarsgae, etc.
3. Edit the appengine-web.xml file, then change the value of the <application> element to be hellomoongae.

2.8.2 Upload from Eclipse

To upload your application from Eclipse:

1. click on the Google button in the Eclipse toolbar, then select "Deploy to App Engine.;
2. provide the Application ID hellomoongae, your Google account username, and your password;
3. click the Deploy button.

Please note that you can use directly the SDK command appcfg.sh (Linux, Mac) or appcfg.cmd (Windows) to deploy the application. For the details please refer to the official Google documentation: https://developers.google.com/appengine/docs/java/gettingstarted/uploading .

Now you can use your hellomoongae.appspot.com.

3 AddressBook with Google App Engine [1h30]

In this lab we'll see how to use HTTP methods GET and POST, and how to use the Datastore to save our data in the database.

At the end we will have an AddressBook to create, modify, see and search contacts.

3.1.1 AddressBook part 1: the Database structure

In this part we'll code the classes that represent the Database structure.

Please note that data is written to the Datastore in objects known as **entities**. Each entity has a **key** that uniquely identifies it.

We will make use of the existing project “helloworldgae” to create an address book. You could also create a new project for AddressBook.

3.1.2 Saving a contact

We will create a servlet, called SaveContactServlet (see section 3) that will save contacts in a DB.

As you will see from the code below, we use the name as the key. Obviously in a real world example this is not a good choice because we can have persons with the same name making the key not unique.

3.1.3 POST requests

Let's create a new servlet called “SaveContactServlet”. The class has a doPost method that will serve POST requests as follows.

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    resp.setContentType("text/plain");
    resp.getWriter().println("Save contact servlet. GET method doesn't do anything.");
}
/**
     * Save a contact in the DB. The contact can be new or already existent.
     */
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    // Retrieve informations from the request
    String contactName = req.getParameter("name");
    String contactPhone = req.getParameter("phone");
    String contactEmail = req.getParameter("email");
    String contactPict = req.getParameter("pict");

    // Take a reference of the datastore
    DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();

    // Generate or retrieve the key associated with an existent contact
    // Create or modify the entity associated with the contact
    Entity contact;
    contact = new Entity("Contact", contactName);
    contact.setProperty("name", contactName);
```

```
    contact.setProperty("phone", contactPhone);
    contact.setProperty("email", contactEmail);
    contact.setProperty("pict", contactPict);

    // Save in the Datastore
    datastore.put(contact);

    resp.getWriter().println("Contact " + contactName + " saved with key " +
KeyFactory.keyToString(contact.getKey()) + "!");
}
```

3.1.4 Publishing and run

Modify the web.xml like we saw in the lab1, assigning the url-pattern "/save" and debug as Web Application.

```
<servlet>
<servlet-name>SaveContact</servlet-name>
    <servlet-class>eurecom.fr.helloworldgae.SaveContactServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>SaveContact</servlet-name>
    <url-pattern>/save</url-pattern>
</servlet-mapping>
```

3.1.5 Testing and verification of the POST

To test the our POST actually works, we will use a REST client.

If you use the Chrome Web Browser you can install "Advanced REST client" from the Web Store.

Open it and:

1. The URL is http://localhost:8888/save;
2. Click on "POST";
3. Click on "application/x-www-form-urlencoded' in the last SELECT field;
4. Remove all the headers;
5. Click on "Form" and add the following values:
    - name: value you want;
    - email: value you want;
    - phone: value you want;
    - pict: an URL (find on google images, 128x128px)

When you click on Send you will receive the output, with the Status 200 OK and the message at the end.

3.1.6 Using Google tools to verify the Datastore

Google gives a very advanced console even when you work on the local host.

Access http://localhost:8888/_ah/admin/datastore .

Select “Contact” and then press “List entries”.

You will see your entries.

If you are working on a registered app, please login to https://appengine.google.com and select your app for a complete dashboard.

3.2 AddressBook part 2: visualizing contacts

In this part we'll see how to present a list of contacts and visualize the details for 1 contact.

3.2.1 List of contacts

As we saw in the first part, to present data to the user we can use even JSP. This time we will do it with a servlet.

So, create a new servlet called “ContactsListServlet”, we'll serve GET requests this time, so create the doGet method with the following content:

```
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
            // Take the list of contacts ordered by name
Query query = new Query("Contact").addSort("name", Query.SortDirection.ASCENDING);
 List<Entity> contacts = datastore.prepare(query).asList(FetchOptions.Builder.withDefaults());

 // Let's output the basic HTML headers
 PrintWriter out = resp.getWriter();
 resp.setContentType("text/html");
 out.println("<html><head><title>Contacts list</title></head><body>");
 if (contacts.isEmpty()) {
     out.println("<h1>Your list is empty!</h1>");
 } else {
     // Let's build the table headers
     out.println("<table style=\"border: 1px solid black; width: 100%; text-align: center;\">"
                 + "<tr><th>Name</th><th>Phone Number</th><th>Details</th></tr>");
    for (Entity contact: contacts) {
     out.println("<tr><td>" + contact.getProperty("name") + "</td>"
                  + "<td>" + contact.getProperty("phone") + "</td>"
                  + "<td><a href=\"contactdetails?id="
                 + KeyFactory.keyToString(contact.getKey())
                 + "\">details</a></td>"
                 + "</tr>");
      }
     out.println("</table>");
 }
 out.println("</body></html>");
     }
```

Publish in the web.xml file like /contactslist in a similar way as in /save (see section 3.1.4)

Run the server and point the browser to http://localhost:8888/contactslist .

3.3 See the contact details

Create a new servlet called ContactDetailsServlet, and create in it a doGet like the following:

```
if (req.getParameter("id") == null) {
    resp.getWriter().println("ID cannot be empty!");
    return;
}
// Get the datastore
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
// Get the entity by key
Entity contact = null;
try {
    contact = datastore.get(KeyFactory.stringToKey(req.getParameter("id")));
} catch (EntityNotFoundException e) {
    resp.getWriter().println("Sorry, no contact for the given key");
    return;
}
// Let's output the basic HTML headers
PrintWriter out = resp.getWriter();
resp.setContentType("text/html");
out.println("<html><head><title>Contacts list</title></head><body>");
            // Let's build the table headers
out.println("<table style=\"border: 1px solid black; width: auto; text-align: center;\">"
            + "<tr><td rowspan=3><img src=\""
            + contact.getProperty("pict")
            + "\" alt=\"Profile picture\" border=1/></td>");
out.println("<td>Name: </td><td>" + contact.getProperty("name") + "</td></tr>"
            + "<tr><td>Phone: </td><td>" + contact.getProperty("phone") + "</td></tr>"
            + "<tr><td>Email: </td><td><a href=\"mailto:"
            + contact.getProperty("email") + "\">"
            + contact.getProperty("email") + "</a>"
            + "</td></tr>");
out.println("</table><a href=\"save?id=" + req.getParameter("id")
                        + "\">Modify</a></body></html>");
}
```

Run the server again and access the details from the contacts list page.

Alternatively, copy the key of a contact from the Google App Engine datastore page and modify the id=ID parameter in the browser's URL.

3.4 Modify a contact

There is only 1 feature missing: **changing a contact**.

We already have the servlet to save a contact and, as a result if the name is the same, the action will be to modify the contact.

So, instead of creating a new servlet, we'll modify the already present servlet.

Let's add a doGet method to handle a request to modify.

Part of the code is very similar to the ContactDetailsServlet because we need to retrieve a contact and fill part of the form with the current data:

```
resp.setContentType("text/html");
// Let's output the basic HTML headers
PrintWriter out = resp.getWriter();
out.println("<html><head><title>Modify a contact</title></head><body>");
// Get the datastore
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
// Get the entity by key
Entity contact = null;
String name = "", phone = "", pict = "", email = "";
try {
    contact = datastore.get(KeyFactory.stringToKey(req.getParameter("id")));
    name = (contact.getProperty("name") != null) ? (String) contact.getProperty("name") : "";
    phone = (contact.getProperty("phone") != null) ? (String) contact.getProperty("phone") : "";
    email = (contact.getProperty("email") != null) ? (String) contact.getProperty("email") : "";
    pict = (contact.getProperty("pict") != null) ? (String) contact.getProperty("pict") : "";
} catch (EntityNotFoundException e) {
    resp.getWriter().println("<p>Creating a new contact</p>");
} catch (NullPointerException e) {
// id paramenter not present il the URL
resp.getWriter().println("<p>Creating a new contact</p>");
}
out.println("<form action=\"save\" method=\"post\" name=\"contact\">");
// Let's build the form
out.println("<label>Name: </label><input name=\"name\" value=\"" + name + "\"/><br/>"
            + "<label>Phone: </label><input name=\"phone\" value=\"" + phone + "\"/><br/>"
            + "<label>Email: </label><input name=\"email\" value=\"" + email + "\"/><br/>"
            + "<label>Picture: </label><input name=\"pict\" value=\"" + pict + "\"/><br/>");
out.println("<br/><input type=\"submit\" value=\"Continue\"/></form></body></html>");
}
```

Run again the server and click on the “Modify” link in the details.

Alternatively if no contact exist, go to http://localhost:8888/save to create a new contact. Then goto http://localhost:8888/contactlist, refresh and click details, and then modify

3.5 Present the service to the user

We can just give the user the chance to click on a link instead of remembering where to go to manage accounts.

Create or update the index.html in the directory war/ like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- The HTML 4.01 Transitional DOCTYPE declaration-->
<!-- above set at the top of the file will set     -->
<!-- the browser's rendering engine into           -->
<!-- "Quirks Mode". Replacing this declaration     -->
<!-- with a "Standards Mode" doctype is supported, -->
<!-- but may lead to some differences in layout.   -->

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
   <title>Contact list App Engine</title>
  </head>

  <body>
    <h1>Contact List</h1>

    <table>
      <tr>
        <td colspan="2" style="font-weight:bold;">Available Actions:</td>
      </tr>
      <tr>
        <td><a href="contactlist">Contacts List</a></td>
      </tr>
      <tr>
        <td><a href="save">Add contact</a></td>
      </tr>
    </table>

  </body>
</html>
```

And add in the web.xml file the following lines:

```
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

Now open http://localhost:8888/ .

3.6 Output format

For different use-cases it will be useful to give as a result an output different from “basic” HTML.

Focusing on mobile apps, let's see how to work with JavaScript Object Notation. JSON is a text-based open standard designed for human-readable data interchange. The JSON format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit

data between a server and web application, serving as an alternative to XML.

Open again ContactsListServlet.java, and modify the doGet method as follow:

```
…
// Let's output the basic HTML headers
 PrintWriter out = resp.getWriter();

 /** Different response type? */
 String responseType = req.getParameter("respType");
 if (responseType != null) {
     if (responseType.equals("json")) {
     // Set header to JSON output
                resp.setContentType("application/json");
                out.println(getJSON(contacts, req, resp));
                return;
     } else if (responseType.equals("xml")) {
                resp.setContentType("application/json");
                out.println(getXML(contacts, req, resp));
                return;
     }
 }
resp.setContentType("text/html");
…
```

As you can see we are looking in the HTTP request for a param *respType*. If present, the server will prepare the output according to the requested response type. If no response type is specified, the default will be HTML. Now, add the following methods:

```
private String getXML(List<Entity> contacts, HttpServletRequest req, HttpServletResponse resp) {
     // TODO Auto-generated method stub
return null;
}

private String getJSON(List<Entity> contacts, HttpServletRequest req, HttpServletResponse resp) {
     // Create a JSON array that will contain all the entities converted in a JSON version
     JSONArray results = new JSONArray();
     for (Entity contact: contacts) {
                JSONObject contactJSON = new JSONObject();
                try {
                          contactJSON.put("name", contact.getProperty("name"));
                          contactJSON.put("phone", contact.getProperty("phone"));
                          contactJSON.put("id", KeyFactory.keyToString(contact.getKey()));
                          contactJSON.put("email", contact.getProperty("email"));
                          } catch (JSONException e) {
                          // TODO Auto-generated catch block
                                    e.printStackTrace();
                          }
                          results.put(contactJSON);
                }
                return results.toString();
     }
```

Run the server and check the differences of these 2 URLs:

1. http://localhost:8888/contactslist
2. http://localhost:8888/contactslist?respType=json

Please note that JSONArray and JSONObject are from the package
com.google.appengine.labs.repackaged.org.json .

We can apply the same concept to show a contact's details in JSON format, just need to change the first part of the doGet method (before the resp.setContentType line).

4 Website for your app

1. Go to the war directory, and move the “index.html” to “service.html”
2. Copy the “/datas/teaching/courses/mobserv/fall2015/Lab_GAE/war/index.html” to “war”
3. Copy the website directory from “/datas/teaching/courses/mobserv/fall2015/Lab_GAE/war/website into the “war” directory. This provides you a template for introducing and marketing your mobile app.
4. Deploy and test the web service and the web site,

5 Questions

Write a summary of the lab session, and answer the following questions.

1. What are the main components of a servlet?
2. What is a web service?
3. In our addressbook example, what are the APIs available to the users?
4. How a URL is mapped to a class?
5. How the Contact details is accessed from the contact list?
6. How a contact can be deleted?
7. How the addressbook can be improved?

Note: please zip your report and project under gae.name1_name2_date.zip and put in under fall2015/Lab_GAE. Do not forget to add the URL of your server in the report.

6 Extras

6.1 Google App Engine Java Docs and API

https://cloud.google.com/appengine/docs/java/

https://cloud.google.com/appengine/docs/java/datastore/

https://cloud.google.com/appengine/docs/java/javadoc/

6.2 Methods about URL

```
// Returns the server name
request.getServerName();

// Returns the server port
request.getServerPort();

// Returns the name of the application hosting the servlet
request.getContextPath();

// Returns the servlet path
request.getServletPath();

// Returns the type of the HTTP request used
request.getMethod();

// Returns the parameters sent in the URL
request.getQueryString();

// Returns the URL used to contact the servlet
request.getRequestURL();

// Returns the local address
request.getLocalAddr();

// Returns the local name
request.getLocalName();

// Returns the local port
request.getLocalPort();

// Returns the remote address
request.getRemoteAddr();

// Returns the remote host
request.getRemoteHost();
```

6.3 Methods about paramters

```
// Parameter retrieving by the "parameterName"
request.getParameter(parameterName);

// Returns an enumeration of all the parameters of a request
request.getParameterNames();
```

```
// Returns all the parameter values of "parameterName"
request.getParameterValues(parameterName);

// Returns an iteration of the parameter names
request.getParameterMap();
```

6.4 Methods about the header

```
// Returns the header names
request.getHeaderNames();

// Returns the headers
request.getHeaders();

// Returns the number of bytes of the request
request.getContentLength();

// Returns the content type of the request
request.getContentType();

// Returns the default language
request.getLocale();

// Returns the list of languages
request.getLocales();

// Returns the header date
request.getDateHeader(String name);

// Returns the header specified by the "name"
request.getHeader(String name);

// Returns the headers specified by the "name"
request.getHeaders(String name);
```

6.5 Methods for adding attributes to request

```
// Store an object in the HttpServletRequest object
request.setAttribute(String name, Object o);

// Returns the stored object "name"
request.getAttribute(String name);

// Returns an enumeration of the attribute names
request.getAttributeNames();

// Remove the attribute "name" the attributes of the request
request.removeAttribute(String name);
```

6.6 Methods for adding information to response

```
// Indicates the nature of the information put into the response
```

```
response.setContentType(String type);

// Indicates the local language
response.setLocale(Locale loc);

// Adds the header "name" with the value "value"
response.addHeader(String name, String value);
```