Front-end / Back-end

How does your application communicate with services?

Mission

Help students implement a mock-up that actually gets (and sometimes stores) data using some kind of external service.

The idea is to simulate that the application actually gets data depending on what the user does, and when applicable stores data so that it is available next time the application runs.

Wish-list

I requested from your teachers a wish list for this lecture. The following were the requests they have gathered from you:

- How do we store “stuff”?
- Can we use dreamweaver?
- How does Android work?
- How does [tech buzz word X] work?

What we won’t squeeze in to these lectures

Two hours is not enough time to teach you:

- Android or any other Client frame work (you’ll need to work out the GUI together with the supervisors/teachers)
- How to create a full scale application server with tiers and fancy architecture
- All the inner workings of HTTP and web services
- Deployment and configuration of web services
- HTML, CSS, and web design

The details of the above is something you will learn during supervision.

What we aim to address in these lectures

There seems to be roughly two types of scenarios given the nature of your applications:

1. The application fetches and presents data to the user (ReadOnly)
2. The application both fetches and stores data (ReadWrite)

We'll look into a few means to fetch and store data over the network.

Read-only applications

An application that only reads data from some external source (e.g. looks up time table data from Västtrafik) does not need to send data for storing at a server.

Local settings may be saved, though (e.g. Favorite bus stop etc) but at the client side.

For android, you'll use the built-in SQLite database for storing per-app data. For a Swing application, you may save settings in a Properties object which can be saved to file (and later re-loaded when the application starts again).

For a web application (web page based interface) cookies could be used.

Read-only applications

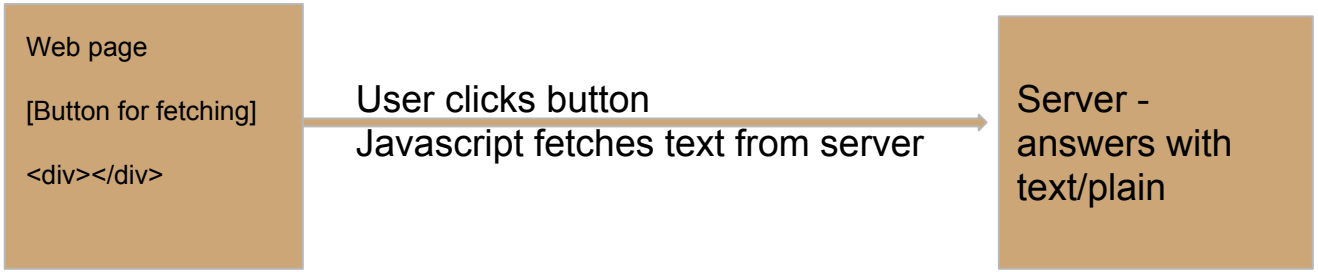
The application must however fetch data (e.g. performing a search) from some server or API (web service).

API? https://en.wikipedia.org/wiki/Web_API

Since both RO applications and RW applications need to fetch data over some kind of API, we'll start with some examples of how such web services (APIs) may look.

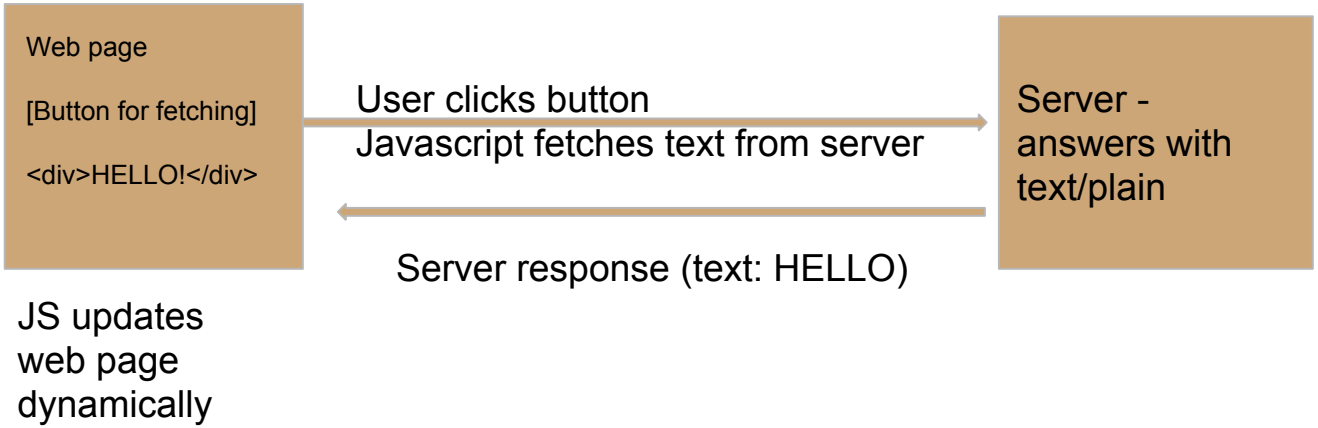
Fetching data 1 - AJAX (read-only part 1)

Simplest form of fetching data from a web page (for instance one of your mock-up applications with a web page implementation) is to use JavaScript to asynchronously fetch data as Text (or XML).



Fetching data 1 - AJAX

Simplest form of fetching data from a web page (for instance one of your mock-up applications with a web page implementation) is to use JavaScript to asynchronously fetch data as Text (or XML).

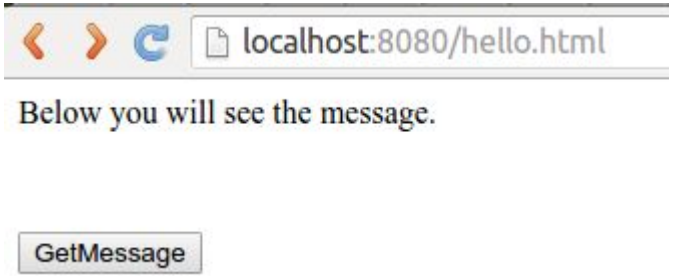


Ajax example

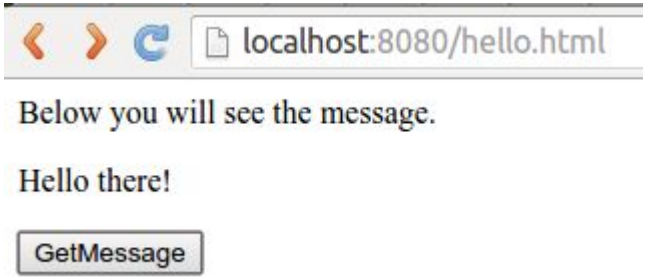
The simplest example is to react to a button click, and fetch a text string from a server and dynamically (without reloading) update an element in the page.

Idea: Present a web page with a button which when clicked fetches a text from a server and updates the web page to show that text.

Ajax example



Ajax example



What's dynamic about that?

The HTML code for the page is always the same. When the button is clicked, the view of the page changes (but not the code!).

This is done using JavaScript AJAX call to a server text resource, and updating "inner HTML" of a named element of the page.

When an element gets updated from JavaScript, it is the browser's representation of the page that gets updated (what the user sees), and not the HTML code of the actual page that the browser has downloaded.

The HTML what now?

This is the code of the page (not including the javascript):

```
<body>
<p>
Below you will see the message.
</p>
<p>
<div id="message"> </div>
</p>
<p>
<input type="submit" value="GetMessage" onClick="showMsg()" />
</p>
</body>
```

The HTML what now?

This is the code of the page (not including the javascript):

```
<body>
<p>
Below you will see the message.
</p>
<p>
<div id="message"> </div> <!--   is the inner HTML of the div -->
</p>                            <!-- with the id “message” -->
<p>
<input type="submit" value="GetMessage" onClick="showMsg()" />
</p>
</body>
```

Uh-huh - and what about that JavaScript?

This is the JavaScript code for the AJAX call to the server:

```
<html><head><script type="text/javascript">
function showMsg() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/getmessage", true);
    xmlhttp.send();
}
</script></head>
```

Uh-huh - and what about that JavaScript?

The highlighted part is similar to adding a listener/callback

```
<html><head><script>
function showMsg() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/getmessage", true);
    xmlhttp.send();
}
</script></head>
```

Uh-huh - and what about that JavaScript?

The highlighted part is an anonymous function (similar to actionPerformed)

```
<html><head><script>
function showMsg() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange =  function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/getmessage", true);
    xmlhttp.send();
}
</script></head>
```

Uh-huh - and what about that JavaScript?

The highlighted part is the async call to the server, requesting some text

```
<html><head><script>
function showMsg() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/getmessage", true);
    xmlhttp.send();
}
</script></head>
```

Uh-huh - and what about that JavaScript?

When the server has answered without errors, this code is actually run

```
<html><head><script>
function showMsg() {
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/getmessage", true);
    xmlhttp.send();
}
</script></head>
```

Uh-huh - and what about that JavaScript?

And when is the JavaScript run? When someone clicks the button!

```
<input type="submit" value="GetMessage"  onClick="showMsg()" />
```

What goes on behind the scenes here?

The browser requests hello.html from the server and shows it to the user.

The page contains an (almost) empty div element with an id “message”.

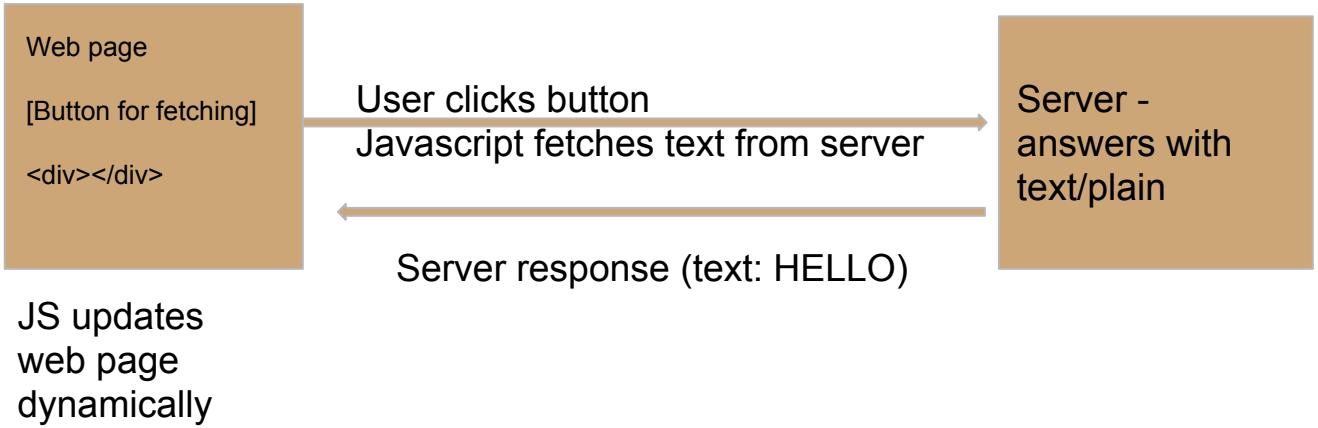
The page contains a javascript that calls the server and requests a resource (a text string in this case).

The page contains a button with the property onClick set to call the script.

The script requests the server for the text asynchronously and when it gets a response, only then is the view of the page updated to display the message.

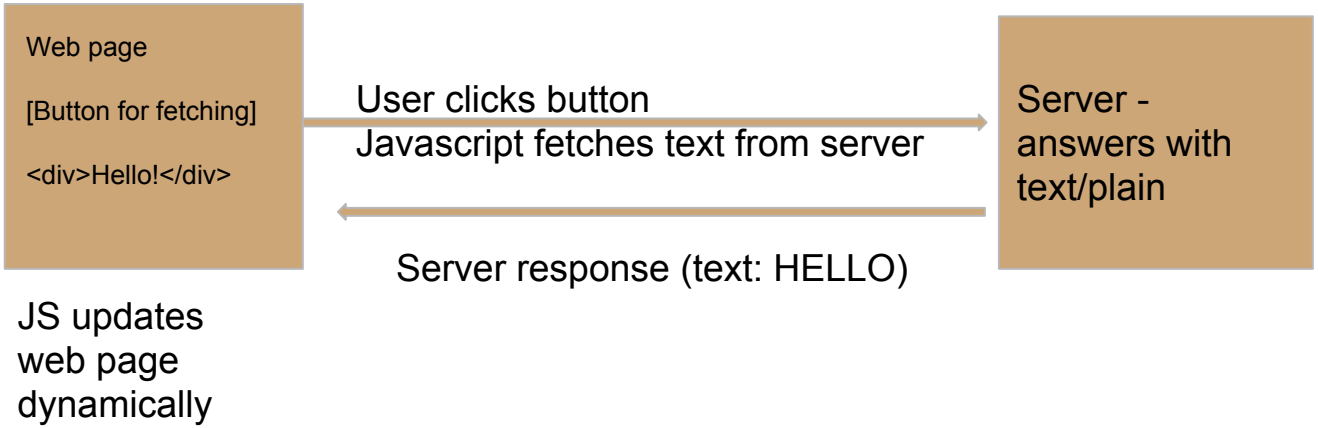
What goes on behind the scenes here? (reprise)

Dynamically updating a page with data from the server using AJAX:



What goes on behind the scenes here? (reprise)

Dynamically updating a page with data from the server using AJAX:



Break!

End of part 1 - Simple AJAX call for fetching text and dynamically updating page.

AJAX with GET parameters (read-only part 2)

It is convenient to have a server being able to reply with different data depending on a parameter sent from the client.

Let’s say our server is a little more intelligent than simply being able to respond with the same text every time.

Let’s also say our server is used as a data lookup for, say, how many minutes until the next BUS 16 arrives? (Everyone’s favorite bus!)

It depends on what bus stop we are at, and what direction of course.

The client must provide bus stop and direction!

What is a GET parameter?

HTTP is the protocol for sending web related data between clients and servers. A typical client is a browser, and a typical server is a web server.

GET parameters are part of the URL being requested:

`http://localhost:8080/nextbus?stop_id=2&direction=1`

The resource is /nextbus and the parameters occur after the ?

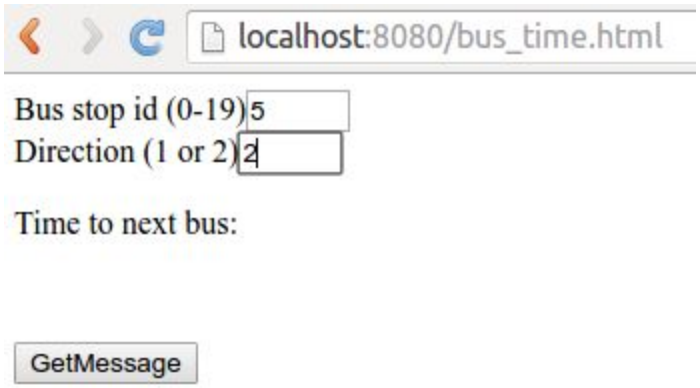
Multiple GET parameters are separated with &

Page has two text fields in a form

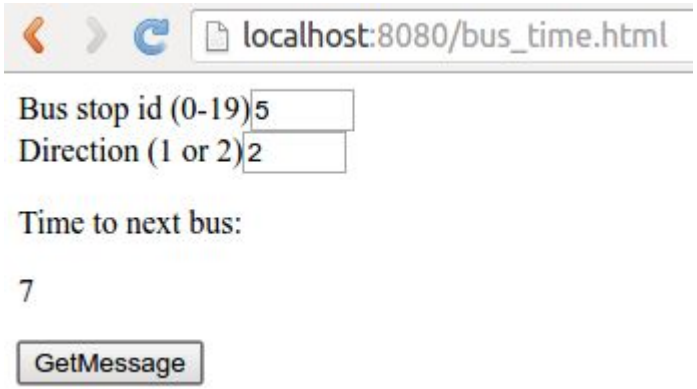
Idea here is to present the user with a form with two text fields, one for stop_id and one for direction.

For simplicity we only represent a bus stop as an integer id and the same for directions (1 or 2).

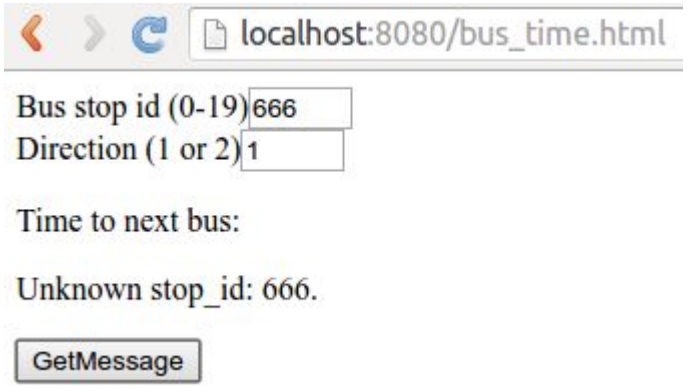
The application (web form)



The application (web form)



The application (web form)



How does the web service API work?

The endpoint is /nextbus

The parameters are:

stop_id (HTTP GET) - an id of a bus stop (an integer between 0-19)

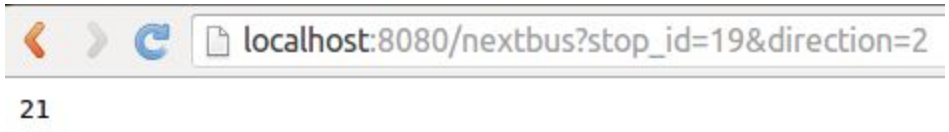
direction (HTTP GET) - the direction as an integer - 1 or 2

Example request:

/nextbus?stop_id=5&direction=2

Can we call the endpoint from a plain browser?

Of course we can! It is standard HTTP so all we need to do is type in the URL and supply the parameters, for instance:



Or from the command line:

```
$ lwp-request
'http://localhost:8080/nextbus?stop_id=19&direction=2'
21
```

What does the JavaScript function look like?

```
function showMsg() {
    var stop=document.getElementById("stop").value;
    var dir=document.getElementById("dir").value;
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/nextbus?direction="+dir+"&stop_id="+stop, true);
    xmlhttp.send();
}
```

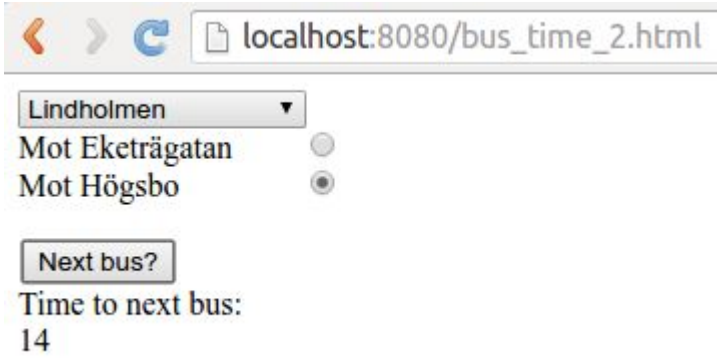
What does the JavaScript function look like?

```
function showMsg() {
    var stop=document.getElementById("stop").value;
    var dir=document.getElementById("dir").value;
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/nextbus?direction="+dir+"&stop_id="+stop, true);
    xmlhttp.send();
}
```

What does the JavaScript function look like?

```
function showMsg() {
    var stop=document.getElementById("stop").value;
    var dir=document.getElementById("dir").value;
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.onreadystatechange = function() {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        document.getElementById("message").innerHTML = xmlhttp.responseText;
      }
    };
    xmlhttp.open("GET", "/nextbus?direction="+dir+"&stop_id="+stop, true);
    xmlhttp.send();
}
```

Bonus - Another front-end for the same service



Note! Only the HTML front-end has changed. The server back-end remains the same.

Break!

End of part 2 - Ajax with GET parameters

What was the server at the “back-end”? (RO 3)

In this simple example with the next bus request, there was a simple Servlet responding to requests.

A servlet is a Java class capable of handling HTTP requests (as Java-objects) and writing HTTP responses (as Java-objects).

You need to have a Servlet container that listens to normal HTTP and forwards certain requests to the Servlet (and handles the response from the Servlet).

The simplest Servlet container (I know of)

Winstone is the simplest Java-container I know of. It is a stand-alone jar-file that you can run from the command line. All you have to do is tell it where the web root is located, and configure what servlet exists.

There is no time for the details in this lecture but the supervisors/teachers will help you with this (they have been notified about the topics here and will help you with the practical details).

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html
    |-- getmessage
    |-- hello.html
    `-- WEB-INF
        |-- classes
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage
    |-- hello.html
    `-- WEB-INF
        |-- classes
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html
    `-- WEB-INF
        |-- classes
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF
        |-- classes
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF ← this is the servlet stuff directory
        |-- classes
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF ← this is the servlet stuff directory
        |-- classes ← here are the servlet packages
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF ← this is the servlet stuff directory
        |-- classes ← here are the servlet packages
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib ← here you may put 3pp libraries e.g. sqlite.jar
        |-- servlet-api-2.5.jar
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF ← this is the servlet stuff directory
        |-- classes ← here are the servlet packages
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib ← here you may put 3pp libraries e.g. sqlite.jar
        |-- servlet-api-2.5.jar ← this file is for compiling servlets
        `-- web.xml
6 directories, 6 files
```

Servlet side directory layout

```
.
`-- webroot ← this is the / i.e. the root directory for the web server
    |-- bus_time.html ← this is the HTML page with the form
    |-- getmessage ← this is the textfile for the simple hello example
    |-- hello.html ← this is the hello webpage with the simple ajax req.
    `-- WEB-INF ← this is the servlet stuff directory
        |-- classes ← here are the servlet packages
        |   `-- org
        |       `-- ux
        |           `-- NextBus.java
        |-- lib ← here you may put 3pp libraries e.g. sqlite.jar
        |-- servlet-api-2.5.jar ← this file is for compiling servlets
        `-- web.xml ← this is the servet container configuration file
6 directories, 6 files
```

Servlet code

The interesting part of the servlet is the method called doGet().

It takes two parameters, request and response. The request is a Java-object representing the original HTTP request to the web server, the response is a Java-object representing the HTTP response we want to send back (via the web server).

Servlet code - the doGet() method

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
            throws ServletException, IOException{
  response.setContentType("text/plain");
  PrintWriter out = response.getWriter();
  StringBuilder page = new StringBuilder(253);
  String stopID = request.getParameter("stop_id");
  String direction = request.getParameter("direction");
  getTime(stopID, direction, page);
  out.println(page);
  out.close();
}
```

Servlet code - the doGet() method

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
            throws  ServletException, IOException{
  response.setContentType("text/plain");
  PrintWriter out = response.getWriter();
  StringBuilder page = new StringBuilder(253);
  String stopID = request.getParameter("stop_id");
  String direction = request.getParameter("direction");
  getTime(stopID, direction, page); // A method pretending to calculate time
  out.println(page);
  out.close();
}
// Shown in bold are classes you need to import for this method to compile
```

Running winstone

This is the command line I used for running the server in the examples:

```
java -jar [path-to]/jenkins-winstone-0.9.10-jenkins-47.jar --webroot=[path-to]/webroot
```

This is the compilation line used for the servlet:

```
javac -cp [path-to]/servlet-api-2.5.jar:. org/ux/NextBus.java
```

The servlet (as every real Java application) resides in a package.

The servlet imports stuff from javax.servlet - so we need that api on the class path in order to compile. The servlet jar may be anywhere on your computer.

Break!

Read-Write applications (R/W #1)

For applications which need to store data from the client, the server needs to handle also such requests. For small amounts of data you may use GET. For larger amounts, you need to use POST.

One other difference is that GET are visible as part of the URL request, whereas POST requests are not visible as part of the URL (they are only part of the HTTP request itself).

A servlet which saves user data

Idea: Send three parameters to the saveuser service which will save the user in some kind of storage.

Parameters:

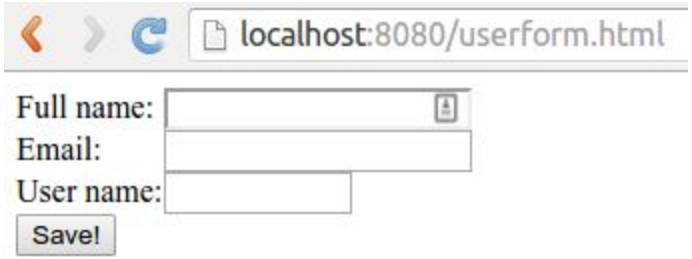
name (string with the real name of the user)

email (string with the email of the user)

user_name (string with the user name of the user)

Form for saving a user

This time we won’t use Ajax. We’ll keep things simple and use a plain form that uses GET to send the data to the service.



HTML for the form

```
<form action="/saveuser" method="GET">
<p>
<label>Full name:</label><input type="text" size="20" name="name" /><br />
</p>
<p>
<label>Email:</label><input type="text" size="20" name="email" /><br />
</p>
<p>
<label>User name:</label><input type="text" size="10" name="user_name" /><br />
</p>
<p>
<input type="submit" value="Save!" />
</p>
</form>
```

HTML for the form

```
<form action="/saveuser" method="GET">
<p>
<label>Full name:</label><input type="text" size="20" name="name" /><br />
</p>
<p>
<label>Email:</label><input type="text" size="20" name="email" /><br />
</p>
<p>
<label>User name:</label><input type="text" size="10" name="user_name" /><br />
</p>
<p>
<input type="submit" value="Save!" />
</p>
</form>
```

HTML for the form

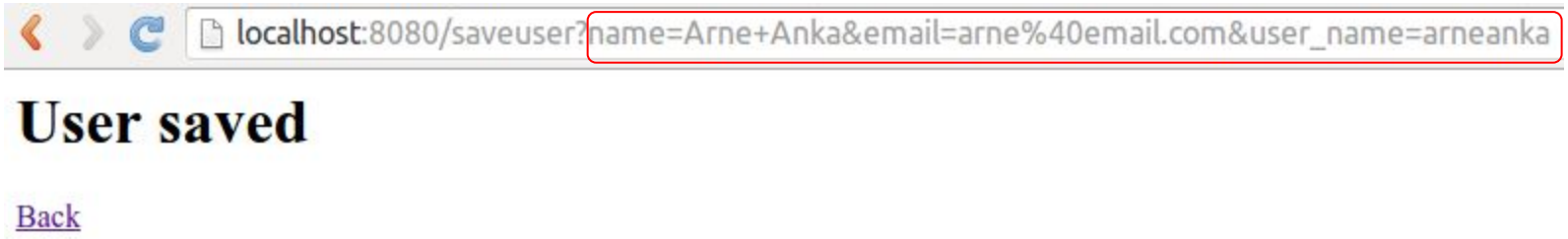
```
<form action="/saveuser" method="GET">
<p>
<label>Full name:</label><input type="text" size="20"name="name" /><br />
</p>
<p>
<label>Email:</label><input type="text" size="20"name="email" /><br />
</p>
<p>
<label>User name:</label><input type="text" size="10"name="user_name" /><br />
</p>
<p>
<input type="submit" value="Save!" />
</p>
</form>
```

Resulting landing page

Resulting landing page

action=”/saveuser”

Resulting landing page



GET parameters correspond to the name attributes of the input elements.

The values come from the user input.

What could go wrong?

Service is called without parameters:

Command line:

```
$ lwp-request -m GET http://localhost:8080/saveuser?
<html><body><h1>Required parameters missing</h1></body></html>
$ lwp-request -m HEAD http://localhost:8080/saveuser?
400 Bad Request
...
```

What could go wrong?

User name already exists:

(The username entry is primary key in the database of course, and therefore we can’t save the same username twice)

In this case *donaldd* was already in the database.

```
CREATE TABLE users(name text, email text, user_name text primary key);
```

A better error message

How could we do that? In the SQL tier, we could check the message of the SQLException caught:

```
String msg;
if(sqle.getMessage().startsWith("UNIQUE")){
  msg = "User name already taken.";
}else{
  msg = sqle.getMessage();
}
throw new StorageException(msg);
```

Break!

End of R/W #1

Json as information carrier

If a service response carries a lot of information it is common (and convenient) to package the information in some format. We'll touch upon Json format (and later XML).

Json - JavaScript object notation

```
{
    "students":[
        {
            "studentName":"Anders Andersson",
            "studentID":1
        },
        {
            "studentName":"Beata Bengtsson",
            "studentID":2
        },
        {
            "studentName":"Charlie Christensen",
            "studentID":3
        },
        {
            "studentName":"Dick Dale",
            "studentID":4
        },
        {
            "studentName":"Edward Eriksson",
            "studentID":5
        }
    ]
}
```

Json lingo

Objects are written between { and }

Objects can contain key-value pairs:

```
"studentName":"Anders Andersson"
```

Objects can contain arrays written between [and] and arrays consist of objects separated by , (comma)

Getting a list of Users as Json

Let’s say we want to retrieve all the users added using the previous example.

A list of users can be formatted as a Json document, and processed (“parsed”) by the client and turned back to Java User objects.

API

Endpoint: /userservice

Parameters: command - value “fetch_all” fetches all users as Json

Example URL:

/userservice?command=fetch_all

Test run and result

```
$ lwp-request -m GET http://localhost:8080/userservice?command=fetch_all
{
    "users":[
        {
            "Name":"Kalle Anka",
            "Email":"donald@email.dt",
            "UserName":"donaldd"
        },
        {
            "Name":"Joakim von Anka",
            "Email":"scrooge@email.dt",
            "UserName":"onkelscrooge"
        },
        {
            "Name":"Arne Anka",
            "Email":"arne@email.com",
            "UserName":"arneanka"
        }
    ]
}
```

Creating Json using Java

```
PrintWriter out = response.getWriter();
StringBuilder page = new StringBuilder();
Map<String, Boolean> config = new  HashMap<String, Boolean>();
config.put(JsonGenerator.PRETTY_PRINTING, true);
StringWriter writer = new StringWriter();
JsonWriterFactory jwf = Json.createWriterFactory(config);
JsonWriter jWriter = jwf.createWriter(writer);
JsonObjectBuilder job = Json.createObjectBuilder();
JsonArrayBuilder jab = Json.createArrayBuilder();
for( User user : storage.getAllUsers() ){
  jab.add(Json.createObjectBuilder()
       .add("Name", user.name())
       .add("Email", user.email())
       .add("UserName", user.userName()));
}
job.add("users", jab.build());
jWriter.writeObject(job.build());
jWriter.close();
page.append(writer.toString());    // You need to import  classes and interfaces in bold
```

Creating Json using Java (javax.json.jar)

```
PrintWriter out = response.getWriter();     ← Get the writer from the servlet response
StringBuilder page = new StringBuilder();  ← StringBuilder for building the Json text
Map<String, Boolean> config = new HashMap<String, Boolean>(); ← Used for pretty printing
config.put(JsonGenerator.PRETTY_PRINTING, true);
StringWriter writer = new StringWriter();   ← We want to write Json to a String
JsonWriterFactory jwf = Json.createWriterFactory(config); ← Factory for creating a JsonWriter
JsonWriter jWriter = jwf.createWriter(writer);            ← The JsonWriter
JsonObjectBuilder job = Json.createObjectBuilder();       ← Builder for a JsonObject
JsonArrayBuilder jab = Json.createArrayBuilder();         ← Builder for a JsonArray
for( User user : storage.getAllUsers() ){ ← Our List<User> from storage
  jab.add(Json.createObjectBuilder()      ← Add every JsonObject to the JsonArray
       .add("Name", user.name())
       .add("Email", user.email())
       .add("UserName", user.userName()));
}
job.add("users", jab.build());        ← Add the JsonArray to the root JsonObject
jWriter.writeObject(job.build());     ← Write the JsonObject to the JsonWriter
jWriter.close();
page.append(writer.toString());    // You need to import classes and interfaces in bold
```

Whatabout XML?

XML can also be used to format data in a response.

We'll show you next what the response would look like in XML.

Example request and XML response

```
$ lwp-request -m GET http://localhost:8080/userservicexml?command=fetch_all
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<users>
  <user>
    <name>Kalle Anka</name>
    <email>donald@email.dt</email>
    <username>donaldd</username>
  </user>
  <user>
    <name>Joakim von Anka</name>
    <email>scrooge@email.dt</email>
    <username>onkelscrooge</username>
  </user>
  <user>
    <name>Arne Anka</name>
    <email>arne@email.com</email>
    <username>arneanka</username>
  </user>
</users>
```

Creating XML from Java (javax.xml, org.w3c.dom)

```
PrintWriter out = response.getWriter();
response.setContentType("application/xml");
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
Document doc = docBuilder.newDocument();
Element rootElement = doc.createElement("users");
doc.appendChild(rootElement);
```

Creating XML from Java (javax.xml, org.w3c.dom)

```
for(User user : storage.getAllUsers() ){
  Element usr = doc.createElement("user");
  Element name  = doc.createElement("name");
  Element email = doc.createElement("email");
  Element userName = doc.createElement("username");
  name.appendChild(doc.createTextNode(user.name()));
  email.appendChild(doc.createTextNode(user.email()));
  userName.appendChild(doc.createTextNode(user.userName()));
  usr.appendChild(name);
  usr.appendChild(email);
  usr.appendChild(userName);
  rootElement.appendChild(usr);
}
```

Creating XML from Java (javax.xml, org.w3c.dom)

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty
     ("{http://xml.apache.org/xslt}indent-amount", "2");
DOMSource source = new DOMSource(doc);
StringWriter sw = new StringWriter();
StreamResult result = new StreamResult(sw);
transformer.transform(source, result);
out.println(sw);
```

Resources for further studies

Remember, just mentioning various techniques is not the same as teaching them ;-)

Not even showing sample code will teach you all the previous stuff.

You will only really learn when you try this out yourself, with the supervision and guidance of your supervisors/teachers.

Next is a list of some pointers for further studies etc.

Resources for further studies

http://www.w3schools.com/html/html_forms.asp
http://www.w3schools.com/js/js_intro.asp
http://www.w3schools.com/ajax/default.asp
http://www.w3schools.com/json/default.asp
http://www.w3schools.com/xml/default.asp
http://www.tutorialspoint.com/http/
http://www.javatpoint.com/servlet-tutorial

Resources for further studies

Materials from this lecture (source code etc) may be downloaded from:

https://github.com/progund/java-web/tree/master/extra-lectures/form-and-ajax

http://wiki.juneday.se/mediawiki/index.php/Java-Web:Servlet_as_back-end_-_HTML_and_AJAX_as_front-end