Web-based information systems 1

General introduction

Michiel Noback (NOMI)
Institute for Life Sciences and Technology
Hanze University of Applied Sciences

introduction

- The subject of this module is developing web applications
- We will investigate the power of dynamic web pages using server-side and client-side technology, and the ways they interact
- For the server-side, Java web technology was chosen
- For the client side, we'll fiddle around with HTML(5), CSS, Javascript and its libraries

Ten Simple Rules for Providing a Scientific Web Resource

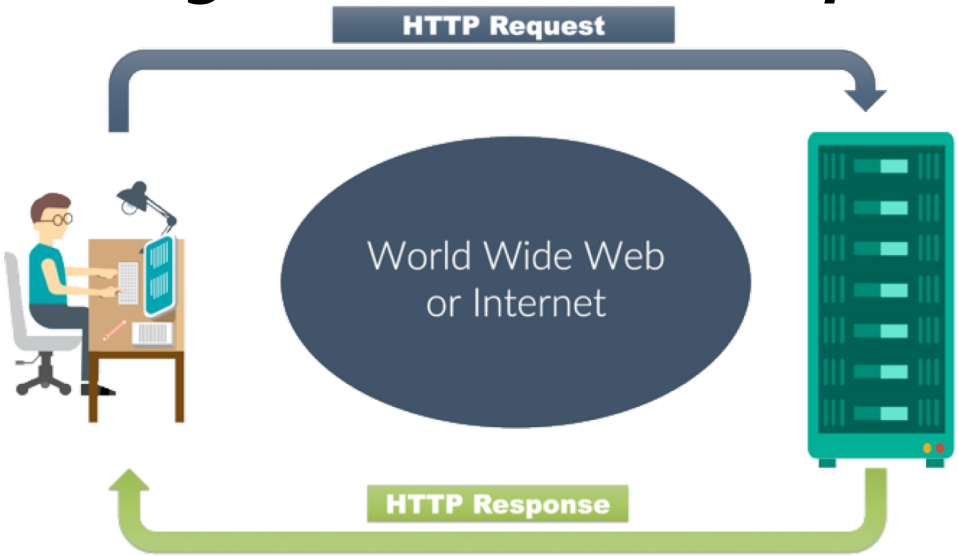
1. Plan Your Resource
2. Discuss Responsibilities
3. Know Your User Base
4. Use Services Available to You during Development
5. Ensure Portability
6. Create an Open Source Project
7. Provide Ample Documentation and Listen to Feedback
8. Facilitate Reproducibility
9. Plan Ahead: Long-Term Maintenance
10. Switch off an Unused Resource

tools

- these are the tools we are going to use to build our dynamic web applications:
  - **Java**: plain old Java to build the logic
  - **Servlets & JSP**: the server side Java technology to generate dynamic content
  - **Tomcat**: a container where the servlets live
  - **Javascript**: client-side scripting language to manipulate the view
  - **jQuery**: a Javascript library/platform for performing many tasks that are pretty hard to implement using raw Javascript
  - **AJAX**: to change the view without a page refresh (is actually a part of Javascript)
  - **css** (cascading style sheets): to style the view

the basics

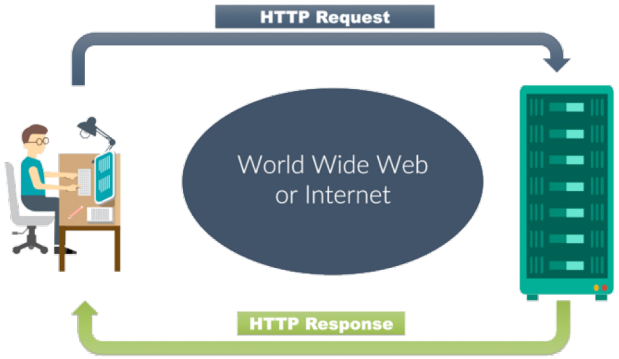
- when you click a simple link in a web page (the ***client***), you request a ***resource***
- the web ***server*** receives the request, locates the resource and returns something to the user: a ***response***



https://www.webnots.com/what-is-http/

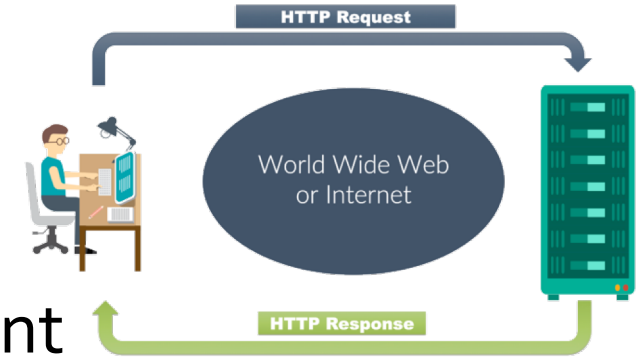
the client

- the client (usually a web browser) formats a request and sends it to the server
- the client receives the *response* and *processes* it
- when the response is html, the response is ***rendered*** by the browser into a web page ***view***



the server

- the server receives the request, locates the correct resource and formats the response before sending it to the client
- the server can **not**:
  - save anything
  - generate dynamic content
  - remember you or your previous requests
- these tasks have to be *delegated*



http request types

http requests come in 7 flavors:

Common requests (99.99999% of the use cases)

- **get** a simple resource fetch request
- **post** a request that sends data to the server or requests to change its data

There are others, not often used and not always supported because of security issues:

- **delete**
- **head**
- **options**
- **put**
- **trace**

http request with a form

```
<html>
  <head>
    <title>a page with a form</title>
  </head>
  <body>
    <h1>request bird pictures</h1>
    <form action="birdpic">
      Species name: <input type="text" name="species">
      <br />
      Minimum rating: <input type="text" name="rating">
      <br />
      <input type="submit" value="find">
    </form>
  </body>
</html>
```

If not specified, the http request type is GET

request bird pictures

Species name:

Minimum rating:

find

The HTML is rendered into this view

http GET request

Filling in the form with a GET request type and clicking ***find*** generates this *location* (URL)

...and this HTTP GET request

here have your parameters gone

```
GET ~michiel/WebBased/birdpic?species=Roodborst&rating=4 HTTP/1.1
Host: www.bioinf.nl
User-Agent: Mozilla/5.0(...)
Accept: text/xml.application/xml,(...)
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8,q=0.7,*;q=0.7
Keep-Alife: 300
Connection: keep-alife
```

http POST request

```
<form action="birdpic" method="POST" >
  Species name: <input type="text" name="species">
  Minimum rating: <input type="text" name="rating">
  <input type="submit" value="find">
</form>
```

Changing the method to POST

gives this no-params location…

…and this HTTP POST request

```
POST ~michiel/WebBased/birdpic HTTP/1.1
Host: www.bioinf.nl
User-Agent: Mozilla/5.0(...)
Accept: text/xml.application/xml,(...)
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8,q=0.7,*;q=0.7
Keep-Alife: 300
Connection: keep-alife

species=Roodborst&rating=4
```

your parameters are now located at the end of the request

http response

- the response coming back from the server will look like this:

```
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0AAB6CGSTGGC56DS3FF78; Path=/WebBased
Content-Type: text/html
Content-Length: 397
Date: Sun, 9 Nov 2009 02:50:40 CET
Server: Apache-Coyote/1.1
Connection: close

<html>
  <head>
  ...
</html>
```

This is the HTTP response header with a **status code** (200 / OK), a cookie and other important information about the response

This is the HTTP response body with the HTML that the browser is going to display

GET or POST

- use GET for (simple) requests that do not alter the data on the server side; these are called ***idempotent*** because they can be made over and over again without any effect on the server
- use POST for complex requests and requests that alter the data on the server side

URL

- URL stands for Uniform Resource Locator
- Every web resource has its own unique address in the URL format:

```
http://www.bioinf.nl:80/~michiel/WebBased/index.hml
```

http://	protocol
www.bioinf.nl	server
:80	port
~michiel/WebBased/	path
index.hml	resource

if you don't specify a port, it will default to the HTTP port which is 80

TCP ports

- The TCP port is a 16-bit number that identifies a specific software program on the server hardware
- A server can have up to 65536 different server apps running
- TCP port numbers from 0 to 1023 are reserved for well-known services:
  - 21 FTP (file transfer protocol)
  - 22 SSH (secure shell)
  - 25 SMTP (mail)
  - 80 HTTP (Hypertext Transfer Protocol)
  - 443 HTTPS
- Don't use these ports for your own server programs
- Tomcat usually runs on port 8080

static versus dynamic content

- Web servers can ONLY serve static content (html pages, pictures, xml documents etc)
- If you want dynamic content, you have to delegate this to another application

```
<html>
  <body>
    The current time is [insertTimeOnServer]
  </body>
</html>
```

a web server can't help you with this

Non-Java programs that can do this are called CGI scripts. Examples of languages used to write these are Perl, PHP, Python etc

the CGI way of dynamic content

- This is a PHP script printing current time:

```
<html>
  <body>
    The current time is
      <?php
       // Prints something like: Monday 8th of July 2005 03:12:46 PM
       echo date('l jS \of F Y h:i:s A');
      ?>
  </body>
</html>
```

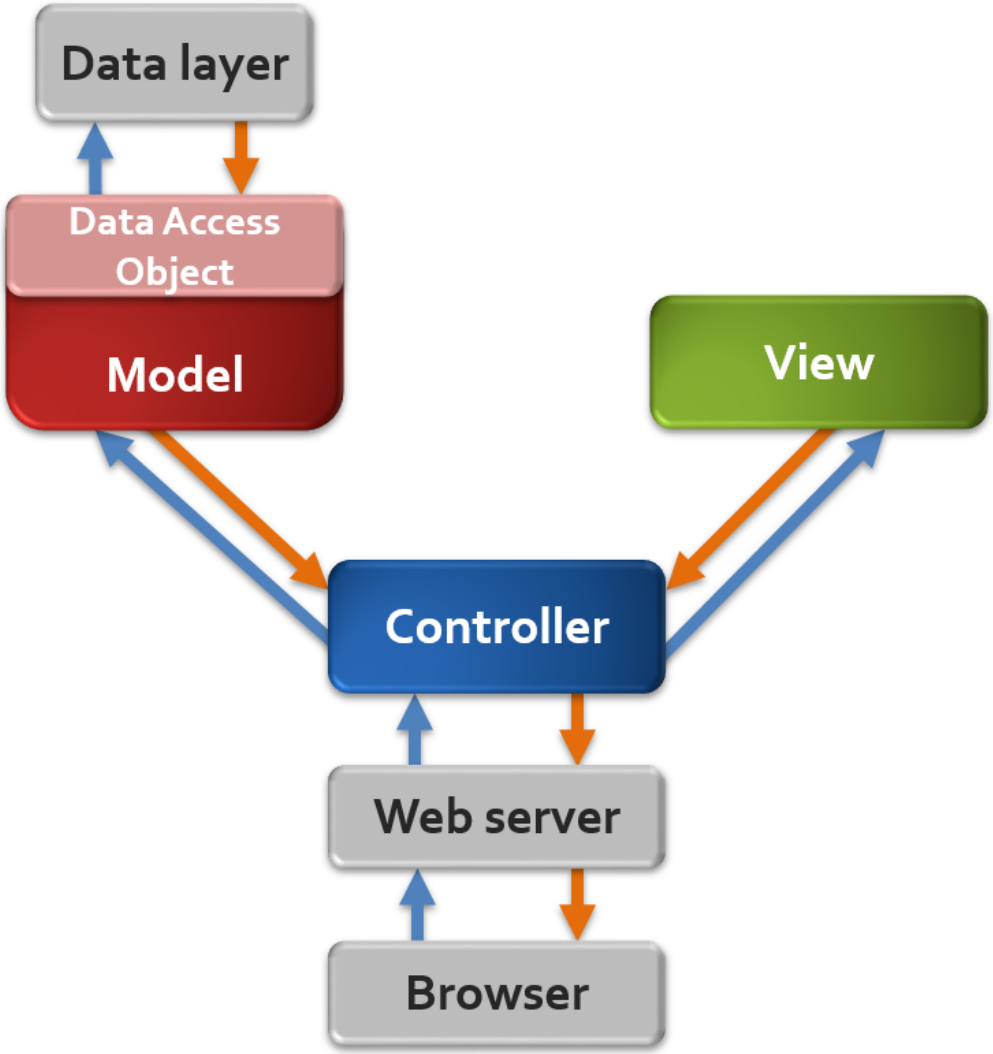
why use Java?

- Although Java is more of a fuss to get up and running, it is much better in serving heavy-duty analysis tools
- If you want a simple interface with forms accessing database data and presenting them, use Python or PHP
- If you want a real analysis tool running behind a web application, use Java
- But always…

separation of responsibilities

- Always ***separate the responsibilities*** of the view, the data and the model!
- ***Avoid mixing html, styling, scripting, database access and application logic at all costs!***
- This is the essence of the ***Model View Controller (MVC) design pattern (or paradigm)***

The MVC design pattern



MVC on different levels

- MVC can be applied on different levels
  - at the server for whole-application architecture (Servlet control – JSP view – Java model)
  - In a single web page DOM element: A Form HTML/css view, its Javascript controller and a Javascript model

“old” versus “modern” web apps

- In the old days, almost every action or resource requested from the server resulted in a completely new page
- Now, single-page web apps are becoming the standard.
- The key to this is Ajax: it lets you load and update only those parts of the page where this is required
- Also, Javascript can take a LOT of logic away from the server

preparing the environment: Tomcat

- To start working with Java web technology, you have to get Tomcat and install it:
  - go to http://tomcat.apache.org/ and download the latest version
  - extract it in a suitable location
  - To test, start up Tomcat using

    /tomcat/bin/startup.sh

    (you will need to make it executable first)
  - direct your browser to http://localhost:8080/ and you should see the Tomcat welcome page

preparing the environment: Java dev kit

- Besides Tomcat you will need:
  - Java EE 6 SDK
  - IntelliJ Idea Ultimate >=2017 (You can get a free educational license)
  - A MySQL server and account (you can set this up at home using LAMP or WAMP)
  - A modern web browser, preferably Firefox or Chrome

Debugging web sites

- All modern browsers have extensive debugging support
- Do not use the JavaScript function `alert("my debug message")`, but `console.log("my debug message")`
- Use the context menu option “Inspect element” in the browser:

The Chrome/Firefox inspectors

- Use the inspector to investigate, change and track DOM structure, bugs, network traffic, styling information, print info from JavaScript, change JavaScript variables…