

# Database-enabled web technology

## Web Programming

Instructor: Çağrı Çöltekin

`c.coltekin@rug.nl`

Information science/Informatiekunde

Fall 2012/13

## About initial project reports

Your initial report should describe what you are going to implement. It should include,

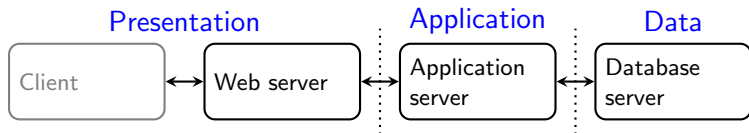
- ▶ the requirements
- ▶ your initial database design

You will get feedback within a few days after you submit your report.

## So far...

- ▶ A quick introduction to handling forms in PHP.
- ▶ Another introduction on `git` version management system.

# The multi-tier (or 3-tier) architecture

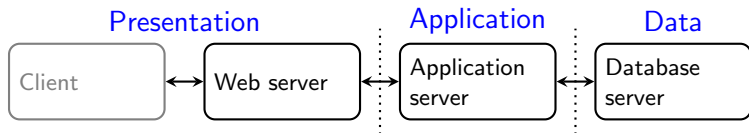


**Presentation** tier interacts with the user (e.g., ask the seat preference in an airline online check-in system).

**Application** tier implements the 'business logic' (e.g., check and reserve a seat, possibly using multiple queries and updates).

**Data** tier stores the data (e.g., retrieve and/or update the relevant data records).

# The multi-tier (or 3-tier) architecture



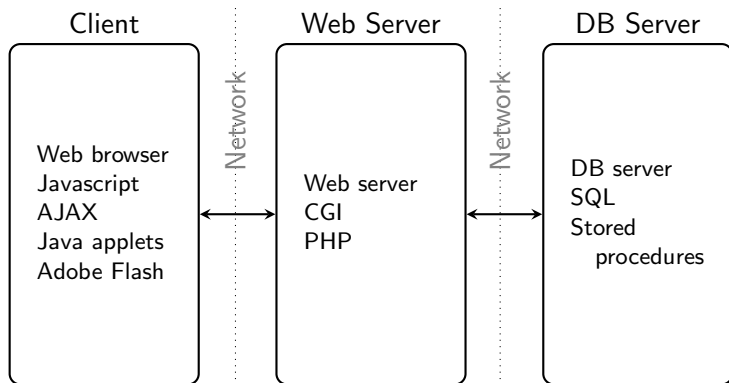
**Presentation** tier interacts with the user (e.g., ask the seat preference in an airline online check-in system).

**Application** tier implements the 'business logic' (e.g., check and reserve a seat, possibly using multiple queries and updates).

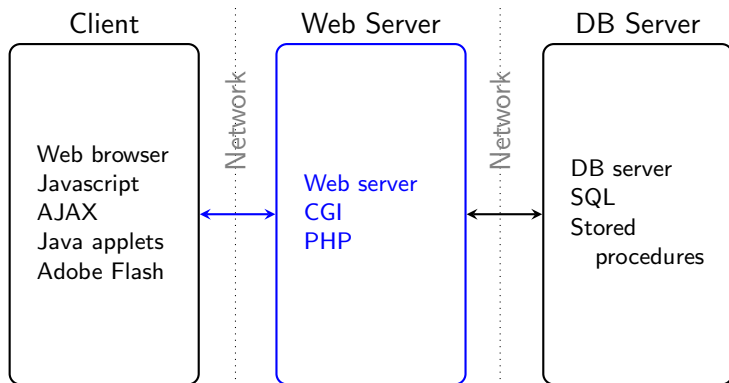
**Data** tier stores the data (e.g., retrieve and/or update the relevant data records).

In practice, division may not match the figure above. However separating presentation from application is always a good idea.

# Web-programing model: what runs where



# Web-programing model: what runs where



In this lecture we will study the client-server interaction and server-side programming

# What happens when you click on a link?

C Extract the URL: `http://www.rug.nl/let/informatiekunde`

C: Client, S: Server



## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`
- S Read the request, process it

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`
- S Read the request, process it
- S Form a response and send it

C: Client, S: Server



## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`
- S Read the request, process it
- S Form a response and send it
- C Read the response, process it

C: Client, S: Server

## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`
- S Read the request, process it
- S Form a response and send it
- C Read the response, process it
- C Close the connection

C: Client, S: Server

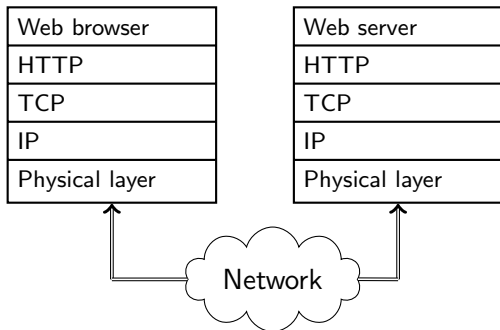
## What happens when you click on a link?

- C Extract the URL: `http://www.rug.nl/let/informatiekunde`
- C Parse the URL: Host: `www.rug.nl`, Protocol: `HTTP`, Resource: `/let/informatiekunde`
- C Resolve the host name: `129.125.2.51`
- C Find the default port number for `HTTP`: `80`
- C Open a `TCP` connection to the `IP:port`
- S Accept the connection
- C Send the `HTTP` request: `GET /let/informatiekunde HTTP/1.1...`
- S Read the request, process it
- S Form a response and send it
- C Read the response, process it
- C Close the connection

This is still an overview, a lot more happens under the hood.

C: Client, S: Server

## Layers in communication



- ▶ Web server and web browser talks to each other using HTTP.
- ▶ The HTTP messages goes through a set of networking layers.
- ▶ We are mainly interested in HTTP, some aspects of TCP/IP networking is relevant to web programming.

## Three-slide introduction to TCP/IP (1)

- ▶ TCP/IP is the name of the network protocol family used in the Internet.
- ▶ It is more than TCP and IP. Just to list a few: UDP, BGP, DHCP, ICMP, DNS, ...
- ▶ The IP protocol is connectionless, it does its best to deliver a network packet to its destination.
- ▶ IP does not guarantee the delivery of every packet.
- ▶ TCP works on IP, it is connection oriented. With TCP, you do not worry about the lost packets.

## Three-slide introduction to TCP/IP (2)

- ▶ The hosts in a TCP/IP network is identified with a unique IP address.
- ▶ IP(v4) addresses are 4-byte integers, e.g., 129.125.2.51 (New version of IP, IPv6, uses longer IP addresses).
- ▶ DNS maps human readable domain names, like `www.rug.nl` to IP addresses.

DNS can be used for distributing load:

A particular host name can be assigned multiple IP addresses. For each DNS query, a DNS server will issue one of the IP addresses in a round-robin fashion

## Three-slide introduction to TCP/IP (3)

- ▶ Commonly used services have reserved port numbers, for example 80: HTTP, 443: HTTPS, 22: SSH . . .
- ▶ A server typically 'listens' on a reserved port for client connections.
- ▶ Clients reserve temporary port numbers.
- ▶ Each end of a connection is identified by IP address/port number pairs.
- ▶ Connection is typically initiated by a client, any of the parties can close the connection.

# Anatomy of a URL

`http://www.rug.nl:80/let/informatiekunde`

①

②

③

④

- ① **Scheme** indicates the protocol. The rest of the URL may be different depending on the scheme.
- ② **Domain name** is the name of the host where the HTTP service runs.
- ③ **Port number** can optionally be given in cases where the service do not run on the default port.
- ④ **Path** typically identifies the (HTML) files on the server, but can express more than a file name. The interpretation is dependent on the web server.



# HTTP: an overview

- ▶ HTTP is a request-response protocol. Clients asks for an operation on a resource, possibly with some content, and server responds, likely with some content.
- ▶ The requested operation has to be one of 9 HTTP **methods**, like **GET**, **HEAD** or **POST**.
- ▶ Response message starts with a status message.
- ▶ Both request and response can include additional **headers**, which provide additional information.
- ▶ HTTP protocol does not encrypt the communication, nor has it any mechanism to verify the identity of server or the client.
- ▶ HTTPS is an extension of HTTP that uses Secure Socket Layer (SSL).

# HTTP requests

```
1 GET / HTTP/1.1
2 Host: www.rug.nl
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:8.0) ...
4 Accept: text/html,application/xhtml+xml, ...
5 Accept-Language: en-us,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Accept-Charset: UTF-8,*
8 Connection: keep-alive
9 Cookie: acceptedLanguages=en; s_nr=1321910886052 ...
10
```

- ▶ First line is the actual request, here using method **GET**.
- ▶ The rest of the lines are headers that provide additional information.
- ▶ The empty line (10) is important. It signals the end of headers.

# HTTP response

```
1 HTTP/1.1 200 OK
2 Date: Wed, 23 Nov 2011 01:11:25 GMT
3 Server: Apache/2.2
4 Last-Modified: Tue, 11 Mar 2008 11:35:02 GMT
5 Content-Length: 260
6 Content-Type: application/html
7
8 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML ...
9 <html>
10 ...
```

- ▶ The first line is the status line.
- ▶ Again, server gives us a set of header lines followed by an empty line and the content.
- ▶ The response can also indicate a permanent or temporary error, or a redirection message.

# HTTP methods

HTTP standard defines 9 methods, but we are only interested in

**GET** Typically used to get a static content (e.g., file). But it can also be used for dynamic content (we will return to this).

**HEAD** It is like GET, but server only responds with headers, no content is transferred.

**POST** is used when client needs to transfer some content. Typically content is the name/value pairs from a HTML form. However, it can be anything that server/client agree on.

Others (for the sake of completeness): PUT, DELETE, OPTIONS, CONNECT, PATCH, TRACE.

# HTTP headers

- ▶ Both requests and responses may be of interest for server-side programming.
- ▶ Request headers include users' preferences, such as `Accept-Language` or certain information about users' environment that you may want to know, such as `User-agent`.
- ▶ You can set response headers, to communicate with the browsers. For example, `Refresh` will instruct the browser to reload the page after specified time, or `Cache-Control` gives you a way to tell the browser if/how long the content can be cached.

# HTTP Cookies

- ▶ A specific HTTP header field `Cookie` (in request) or `Set-Cookie` (in response), is widely used for web programming.
- ▶ A cookie is a piece of information a HTTP servers asks the client to retain until a specific expiry date.
- ▶ The server sends a cookie to a client using, `Set-Cookie: name=val, expires=datetime, domain=d, path=p`
- ▶ The client (if enabled) sends the matching cookie that are not expired with every request.
- ▶ Cookies are typically used for session management, (some form of) authentication, or applications like shopping charts.

# HTTP Cookies

- ▶ A specific HTTP header field `Cookie` (in request) or `Set-Cookie` (in response), is widely used for web programming.
- ▶ A cookie is a piece of information a HTTP servers asks the client to retain until a specific expiry date.
- ▶ The server sends a cookie to a client using,  
`Set-Cookie: name=val, expires=datetime, domain=d, path=p`
- ▶ The client (if enabled) sends the matching cookie that are not expired with every request.
- ▶ Cookies are typically used for session management, (some form of) authentication, or applications like shopping charts.

## A summary so far

- ▶ The WWW, and as a result, the web-programming environment works over HTTP.
- ▶ HTTP is a request-response protocol.
- ▶ A HTTP request is originated by a client (e.g. browser) and includes a method, and a set of headers.
- ▶ A HTTP response includes a status code, additional headers, and the content.
- ▶ A server-side web-application (whether it is a CGI program, or a embedded interpreter) has access to raw HTTP data sent by the client, and form the response the way it wants.
- ▶ You will not typically deal with the raw HTTP messages, but knowing what lies underneath helps.



## SQL and programming

- ▶ SQL has limited use unless combined with a general purpose programming language.
- ▶ SQL has the advantage that it abstracts away the way data is stored from the application.
- ▶ However, it cannot do many things that a typical application program would require. Just to list a few:
  - ▶ arbitrary computation
  - ▶ flexible I/O, user interaction
  - ▶ formatted input output
  - ▶ graphical presentation of data
- ▶ There are a number of ways to combine SQL and general purpose programming
  - ▶ On DB side: [stored procedures](#).
  - ▶ On application side: [embedded SQL](#), or [call-level interfaces](#)
- ▶ We will be using call-level interfaces in this course.

## A first PHP/MySQL example

```
1  <?php
2      $host="hostname";
3      $user="username";
4      $pass="password";
5      $db="dbname";
6      mysql_connect($host,$user,$pass);
7      mysql_select_db($db);
8      $q = 'select * from book';
9
10     $res = mysql_query($q);
11     echo "<table border=\"1\">";
12     echo "<tr><th>ISBN</th><th>title</th></tr>";
13     while ($row = mysql_fetch_assoc($res)) {
14         echo "<tr><td>${row['ISBN']}</td>";
15         echo "<td>${row['title']}</td></tr>";
16     }
17     echo "</table>";
18     mysql_close();
19  ?>
```

## DB access from PHP using Pear DB

- ▶ There are multiple ways of connecting to databases, even multiple methods to connect to the same DBMS (For example, MySQL `mysql_` and `mysqli_` interfaces).
- ▶ We will follow a unified approach through Pear `DB` library.
- ▶ `DB` library allows a unified way to access different database management systems.
- ▶ A newer version, `MDB2`, with some improvements exists. We will stick to better-known and wide-spread `DB` library. You should consider `MDB2` if it is available.

## Pear DB: a first example

```
1 <?php
2     require_once('DB.php');
3     require_once('db-config.php');
4     $conn = DB::connect("mysql://$user:$pass@$host/$db");
5
6     $res = $conn->query('select * from book');
7
8     echo "<table border=\"1\">";
9     echo "<tr><th>ISBN</th><th>title</th></tr>";
10    while ($row = $res->fetchRow(DB_FETCHMODE_ASSOC)) {
11        echo "<tr><td>${row['ISBN']}</td>";
12        echo "<td>${row['title']}</td></tr>";
13    }
14    echo "</table>";
15    $conn->disconnect();
16    ?>
```

## Pear DB: the DSN

```
phptype://user:pass@proto+host:port/db?opt=val&opt=val...
```

**phptype** DB connection type (e.g., mysql, mysqli, odbc, sqlite, ...)

**username** DB user name.

**password** Password to connect to the DB

**protocol** Connection protocol (e.g., tcp, unix, ...)

**hostspect** Host name (or IP address)

**port** Port number of the DB server

**database** Name of the database

**option=value** additional options for the DB connection

## Pear DB: connecting to a database

- ▶ `DB::connect()` takes a `DSN` argument and an optional set of `options`, and returns a connection handler if successful.
- ▶ `DSN` can be specified either as a string, or an array of the form

```
$dsn = array('phptype' => 'mysql',  
            'hostspec' => 'localhost',  
            ... )
```

- ▶ A second optional argument can be used to specify options like debug level, various portability options etc.

```
$options = array(  
    'debug' => 0,  
    'portability' => DB_PORTABILITY_LOWERCASE,  
    ... )
```

- ▶ You can also set the options later using `setOption()`.

# Error Handling

- ▶ The database operations do not always get executed successfully. You should check for errors.
- ▶ `PEAR::isError()` returns true if given DB object is an error.
- ▶ `getMessage()` can be used to get the error message.
- ▶ The same functions are usable on other DB objects, for example query result.

```
$db = DB::connect($dsn, $options);  
if (PEAR::isError($db)) {  
    die($db->getMessage());  
}
```

## Pear DB: query

- ▶ Pear DB supports a number of query functions. `query()` is the simplest form.
- ▶ `query()` returns a 'result handler'.
- ▶ `fetchRow()` can be used to process each row in a result.
- ▶ `numCols()`, and `numRows()` give the number of columns and rows returned for a query. `affectedRows()` gives the number of rows affected by an `insert` or `update`.

```
$q = "insert into book (ISBN, title)
      values ($isbn, $title)";
$res = $db->query($q);
if (PEAR::isError($res)) {
    echo "insert failed: $res->getMessage()";
} else {
    echo "$res->affectedRows() lines changed";
}
```



# Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!

# Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!
- ▶ Consider the statement:  
`insert into book values ('$isbn', '$title')`.  
where we take user input  
`The Hitchhiker's Guide to the Galaxy`

# Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!

- ▶ Consider the statement:

```
insert into book values ('$isbn', '$title').
```

where we take user input

```
The Hitchhiker's Guide to the Galaxy
```

- ▶ The SQL statement, after PHP replaces the values will be:

```
insert into book values ('0330258648',  
    'The Hitchhiker's Guide to the Galaxy')
```

# Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!

- ▶ Consider the statement:

```
insert into book values ('$isbn', '$title').
```

where we take user input

```
The Hitchhiker's Guide to the Galaxy
```

- ▶ The SQL statement, after PHP replaces the values will be:

```
insert into book values ('0330258648',  
                          'The Hitchhiker's Guide to the Galaxy')
```

- ▶ This is an invalid statement. We want

```
'The Hitchhiker\'s Guide to the Galaxy'
```

# Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!

- ▶ Consider the statement:

```
insert into book values ('$isbn', '$title').
```

where we take user input

```
The Hitchhiker's Guide to the Galaxy
```

- ▶ The SQL statement, after PHP replaces the values will be:

```
insert into book values ('0330258648',  
                          'The Hitchhiker's Guide to the Galaxy')
```

- ▶ This is an invalid statement. We want

```
'The Hitchhiker\'s Guide to the Galaxy'
```

## Input validation

- ▶ Not validating user input introduces bugs, and possible security problems!

- ▶ Consider the statement:

```
insert into book values ('$isbn', '$title').
```

where we take user input

```
The Hitchhiker's Guide to the Galaxy
```

- ▶ The SQL statement, after PHP replaces the values will be:

```
insert into book values ('0330258648',  
                        'The Hitchhiker's Guide to the Galaxy')
```

- ▶ This is an invalid statement. We want

```
'The Hitchhiker\'s Guide to the Galaxy'
```

This is also a security risk (to which we will return later).

## Input validation in Pear DB

- ▶ Pear DB provides functions for escaping strings depending on DBMS in use.
- ▶ `escapeSimple()` escapes the string values.
- ▶ `escapeSmart()` also converts numeric/boolean values if necessary (e.g., convert `true` to `1` if required by the DBMS).
- ▶ `quoteIdentifier()` quotes identifiers, for example table or column names. For example, if you are using a reserved word, or space in a column name.
- ▶ **You should always escape user input in your SQL statements.**

```
$q = "insert into book values ('"  
    . $db->escapeSimple($isbn) . "', '"  
    . $db->escapeSimple($title) . "'");
```

## Pear DB prepare/execute

- ▶ For SQL statements that are used multiple times with different values, an alternative to `query()` is using `prepare()` and `execute()` functions.
- ▶ `prepare()` takes a query string with missing values:  
`$qh = $db->prepare('insert into book values(?,?)');`
- ▶ `execute()` takes the handle returned by `prepare()` and an array of values, and replaces the `?` with the values in the array:  
`$db->execute($qh, array($isbn, $title));`
- ▶ `prepare()/execute()` automatically escapes the input.
- ▶ If you do not want that for some reason, you can use `!` instead of `?`.
- ▶ If you want to pass contents of a file as is (e.g., image data), use `&` in `prepare()` and give the filename in `execute()`.



# Summary & Next week

This week:

- ▶ The HTTP protocol, and a bit of TCP/IP.
- ▶ HTML cookies, and how to use them in PHP.
- ▶ Using databases through PHP (mainly pear DB).

Next:

- ▶ Session management.
- ▶ More on databases & PHP.
- ▶ (possibly) some notes on security.