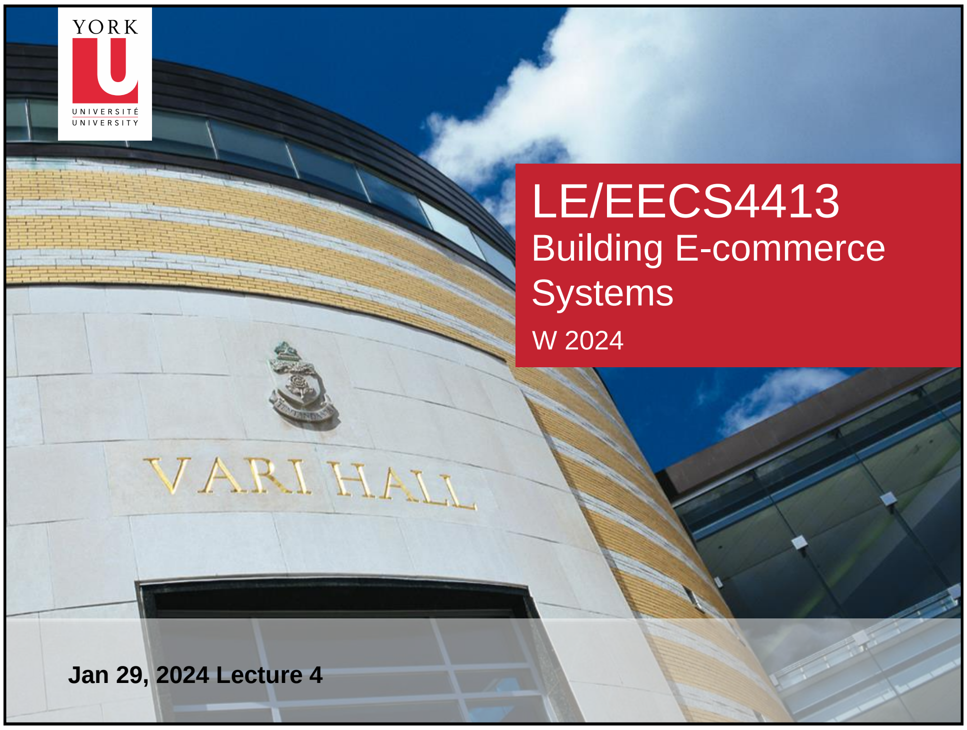


LE/EECS4413
Building E-commerce Systems
W 2024

Jan 29, 2024 Lecture 4

1

Main topics (tentative)

- Web App Architecture. Preliminary knowledge/Review
  - Client side: HTML CSS JavaScript
  - UML, design patterns, Java (cmd, thread, serialization),
- Client-Server, low level: socket programming
- Web applications (server side)
  - LAMP/CGI
  - Java Servlet
  - JSP, JavaBean, MVC pattern
  - SQL, Database access: JDBC. JPA
  - More: listener, filter, Ajax, JSON
- Web (RESTful) services, micro services
- Advanced topics (TBD): Deployments: Docker container, Node JS, React, Angular, Spring
- Other advanced topics (TBD) More design patterns, Performance, security

Introduction 3

3

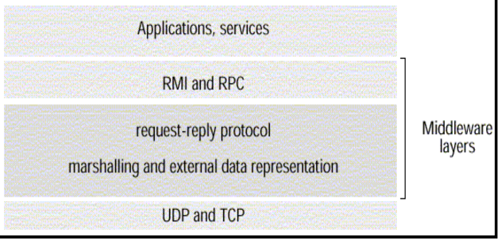
Protocols on sockets

- Low-level Sockets provide a “direct” way for two processes to send and receive data
  - Requires a protocol of formatting the messages exchanged so that the receiving process to know what to do with the received message
- Designing such a protocol is hard and error-prone (how to avoid deadlock?)
- **RPC** is a remote procedure call (RPC) protocol which uses some languages to encode its calls and HTTP as a transport mechanism
- BUT in case of user defined structured data these mechanisms, require a significant coding effort;
- **RMI** is a way of a client process to reference a remote object as if it was local, and call one or more of its methods

4

Protocols on sockets

- **RPC** is a remote procedure call (RPC) protocol which uses some languages to encode its calls and HTTP as a transport mechanism
- XML-RPC and JSON-RPC are frameworks that allow for standardized JSON or XML based messages to be exchanged between a client and server over HTTP for a sending process (client) to call a service (i.e. operation) on the server process
  - JSON and XML are meta-languages to define the content of the messages
  - JSON and XML strings can be transformed to Java objects and vice versa
- BUT in case of user defined structured data these mechanisms, require a significant coding effort;
- **RMI** is a way of a client process to reference a remote object as if it was local, and call one or more of its methods



5

RMI

- **Remote Method Invocation (RMI)** is a bit different from RPC
- It is also a full server-side service offering model
- The idea here is that a client does not issue a plain message to a service that listens to a port in the server, but references a "remote" object and calls a method of this remote object remotely (hence Remote Method Invocation)
- RMI is usually defined between Java programs. The CORBA (Common Object Request Broker Adapter) framework specification allowed RMI between programs written in arbitrary languages (for the ones there were interface adapters)

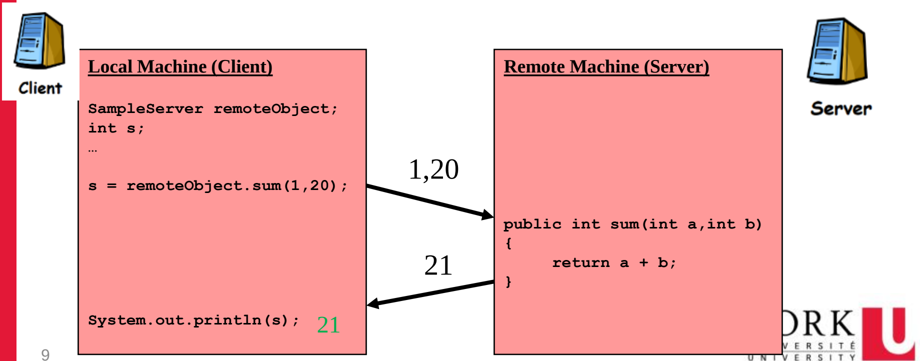
7

RMI

- RMI = Remote Method Invocation.
- Allows Java programs to invoke methods of remote objects.
- Only between Java programs.
- Several versions (JDK-1.1, JDK-1.2)
- In RMI, methods of remote objects can be invoked from other JVMs
- In doing so, the programmer has the illusion of calling a local method (but all arguments are actually sent to the remote object and results sent back to callers)

8

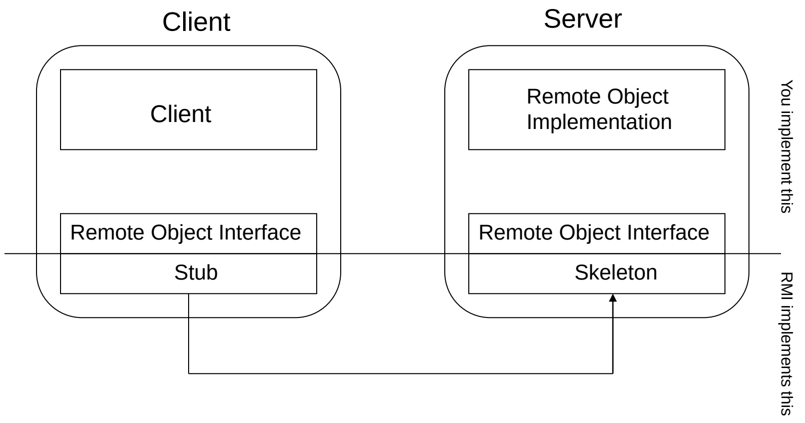
- In RMI, methods of remote objects can be invoked from other JVMs
- In doing so, the programmer has the illusion of calling a local method (but all arguments are actually sent to the remote object and results sent back to callers)



```
SampleServer remoteObject;
int s;
…

s = remoteObject.sum(1,20);

System.out.println(s);
```

```
public int sum(int a,int b)
{
      return a + b;
}
```

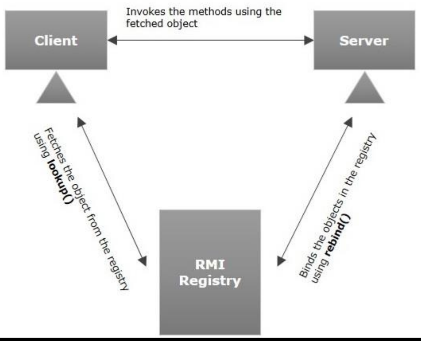
Remote Method Invocation



RMI Registry

- RMI registry is a namespace on which all server objects are placed.
- Each time the server creates an object, it registers this object with the RMI registry using **bind()** or **reBind()** methods). These are registered using a unique name known as **bind name**.

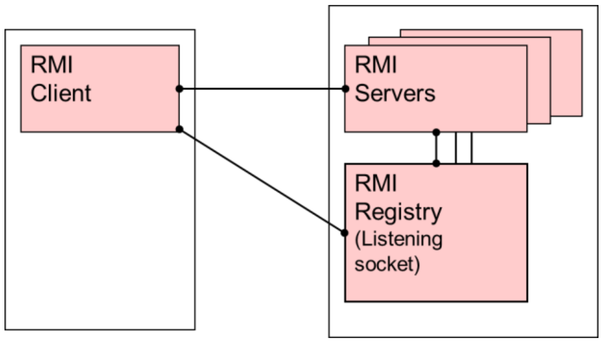
- To invoke a remote object, the client needs a reference of that object. At that time, the client fetches the object from the registry using its bind name (using **lookup()** method).

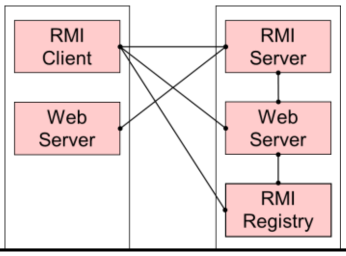

Client
Server
Invokes the methods using the fetched object
Fetches the object from the registry using lookup()
Binds the objects in the registry using rebind()
RMI Registry

Programs can access the registry thanks to *java.rmi.Naming* class.

13

Java RMI Overview





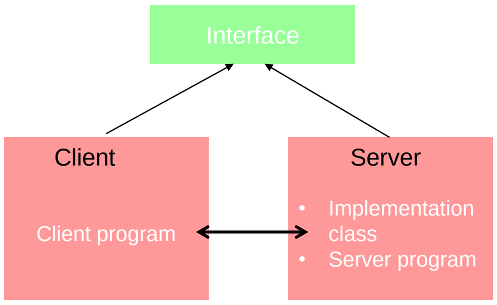
14

Internally, two servers

14

To write an RMI Java application, you would have to follow the steps −

- Define the (remote) interface
- Develop the implementation class (remote object)
- Develop the server program
- Develop the client program
- Compile the application
- Start registry
- Execute the applications



Simple program:
  Interface for remote object: Calculator.java
  implementation of object: CalculatorImpl.java
  server to run object: CalculatorServer.java

  client to access object: CalculatorClient.java

15

15

Example

Step 1: write interface Calculator.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {

  public long add(long a, long b) throws RemoteException;
  public long sub(long a, long b) throws RemoteException;
  public long mul(long a, long b) throws RemoteException;
  public long div(long a, long b) throws RemoteException;
}
```

Few rules to follow:
- interface must extend ***java.rmi.Remote*** interface –
  - this is a 'remote interface' now
- methods must throw ***java.rmi.RemoteException*** exception
  - since there is a chance of network issues during remote calls,

16

Step 2 **Developing the Implementation Class (Remote Object)**

To develop an implementation class –

- Implement the interface created in the previous step.
- Provide implementation to all the abstract methods of the remote interface.

- Implementation class should respect a few rules:
  - implement the interface (of course)
  - inherit from **the java.rmi.server.UnicastRemoteObject** class
  - have explicit constructor which throws the **java.rmi.RemoteException** exception

- This allows the object to be exported to client when it is instantiated by the server program (by constructor)

- Cannot do multiple inheritance? Then, don't inherit. Later, use **UnicastRemoteObject.export (obj)** to export the instance.

17

Example

Step 2: write remote object CalculatorImpl.java

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class CalculatorImpl extends UnicastRemoteObject
   implements Calculator {
   // Implementations must have an explicit constructor
   public CalculatorImpl() throws RemoteException {
     super();
   }
   public long add(long a, long b) throws RemoteException {
     return a + b;
   }
   public long sub(long a, long b) throws RemoteException {
     return a - b;
   }
   public long mul(long a, long b) throws RemoteException {
     return a * b;
   }
   public long div(long a, long b) throws RemoteException {
     return a / b;
   }
}
```

18

Example

Step 3: write server CalculatorServer.java

To develop a server program –

- Create a class from where you want invoke the remote object.
- Create a remote object by instantiating the implementation class
- Export the remote object
  - Automatically if object inherits **UnicastRemoteObject** class
  - If not, using the method **exportObject()** of the class named **UnicastRemoteObject** which belongs to the package **java.rmi.server**.
- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
  - `Registry registry = LocateRegistry.getRegistry()`
- Bind the remote object created to the registry using the **bind**() **rebind()** method of the class named **Registry**. To this method, pass a string representing the bind name and the object exported, as parameters.
  - `registry.bind("Hello", c);`

21

Example

Step 3: write server CalculatorServer.java

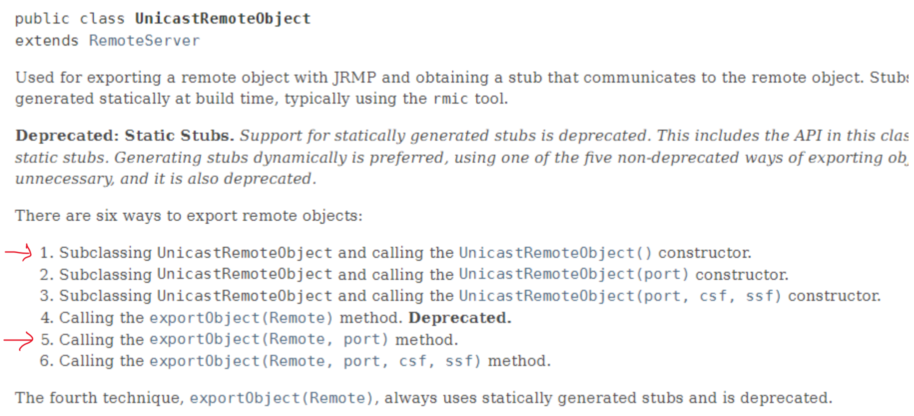
```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorServer {
  public static void main(String args[]) {
    try {
     CalculatorImpl cal = new CalculatorImpl(); // already exported

     Registry registry = LocateRegistry.getRegistry(1234);
                                     //or () default 1109
     registry.bind("Hello", cal);  // or rebind()

    }
    catch (Exception e) {
       System.out.println("Trouble: " + e);
    }
  }
}
```

Server program creates *CalculatorImpl* object.
Registers object to local RMI registry (‘rebind()’):
rebind(String name, Remote obj) associates a name to an object
names are in the form of a URL: rmi://<host_name>[:port]/<service_name>

22

if object could not inherit UnicastRemoteObject class

Due to multiple inheritance constraint or other reasons



```
public class UnicastRemoteObject
extends RemoteServer
```

Used for exporting a remote object with JRMP and obtaining a stub that communicates to the remote object. Stub
generated statically at build time, typically using the `rmic` tool.

Deprecated: Static Stubs. *Support for statically generated stubs is deprecated. This includes the API in this clas static stubs. Generating stubs dynamically is preferred, using one of the five non-deprecated ways of exporting ob unnecessary, and it is also deprecated.*

There are six ways to export remote objects:

1. Subclassing `UnicastRemoteObject` and calling the `UnicastRemoteObject()` constructor.
2. Subclassing `UnicastRemoteObject` and calling the `UnicastRemoteObject(port)` constructor.
3. Subclassing `UnicastRemoteObject` and calling the `UnicastRemoteObject(port, csf, ssf)` constructor.
4. Calling the `exportObject(Remote)` method. **Deprecated.**
5. Calling the `exportObject(Remote, port)` method.
6. Calling the `exportObject(Remote, port, csf, ssf)` method.

The fourth technique, `exportObject(Remote)`, always uses statically generated stubs and is deprecated.

23

23

Example

Step 2: write remote object CalculatorImpl.java

```
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;

public class CalculatorImpl ~~extends UnicastRemoteObject~~
   implements Calculator {
   // Implementations must have an explicit constructor
   public CalculatorImpl() throws RemoteException {
     super();
   }
   public long add(long a, long b) throws RemoteException {
     return a + b;
   }
   public long sub(long a, long b) throws RemoteException {
     return a - b;
   }
   public long mul(long a, long b) throws RemoteException {
     return a * b;
   }
   public long div(long a, long b) throws RemoteException {
     return a / b;
   }
}
```

24

Another way to export – if the object could not extend unicastRemoteObject

Step 3: write server CalculatorServer.java

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorServer {
   public static void main(String args[]) {
     try {
      CalculatorImpl cal = new CalculatorImpl(); // not exported!!!

      // Exporting the object of implementation class
      // (here we are exporting the remote object to the stub)
      Calculator stub = (Calculator) UnicastRemoteObject.exportObject(cal,0);


      Registry registry = LocateRegistry.getRegistry(1234);
                                  //or () default 1109
      registry.bind("Hello", stub);  // or rebind()

     }
     catch (Exception e) {
        System.out.println("Trouble: " + e);
   }
}
```

Client program

To develop a client program –

- Create a client class from where your intended to invoke the remote object.
- Get the RMI registry using the **getRegistry()** method of the **LocateRegistry** class which belongs to the package **java.rmi.registry**.
- Fetch the object from the registry using the method **lookup()** of the class **Registry** which belongs to the package **java.rmi.registry**.
- To this method, you need to pass a string value representing the bind name as a parameter. This will return you the remote object.
- The **lookup**() returns an object of type remote, down cast it to the type of the remote object.
- Finally invoke the required method using the obtained remote object.

Example

Step 4: write CalculatorClient.java

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class CalculatorClient {

   public static void main(String[] args) {
     try {

        // Getting the registry
        Registry registry = LocateRegistry.getRegistry(host, port);

       // Looking up the registry for the remote object
        Calculator stub = (Calculator) registry.lookup("Hello");

        System.out.println(stub.add(4,5));  // use as local object
        System.out.println(stub.sub(4,3));
     }
     catch (Exception e) {
             System.out.println(“Received Exception:”);
             System.out.println(e);
     }
   }
}
```

27

Example

Step 5: compile and run it!

Compile the application –

- Compile the Remote interface.
- Compile the implementation class.
- Compile the server program.

- Compile the client program.

Start the RMI registry: *rmiregistry* or *rmiregistry port*

- ***start rmiregistry*** [Window]
- ***rmigistry &*** [Unix]
- registry must have access to your classes
  - either start the registry in the same directory as the classes (easy) or, make sure directory is listed in $CLASSPATH variable, or just some flags e.g.,

    rmiregistry -J-Djava.class.path=\....

Run server:

- $ java CalculatorServer or IDE

Run client:

- $java CalculatorClient or IDE

```
9
1
$
```

29

Main topics (tentative)

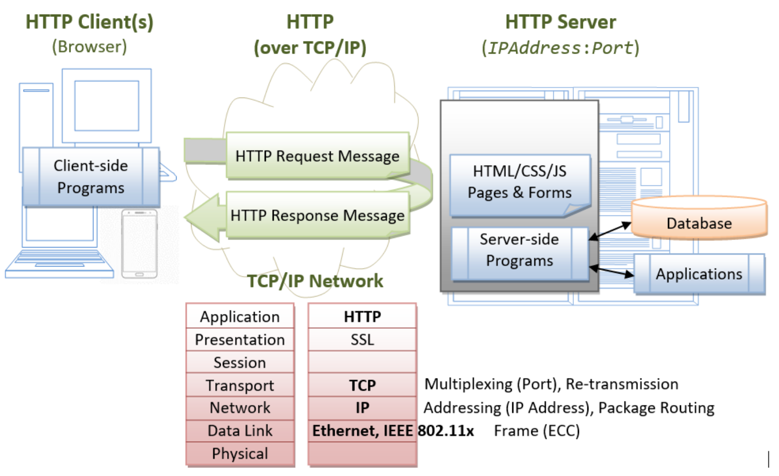
- Web App Architecture. Preliminary knowledge/Review
  - Client side: HTML CSS JavaScript
  - UML, design patterns, Java (cmd, thread, serialization),
- Client-Server, low level: socket programming
- Web applications (server side)
  - LAMP/CGI
  - Java Servlet
  - JSP, JavaBean, MVC pattern
  - SQL, Database access: JDBC. JPA
  - More: listener, filter, Ajax, JSON

- Web (RESTful) services, micro services
- Advanced topics (TBD): Deployments: Docker container, Node JS, React, Angular, Spring
- Other advanced topics (TBD) More design patterns, Performance, security

Server Side Development

Web Server

- A web server is a program running on the server that listens for incoming HTTP requests and services those requests as they come in.
- Once the web server receives a request, it will always return some kind of results to the web browser, even if its simply an error message saying that it couldn't process the request.
- By default, the role of a web server is to serve static pages using the **http** protocol
- Web servers can be made dynamic by adding additional processing capability to the server



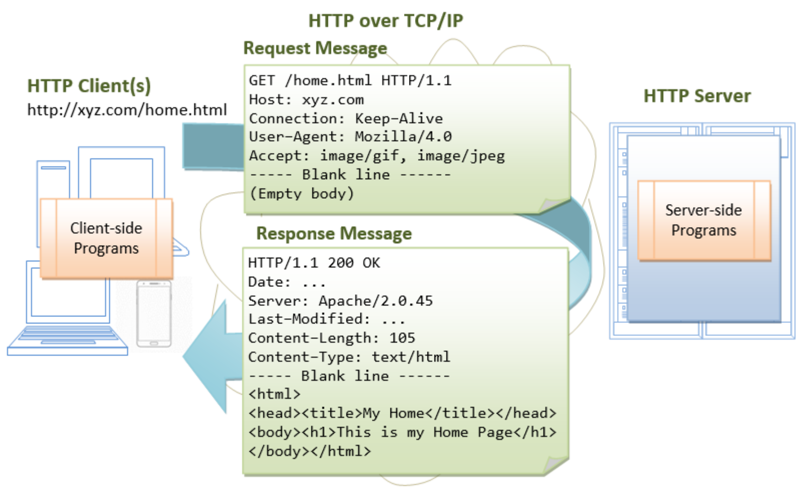
Application	**HTTP**	
Presentation	SSL	
Session		
Transport	**TCP**	Multiplexing (Port), Re-transmission
Network	**IP**	Addressing (IP Address), Package Routing
Data Link	**Ethernet, IEEE 802.11x**	Frame (ECC)
Physical		

There are many reasons an application might want to communicate via HTTP. E.g., compatibility with another browser-bases application. Or, need to gain access to a server through a firewall where direct socket or RMI connection are problematics.

HTTP is the lingua franca of the Net, rapidly become one of the most widely supported protocols in the world.

34

34



```
GET /home.html HTTP/1.1
Host: xyz.com
Connection: Keep-Alive
User-Agent: Mozilla/4.0
Accept: image/gif, image/jpeg
----- Blank line ------
(Empty body)
```

```
HTTP/1.1 200 OK
Date: ...
Server: Apache/2.0.45
Last-Modified: ...
Content-Length: 105
Content-Type: text/html
----- Blank line ------
<html>
<head><title>My Home</title></head>
<body><h1>This is my Home Page</h1>
</body></html>
```

There are many reasons an application might want to communicate via HTTP. E.g., compatibility with another browser-bases application. Or, need to gain access to a server through a firewall where direct socket or RMI connection are problematics.

HTTP is the lingua franca of the Net, rapidly become one of the most widely supported protocols in the world.

35

35

Server Side Development
Server Extensions

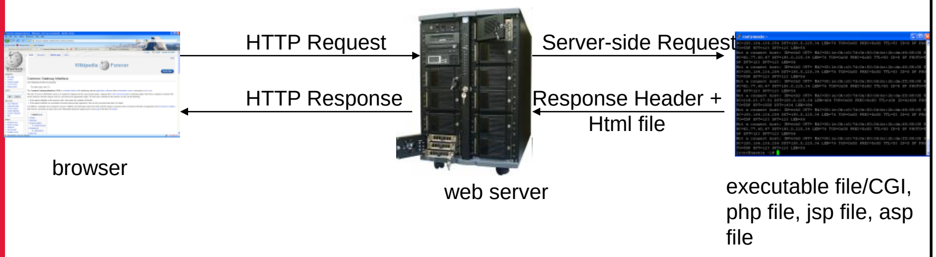
- Several different tools are available for extending the server capabilities
  - CGI scripting – LAMP
  - Active Server Pages (ASP)
  - VB .Net architecture
  - Java enterprise architecture (Servlet, JSP)
  - Node.js -- MEAN MERN
  - Ruby on Rails
  - …
- These tools process incoming requests from the user and generate custom/dynamic html pages

36

36

Server-side web programming

the HTTP Response consists of the output of an exernal program located on the server machine:



executable file/CGI, php file, jsp file, asp file

37

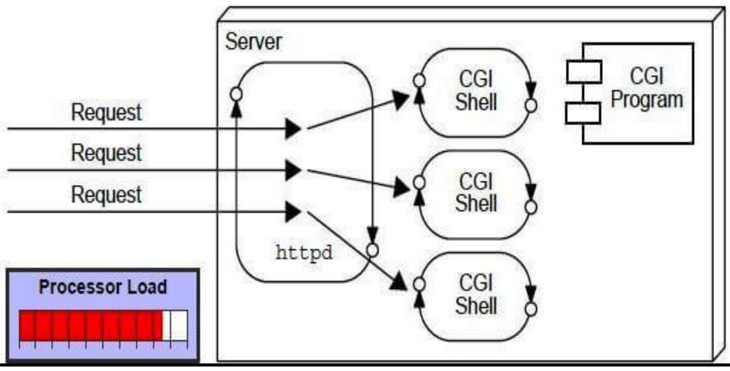
What is CGI?

- CGI: "Common Gateway Interface"
- a standard protocol for interfacing external application software with the web server
- developed in 1993 at NCSA (National Center for Supercomputing Applications)
- CGI 1.1 specified in RFC 3875, 2004
- allows an external executable file to respond to an HTTP Request from the browser
- CGI defines how information is passed from the web server to the executable program and how information is passed from this back to the server

38

CGI(Commmon Gateway Interface)

- CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request.
- For each request, it starts a new process.



39

A CGI example (in shell)

```
#!/bin/bash

echo Status: 200 OK
echo Content-Type: text/html
echo
echo

echo "<html><head></head>"
echo "<body>"
echo "Hello world."
echo "</body></html>"
```

41

Disadvantages of CGI

- because no special web-oriented language is used for writing CGI scripts (e.g. shell, perl, c/c++, python etc.) errors are highly probable and so, security vulnerabilities due to these problems
- usually a new process is created for each run of a CGI script; this increases the load on the server
- It uses platform dependent language e.g. C, C++, perl.
- Java Servlet addressed these issues (later)

42

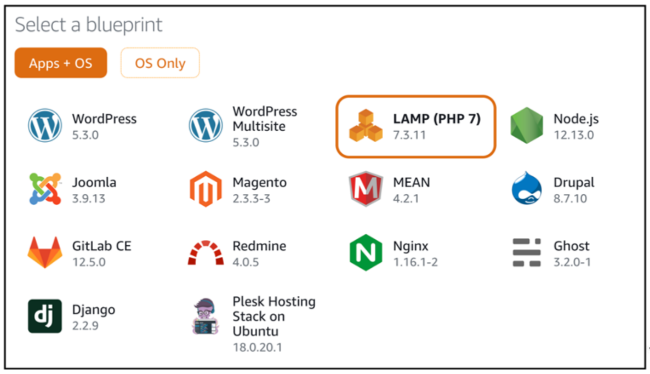
LAMP Stacks Roles

Linux	• Host operating System • Main processing / brainware
Apache	• Web server • http & https request and response
MySql	• Database Management Systems (DBMS)
PHP (or Perl, Python)	• Server side scripting

43

Popularity of LAMP

https://aws.amazon.com/id/getting-started/hands-on/launch-lamp-web-app/

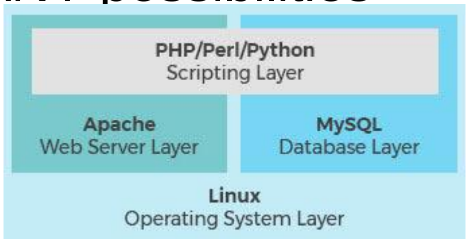

Select a blueprint

Apps + OS | OS Only

WordPress 5.3.0
WordPress Multisite 5.3.0
LAMP (PHP 7) 7.3.11
Node.js 12.13.0
Joomla 3.9.13
Magento 2.3.3-3
MEAN 4.2.1
Drupal 8.7.10
GitLab CE 12.5.0
Redmine 4.0.5
Nginx 1.16.1-2
Ghost 3.2.0-1
Django 2.2.9
Plesk Hosting Stack on Ubuntu 18.0.20.1

44

Dynamic Web- MANY possibilities



- **Linux**
  - OS system
- **Apache Server**
  - Open source server software that serves up HTML, media files etc
- **MySQL**
  - Open-source Relational Databases that supports structured queries and is free to use and install on web servers
- **PHP/Perl/Python**
  - Server-side scripting
  - Open source, simplicity and built-in links to MySQL database

*PHP handles main work on web server, MySQL manages data, and JavaScript looks after web page presentation. JavaScript can also talk with your PHP code on the web server whenever it needs to update something (either on the server or on the web page).

45

The Four Layers of a LAMP Stack

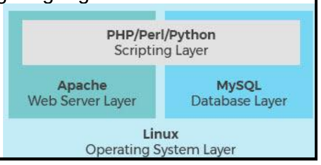
Linux based web servers consist of four software components. These components, arranged in layers supporting one another, make up the software stack. Websites and Web Applications run on top of this underlying stack. The common software components that make up a traditional LAMP stack are:

Linux: The operating system (OS) makes up our first layer. Linux sets the foundation for the stack model. All other layers run on top of this layer.

Apache: The second layer consists of web server software, typically Apache Web Server. This layer resides on top of the Linux layer. Web servers are responsible for translating from web browsers to their correct website.

MySQL: third layer is where databases live. MySQL stores details that can be queried by scripting to construct a website. MySQL usually sits on top of the Linux layer alongside Apache/layer 2. In high end configurations, MySQL can be off loaded to a separate host server.

PHP: Sitting on top of them all is our fourth and final layer. The scripting layer consists of PHP and/or other similar web programming languages. Websites and Web Applications run within this layer.

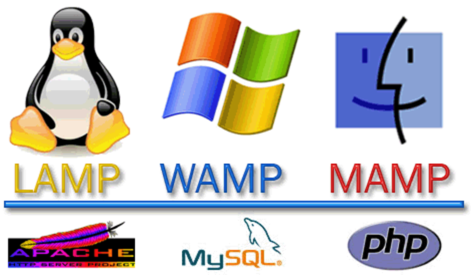


46

Dynamic Web –MANY possibilities

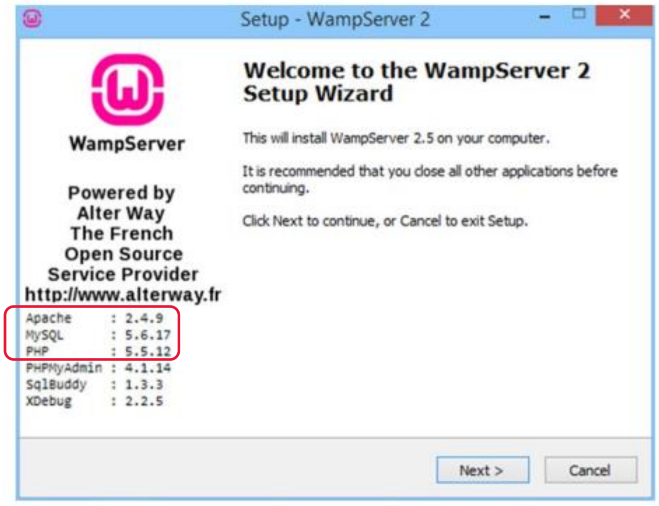
- BASIC POSSIBILITIES (here featuring open source /free solutions)

LAMP = Linux, Apache, MySQL, and PHP (or Perl, Python)
WAMP = Windows, Apache, MySQL, and PHP
MAMP = Mac, Apache, MySQL, and PHP
XAMPP= Windows/Max, Apache, MySQL, and PHP, Perl



47

An example



Welcome to the WampServer 2 Setup Wizard

This will install WampServer 2.5 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Powered by
Alter Way
The French
Open Source
Service Provider
http://www.alterway.fr

```
Apache      : 2.4.9
MySQL       : 5.6.17
PHP         : 5.5.12
PHPMyAdmin  : 4.1.14
SqlBuddy    : 1.3.3
XDebug      : 2.2.5
```

48

48

The Apache Software Foundation
http://www.apache.org/

Apache HTTP Server

- Apache has been the most popular web server since April 1996 when it passed NCSA (National Center for Supercomputing Applications).
- In April 1996, Apache stood at 29% (and IIS at 1.6%) of the web server market:
- Previous market leader 2000 – 2015 → Apache stands at 59% (and IIS at 31%)
- Recent market leader → nginx, apache, Microsoft IIS

YORK UNIVERSITÉ UNIVERSITY

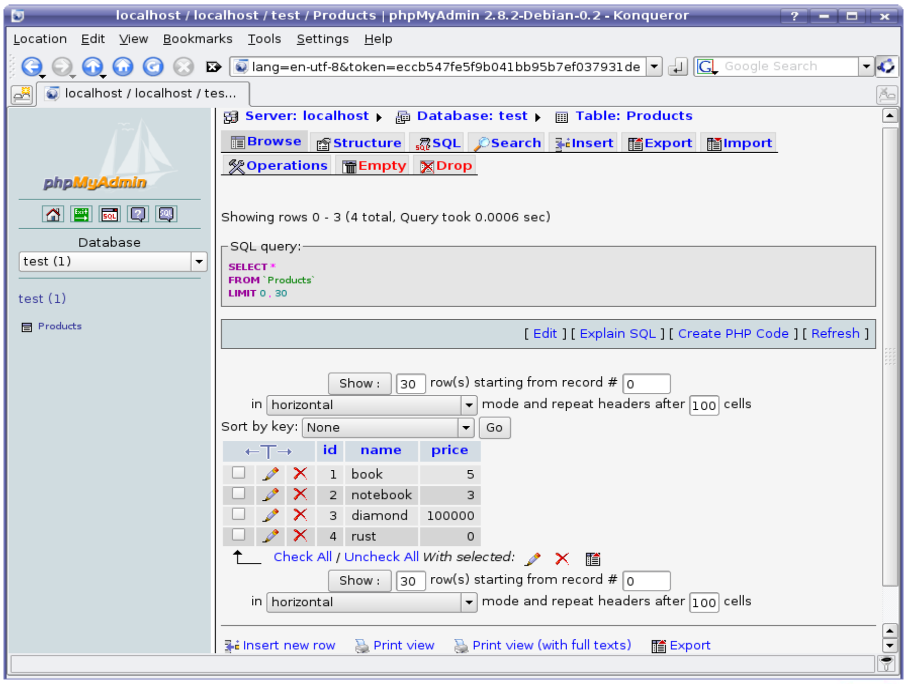
49

MySQL

- MySQL is a popular, scalable, free, open-source database management system (DBMS).
- It runs on Windows, Linux, and many other operating systems.
- MySQL AB, the Swedish company that owns the codebase states that MySQL is used in 11 million installations in the world.
- Can will use phpMyAdmin to access MySQL through Apache.

YORK UNIVERSITÉ UNIVERSITY

51


localhost / localhost / test / Products | phpMyAdmin 2.8.2-Debian-0.2 - Konqueror
Location Edit View Bookmarks Tools Settings Help
lang=en-utf-8&token=eccb547fe5f9b041bb95b7ef037931de
Google Search
localhost / localhost / tes...
phpMyAdmin
Database
test (1)
test (1)
Products
Server: localhost ▸ Database: test ▸ Table: Products
Browse Structure SQL Search Insert Export Import
Operations Empty Drop
[Edit] [Explain SQL] [Create PHP Code] [Refresh]
Check All / Uncheck All With selected:
Insert new row Print view Print view (with full texts) Export

Showing rows 0 - 3 (4 total, Query took 0.0006 sec)

SQL query:

```
SELECT *
FROM `Products`
LIMIT 0 , 30
```

Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Sort by key: None Go

id	name	price
1	book	5
2	notebook	3
3	diamond	100000
4	rust	0

Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells

52

PHP

PHP is popular, simple and lightweight server-side scripting language used to create dynamic web pages.

History:

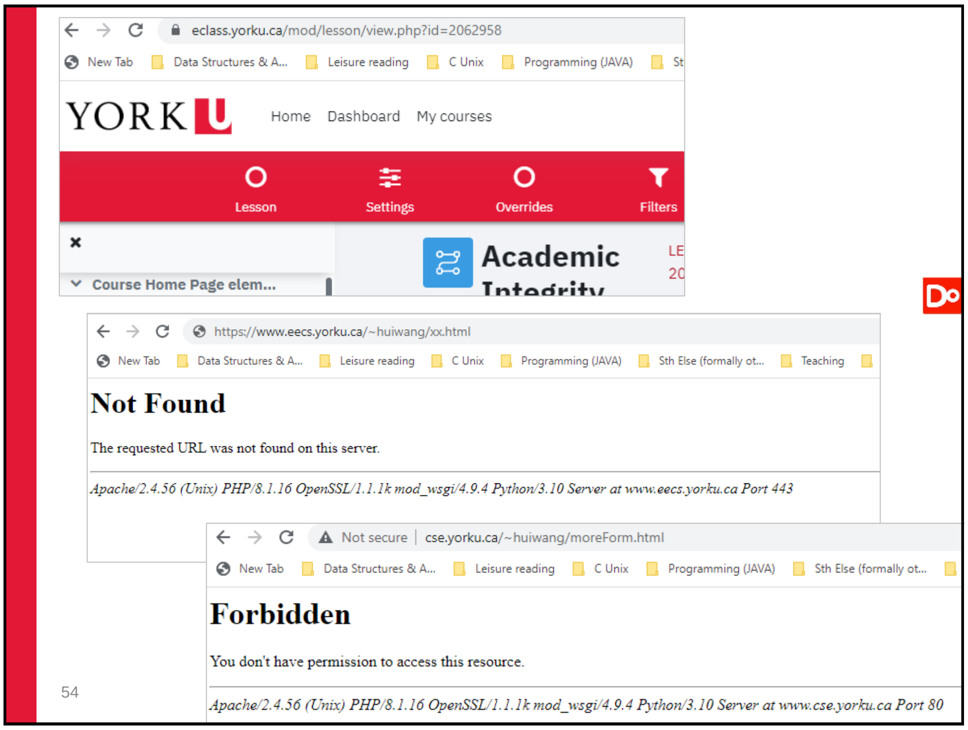
- 1995: PHP 2, “Personal Home Page Tools”
- 1997: PHP 3, “PHP: Hypertext Processor”
- 2000: PHP 4
- 2004: PHP 5

Some sites that use PHP:

- Wikipedia, Yahoo, Digg, Facebook

YORK UNIVERSITÉ UNIVERSITY

53


eclass.yorku.ca/mod/lesson/view.php?id=2062958

https://www.eecs.yorku.ca/~huiwang/xx.html

Not Found

The requested URL was not found on this server.

Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_wsgi/4.9.4 Python/3.10 Server at www.eecs.yorku.ca Port 443

Not secure | cse.yorku.ca/~huiwang/moreForm.html

Forbidden

You don't have permission to access this resource.

Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_wsgi/4.9.4 Python/3.10 Server at www.cse.yorku.ca Port 80

54

54

Server-Side Scripting

On the server, PHP code produces HTML.
Client only receives HTML.
Client can't view the original PHP code.
Compare with HTML:

```
<b>This is readable on the page</b>
<!--  This is visible if you view source -->
<?php   // PHP source is not visible  ?>
```

55

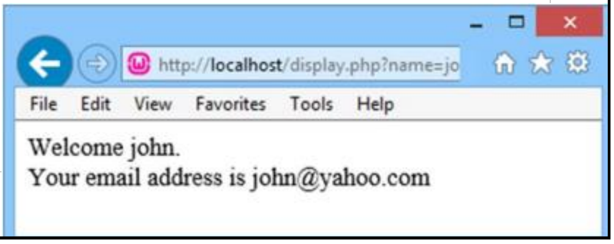
helloworld.php

```
<html>
<head>
<title>PHP: Hello World</title>
</head>
<body>
<h1>PHP Says:</h1>
<?php
echo "Hello World!"
?>
</body>
</html>
```

56

display.php

```
<html>
<head>
<title>PHP: Hello World</title>
</head>
<body>
<h1>PHP Says:</h1>
Welcome <?php echo $_GET["name"]; ?> <br>
Your email address is
<?php echo  $_GET["email_add"];?>
</body>
</html>
```

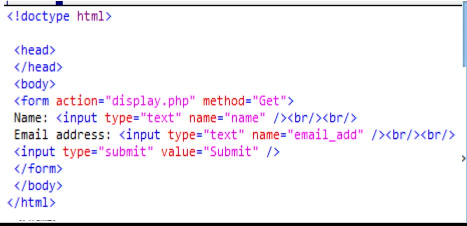


Welcome john.
Your email address is john@yahoo.com

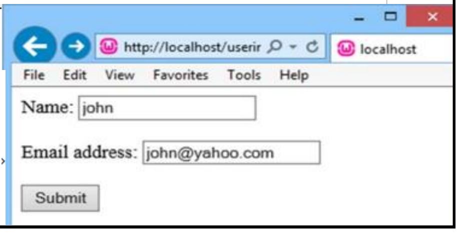
57

display.php

```
<html>
<head>
<title>PHP: Hello World</title>
</head>
<body>
<h1>PHP Says:</h1>
Welcome <?php echo $_GET["name"]; ?> <br>
Your email address is
<?php echo  $_GET["email_add"];?>
</body>
</html>
```



```
<!doctype html>

 <head>
 </head>
 <body>
 <form action="display.php" method="Get">
 Name: <input type="text" name="name" /><br/><br/>
 Email address: <input type="text" name="email_add" /><br/><br/>
 <input type="submit" value="Submit" />
 </form>
 </body>
</html>
```



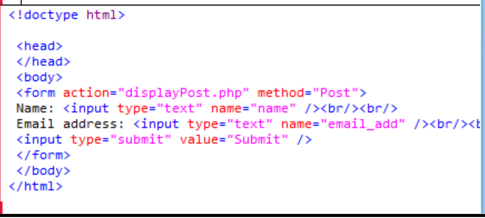
Name: john

Email address: john@yahoo.com

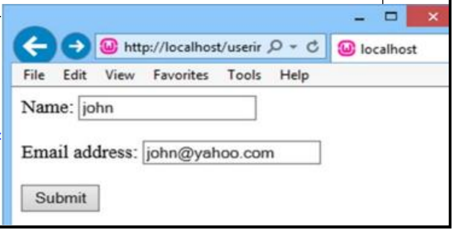
58

display.php

```
<html>
<head>
<title>PHP: Hello World</title>
</head>
<body>
<h1>PHP Says:</h1>
Welcome <?php echo $_POST["name"]; ?> <br>
Your email address is
<?php echo  $_POST["email_add"];?>
</body>
</html>
```



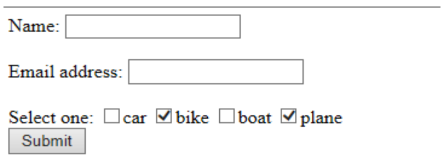
```
<!doctype html>

 <head>
 </head>
 <body>
 <form action="displayPost.php" method="Post">
 Name: <input type="text" name="name" /><br/><br/>
 Email address: <input type="text" name="email_add" /><br/><b
 <input type="submit" value="Submit" />
 </form>
 </body>
</html>
```



Name: john

Email address: john@yahoo.com

59



Name:

Email address:

Select one: ☐ car ☑ bike ☐ boat ☑ plane

Submit

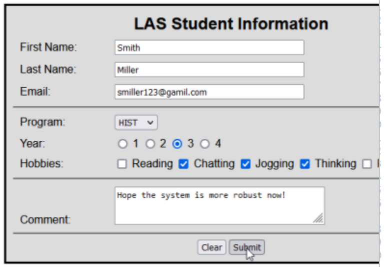
```
<!doctype html>

 <head>
 </head>
 <body>
 <form action="http://www.cse.yorku.ca/~huiwang/displayPostMore.php" method="Post">
 Name: <input type="text" name="name" /><br/><br/>
 Email address: <input type="text" name="email_add" /><br/><br/>


Select one:
<input type="checkbox" value='car' name="ice[]" >car
<input type="checkbox" value='bike' name="ice[]" >bike
<input type="checkbox" value='boat' name="ice[]" >boat
<input type="checkbox" value='planexxx' name="ice[]" >plane
 <br>
  <input type="submit" value="Submit" />

 </form>
 </body>
</html>
```

```
….
<br>
<?php  $a=$_POST["ice"] ;  echo count($a)." ". $a[0]."<br>";

 foreach($_POST['ice'] as $value){
       echo  $value."  ";
 } ?>
</body>
</html>
```

60

60

Lab3 in XMAPP, WMAP, MMAP



LAS Student Information

First Name:	Smith
Last Name:	Miller
Email:	smiller123@gamil.com
Program:	HIST
Year:	○ 1 ○ 2 ● 3 ○ 4
Hobbies:	☐ Reading ☑ Chatting ☑ Jogging ☑ Thinking ☐
Comment:	Hope the system is more robust now!

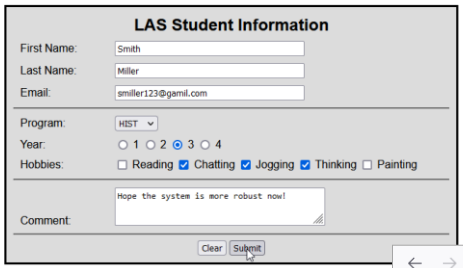
Clear Submit

```
  First Name:
    <input type="text"  name="firstName">
  </p>

    <!-- Last Name input -->
  Last Name:
  <input  type="text" name="lastName">

  Email:
  <input  type="text" name="email">

  <hr>

  Program:
  <select id="provinceList" name="program">
    <option value="CHEM" >CHEM</option>
    <option value="CIVL" >CIVL</option>
    <option value="EECS" selected>EECS</option>
    <option value="ECON" >ECON</option>
    <option value="HIST" >HIST</option>
    <option value="MATH"> MATH</option>
  </select>

 Year:
 <input type="radio" name="year" value="1" checked="checked" /> 1
 <input type="radio" name="year" value="2" /> 2
 <input type="radio" name="year" value="3" /> 3
 <input type="radio" name="year" value="4" /> 4

 Hobbies:
 <input type="checkbox" name="hobby" value="Reading"/> Reading
 <input type="checkbox" name="hobby" value="Chatting"  /> Chatting
 <input type="checkbox" name="hobby" value="Jogging"/>  Jogging
 <input type="checkbox" name="hobby" value="Thinking" checked="checked"/>  Thinking
 <input type="checkbox" name="hobby" value="Painting"/>  Painting

<hr>

  <input type="reset" value="Clear"  >
  <input type="submit" value="Submit" >

</form>
```

61

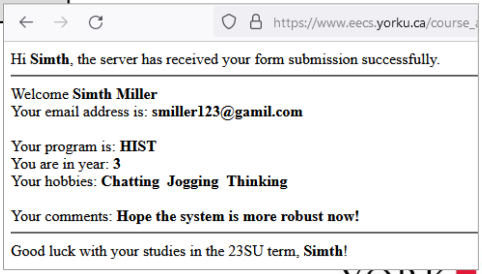
61

Lab3 in XMAPP, WMAP, MMAP



LAS Student Information

First Name:	Smith
Last Name:	Miller
Email:	smiller123@gamil.com
Program:	HIST
Year:	1 2 [✓ 3] 4
Hobbies:	Reading [✓ Chatting] [✓ Jogging] [✓ Thinking] Painting
Comment:	Hope the system is more robust now!



Hi **Simth**, the server has received your form submission successfully.

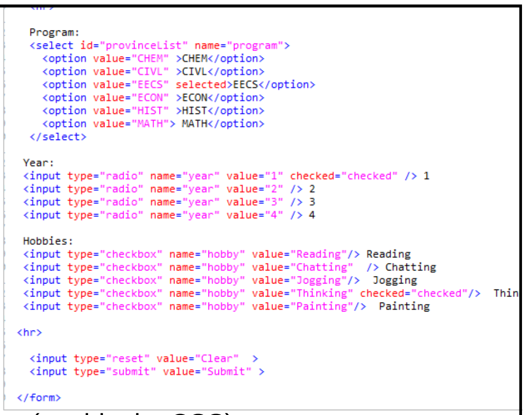
Welcome **Simth Miller**
Your email address is: **smiller123@gamil.com**

Your program is: **HIST**
You are in year: **3**
Your hobbies: **Chatting Jogging Thinking**

Your comments: **Hope the system is more robust now!**

Good luck with your studies in the 23SU term, **Simth**!

Send by form – recall and summary



```
 Program:
 <select id="provinceList" name="program">
   <option value="CHEM" >CHEM</option>
   <option value="CIVL" >CIVL</option>
   <option value="EECS" selected>EECS</option>
   <option value="ECON" >ECON</option>
   <option value="HIST" >HIST</option>
   <option value="MATH"> MATH</option>
 </select>

Year:
<input type="radio" name="year" value="1" checked="checked" /> 1
<input type="radio" name="year" value="2" /> 2
<input type="radio" name="year" value="3" /> 3
<input type="radio" name="year" value="4" /> 4

Hobbies:
<input type="checkbox" name="hobby" value="Reading"/> Reading
<input type="checkbox" name="hobby" value="Chatting"  /> Chatting
<input type="checkbox" name="hobby" value="Jogging"/>  Jogging
<input type="checkbox" name="hobby" value="Thinking" checked="checked"/>  Thin
<input type="checkbox" name="hobby" value="Painting"/>  Painting

<hr>

  <input type="reset" value="Clear"  >
  <input type="submit" value="Submit" >

</form>
```

- Can do GET or POST
  - If not specified, use GET
- **name** attribute is used by server (not id – by CSS)
  - Checkbox, radio button, drop-down uses same name
- **value**:
  - Textbox/arear entered content.
    - can have default value **`value="xxx"`** entered will overwrite
  - checkbox, radio button: **`value="xxx"`**
- If no action, goes to same page
- Can have hidden field

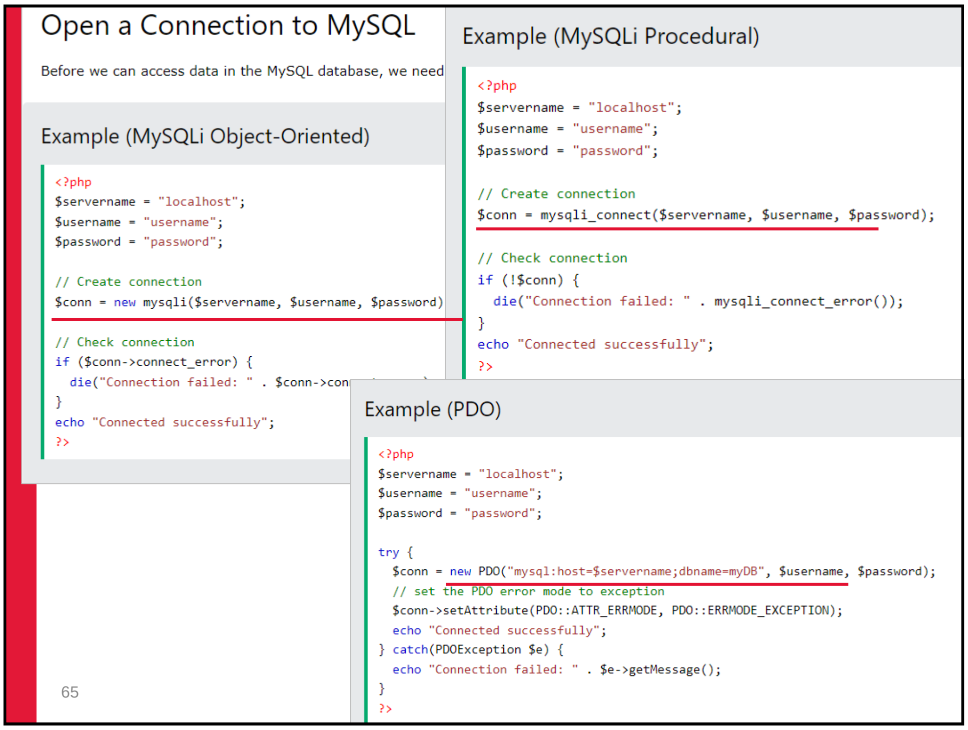
`<input type="hidden" name="custId" value="3487">`

HTTP
GET and POST

- GET and POST allow information to be sent back to the web server from a browser
  - e.g. when you click on the “submit” button of a form the data in the form is send back to the server, as "name=value" pairs.
- Choosing GET as the "method" will append all of the data to the URL and it will show up in the URL bar of your browser.
  - The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.
- A POST sends the information through a socket back to the webserver and it won't show up in the URL bar.
  - This allows a lot more information to be sent to the server
  - The data sent back is not restricted to textual data and it is possible to send files and binary data such as serialized Java objects.

64 of 99

64



Open a Connection to MySQL

Before we can access data in the MySQL database, we need

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password)

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->con
}
echo "Connected successfully";
?>
```

Example (MySQLi Procedural)

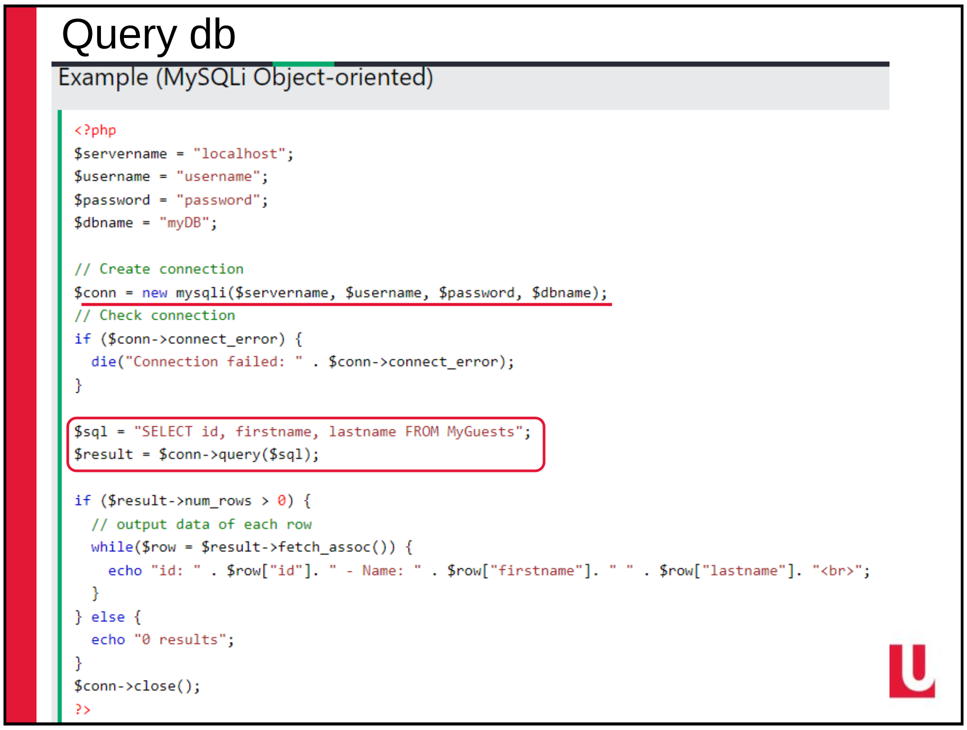
```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  echo "Connected successfully";
} catch(PDOException $e) {
  echo "Connection failed: " . $e->getMessage();
}
?>
```

65

65

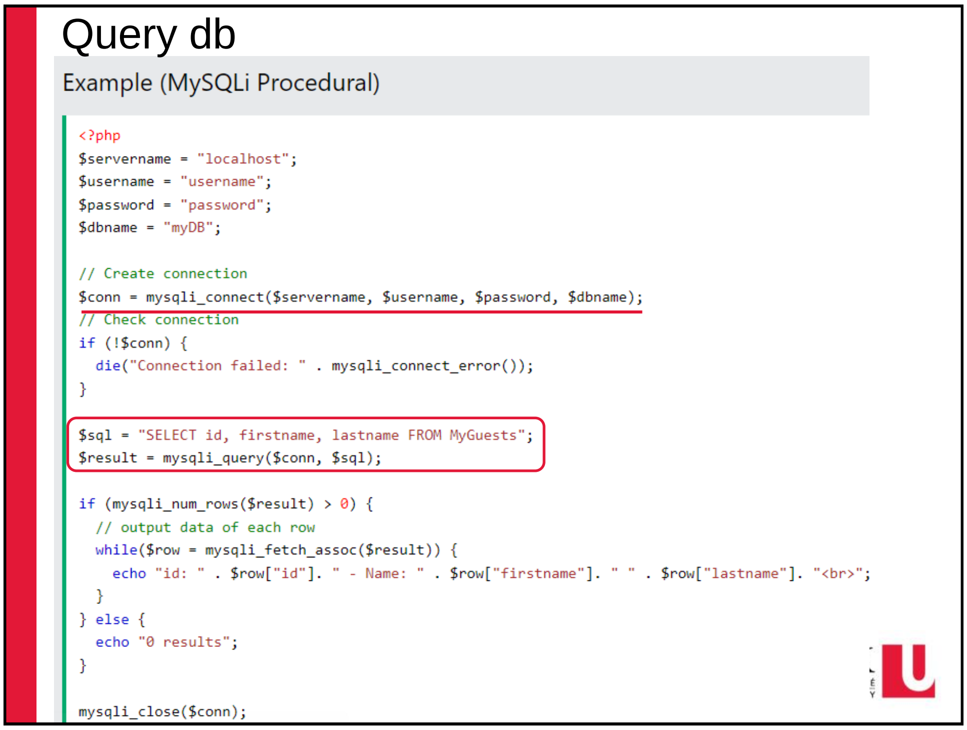


Query db

Example (MySQLi Object-oriented)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
$conn->close();
?>
```

66



Query db

Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}

mysqli_close($conn);
```

67

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
  $stmt->execute();

  // set the resulting array to associative
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
    echo $v;
  }
} catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

68

68

Lab1 database connection

```
<?php
try{
    $myPDO = new PDO('sqlite:./Courses.db');
    echo "Opened database successfully<br><br>\n\n";
}
catch(PDOException $e){
    echo "Error: ";
}

 $result = $myPDO->query("SELECT * from Course where code = '".$_GET["course"]."'" );

 foreach($result as $row)
    {

      $res = $res . "<b>Title</b>: ". $row['title'] . " <br> ";
      $res = $res . "<b>Time</b>: ". $row['time'] ."<br>";
      $res = $res . "<b>Location</b>: ". $row['location'] ."<br> ";


    }
    if($result == "") echo "Sorry, your searched course <b>".$_GET["course"]. "</b> does not exist in databa
    else echo "Your searched course <b>". $_GET["course"] . "</b><br>".$res;

?>



</body>
</html>
```

69

69

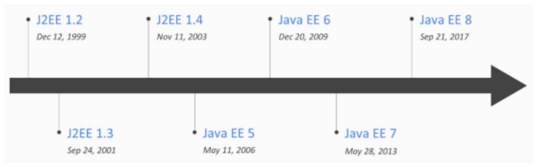
Server Side Development

Server Extensions

- Several different tools are available for extending the server capabilities
  - CGI scripting – LAMP
  - Active Server Pages (ASP)
  - VB .Net architecture
  - Java enterprise architecture (Servlet, JSP)
  - Node.js -- MEAN MERN
  - Ruby on Rails???
  - …
- These tools process incoming requests from the user and generate custom/dynamic html pages

70

70



Java enterprise platform history

Platform version	Released	Specification	Java SE Support	Important Changes
Jakarta EE 10	2022-09-13[9]	10	Java SE 17 Java SE 11	Removal of deprecated items in Servlet, Faces, CDI and EJB (Entity Beans and Embeddable Container). CDI-Build Time.
Jakarta EE 9.1	2021-05-25[10]	9.1	Java SE 11 Java SE 8	JDK 11 support
Jakarta EE 9	2020-12-08[11]	9	Java SE 8	API namespace move from `javax` to `jakarta`
Jakarta EE 8	2019-09-10[12]	8	Java SE 8	Full compatibility with Java EE 8
Java EE 8	2017-08-31	JSR 366	Java SE 8	HTTP/2 and CDI based Security
Java EE 7	2013-05-28	JSR 342	Java SE 7	WebSocket, JSON and HTML5 support
Java EE 6	2009-12-10	JSR 316	Java SE 6	CDI managed Beans and REST JAXRS
Java EE 5	2006-05-11	JSR 244	Java SE 5	Java annotations
J2EE 1.4	2003-11-11	JSR 151	J2SE 1.4	WS-I interoperable web services[13] EL in JSP
J2EE 1.3	2001-09-24	JSR 58	J2SE 1.3	Java connector architecture[14]
J2EE 1.2	1999-12-17	1.2	J2SE 1.2	Initial specification release Servlet, JSP, EJB

71