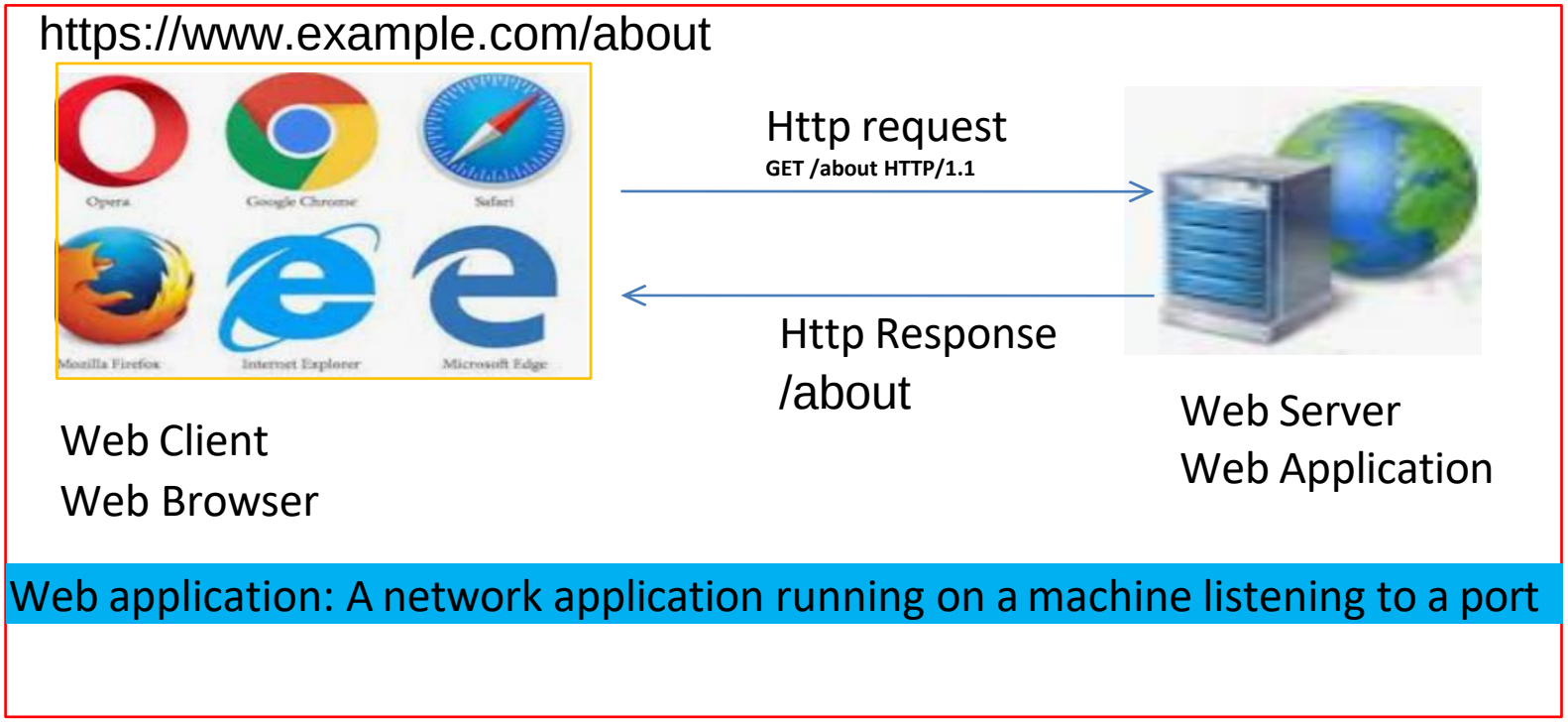
# CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

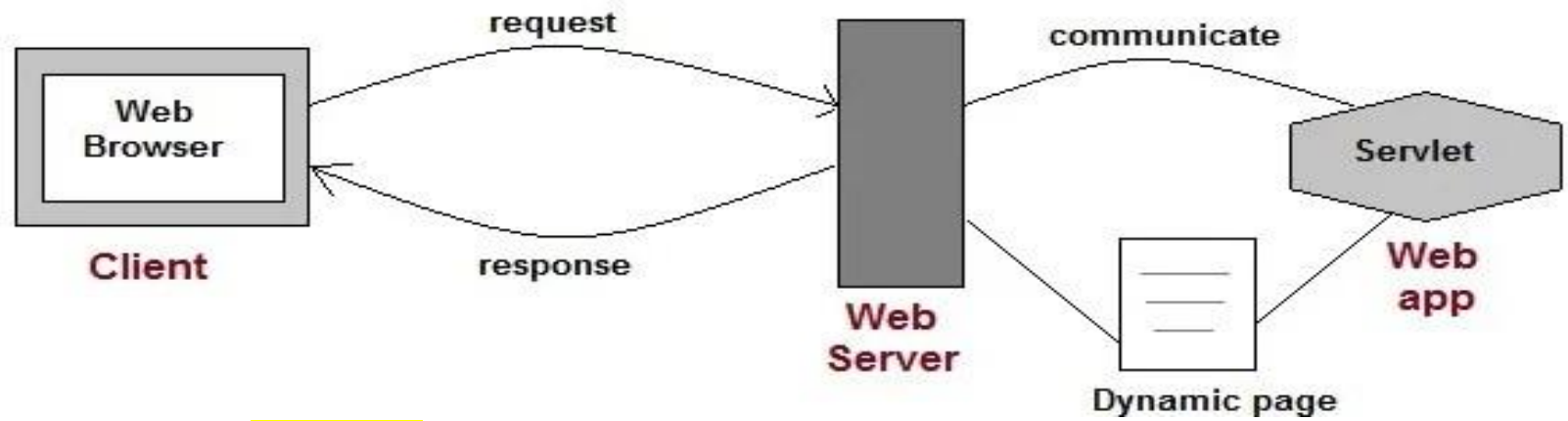
# Web Client-Web Server Communication




**Web browser**: Software used to surf the internet and access websites, like Google Chrome, Firefox, or Safari

**Web server** - System that hosts and delivers the web content.

**HTTP protocol** - the standardized language that both the client (web browser) and server can understand.




# Website vs. Web Application

A **website** is a collection of static files that doesn't require user interaction, typically include HTML pages, images, CSS, and JavaScript files.

Examples: www.annauniv.edu. A personal blog, a news website where content doesn't change dynamically for each user.

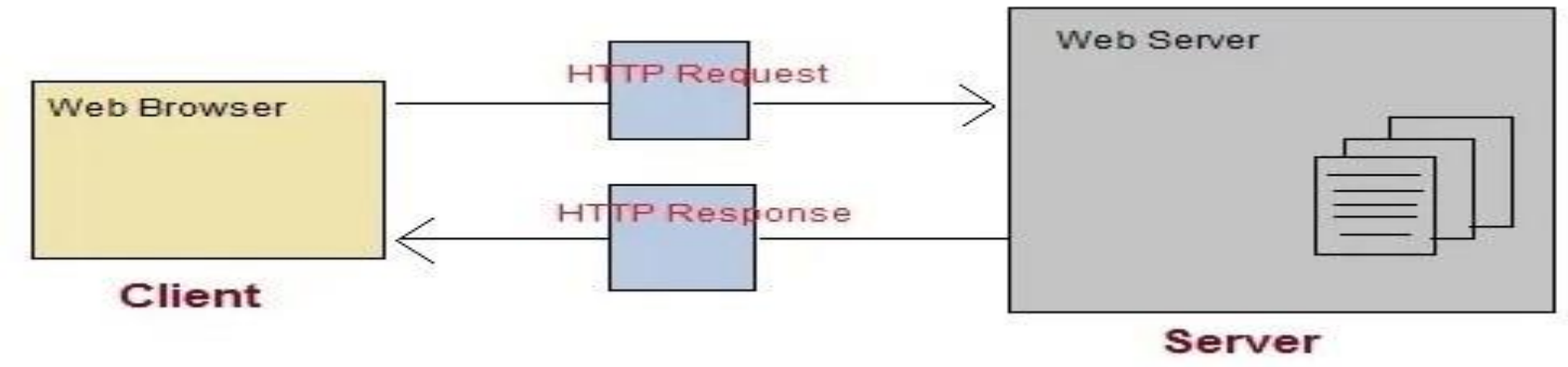
A web application is more dynamic and interactive., typically involving server-side processing and database interactions.

Examples: Google, Facebook, Twitter, Gmail—applications where user inputs trigger changes on the server and are reflected dynamically in the interface.

Servlet Technology is a technology for creating dynamic web applications using Java.

# HTTP

- HTTP is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
- HTTP is a **stateless protocol** i.e HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.



**HTTP Request**
[method] [URL] [version]
[headers]
[body]

An HTTP request is a message sent by a client (such as a web browser, mobile app, or API client) to a server to request a specific resource or perform a particular operation. It is the foundation of communication between clients and servers on the World Wide Web.

REQUEST!

Components of an HTTP Request:
Request Line: Contains:

HTTP method (e.g., GET, POST).
URL or endpoint of the resource (e.g., /index.html).
HTTP version (HTTP/1.1).
Headers: Provide metadata about the request (e.g., Content-Type, Authorization, User-Agent).

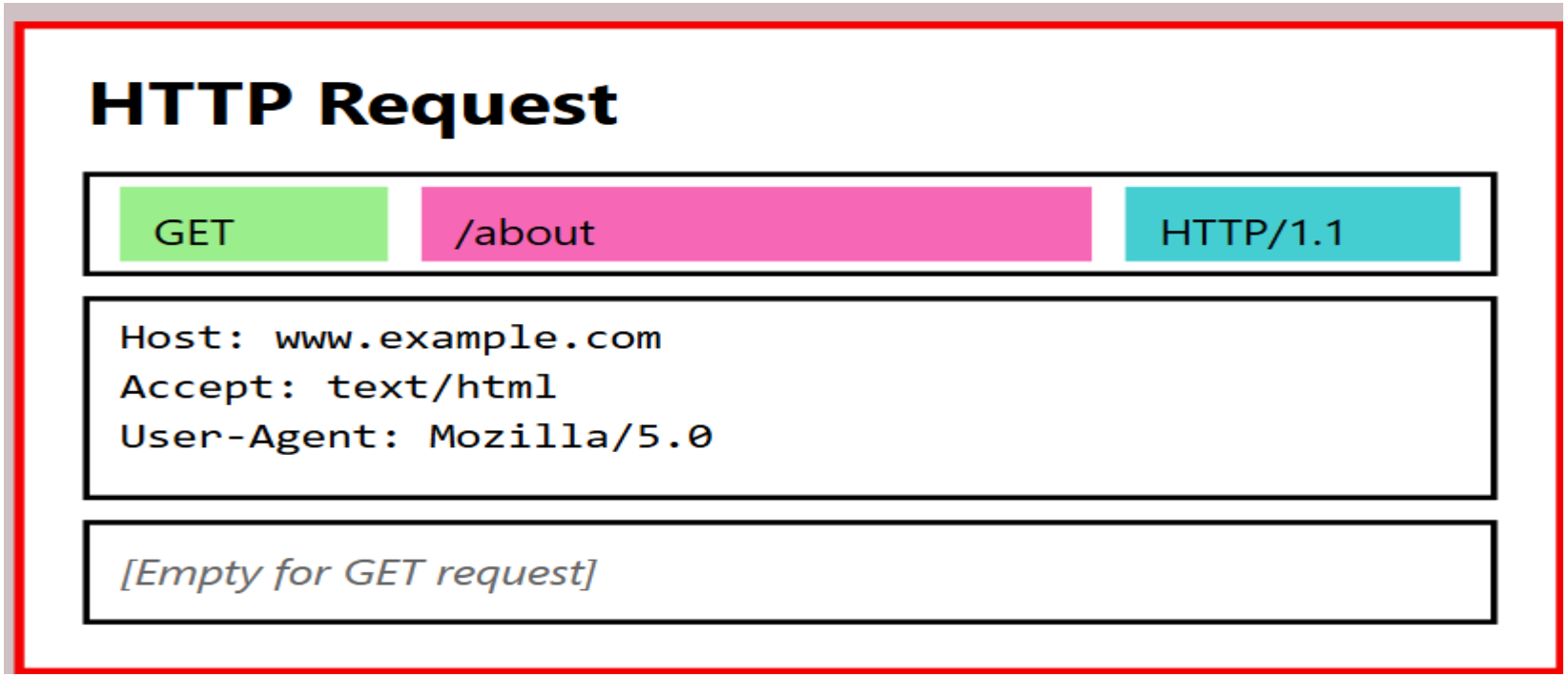
Body: Optional part where data is sent to the server (used in POST, PUT, etc.).

## HTTP REQUEST METHODS

| Method | Description |
|---|---|
| GET | Retrieve a resource i.e viewing a page |
| PUT | Update a resource |
| DELETE | Remove a resource |
| POST | Create a resource i.e user registration, submitting forms |
| HEAD | Retrieve just the response headers for a resource,nobody |

GET https://www.example.com/ HTTP/1.1
Host: www.example.com
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) Chrome/16.0.912.75 Safari/535.7
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8
Referer: http://www.google.com/url?q=www.example.com
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3

https://www.example.com/about




Body section is empty because GET requests typically don't include a body

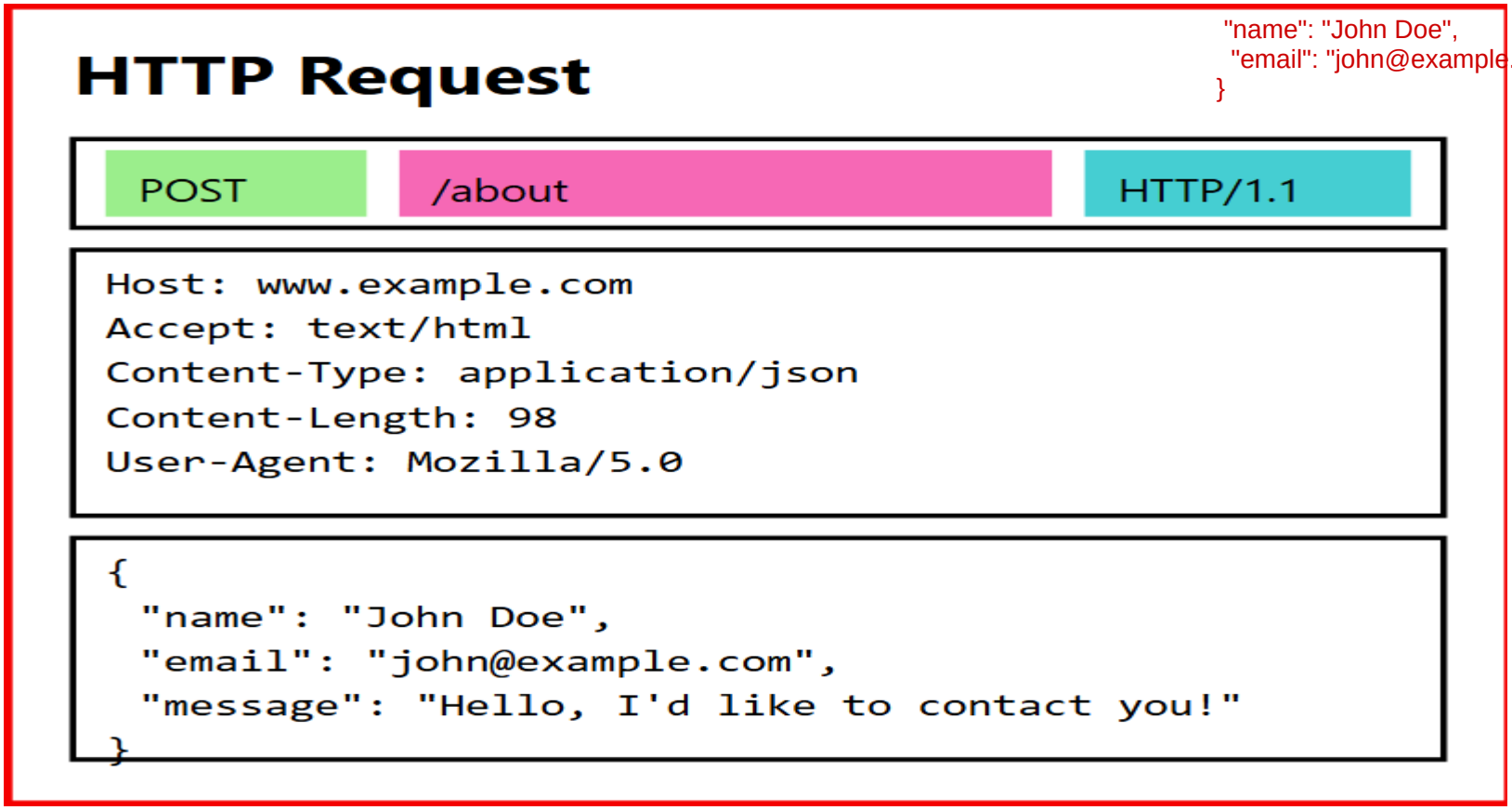
GET

Retrieves data from the server.
Example: Fetching a webpage or API data.
No body is sent in the request.
Example:
http
Copy code
GET /users HTTP/1.1
Host: example.com

https://www.example.com/about

POST /users HTTP/1.1
Host: example.com
Content-Type: application/json

{
 "name": "John Doe",
 "email": "john@example.com"
}




Content-Type: application/json=>indicates sending JSON data)
Content-Length: 98=> tells server how many bytes are in the body

POST

Sends data to the server to create a resource.
Example: Submitting a form or uploading a file.
The data is sent in the body of the request.

# Difference between GET and POST requests

| GET Request | POST Request |
| --- | --- |
| Data is sent in header to the server | Data is sent in the request body |
| Get request can send only limited amount of data | Large amount of data can be sent. |
| Get request is not secured because data is exposed in URL | Post request is secured because data is not exposed in URL. |
| Get request can be bookmarked and is more efficient. | Post request cannot be bookmarked. |

# Forms and GET Requests

```
<form action="/search" method="GET">
        <label for="term">Search:label>
        <input id="term" name="term" type="text" />
        <input type="submit" value="Sign up!"/>
form>
```

```
GET http://localhost:1060/search?term=java HTTP/1.1
Host: localhost:1060
```

```
<form action="/account/create" method="POST">
    <label for="firstName">First namelabel>
    <input id="firstName" name="firstName" type="text" />

    <label for="lastName">Last namelabel>
    <input id="lastName" name="lastName" type="text" />

    <input type="submit" value="Sign up!"/>
form>
```

```
POST http://localhost:1060/account/create HTTP/1.1
Host: localhost:1060

firstName=Scott&lastName=Allen
```

POST parameters go into the body of the HTTP message.
GET parameters go into the query string.

## GET Request (Less Secure)

https://example.com/login?username=john&password=secret123&token=abc

X Data visible in:
- Browser history
- Server logs
- Proxy server logs

## POST Request (More Secure)

https://example.com/login

Request Body (encrypted in transit):
{"username": "john", "password": "secret123", "token": "abc"}

**The Response**

An HTTP response has a similar structure to an HTTP request. The sections of a response are:

[version] [status] [reason] [method][url][version]

[headers]

header and body same!!

[body]

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Sat, 14 Jan 2012 04:00:08 GMT
Connection: close
Content-Length: 17151

<html>
<head>
    <title>.Net related Articles, Code and Resourcestitle>
head>
<body>
 ... content ...
body>
html>
```

**Response Status Codes**

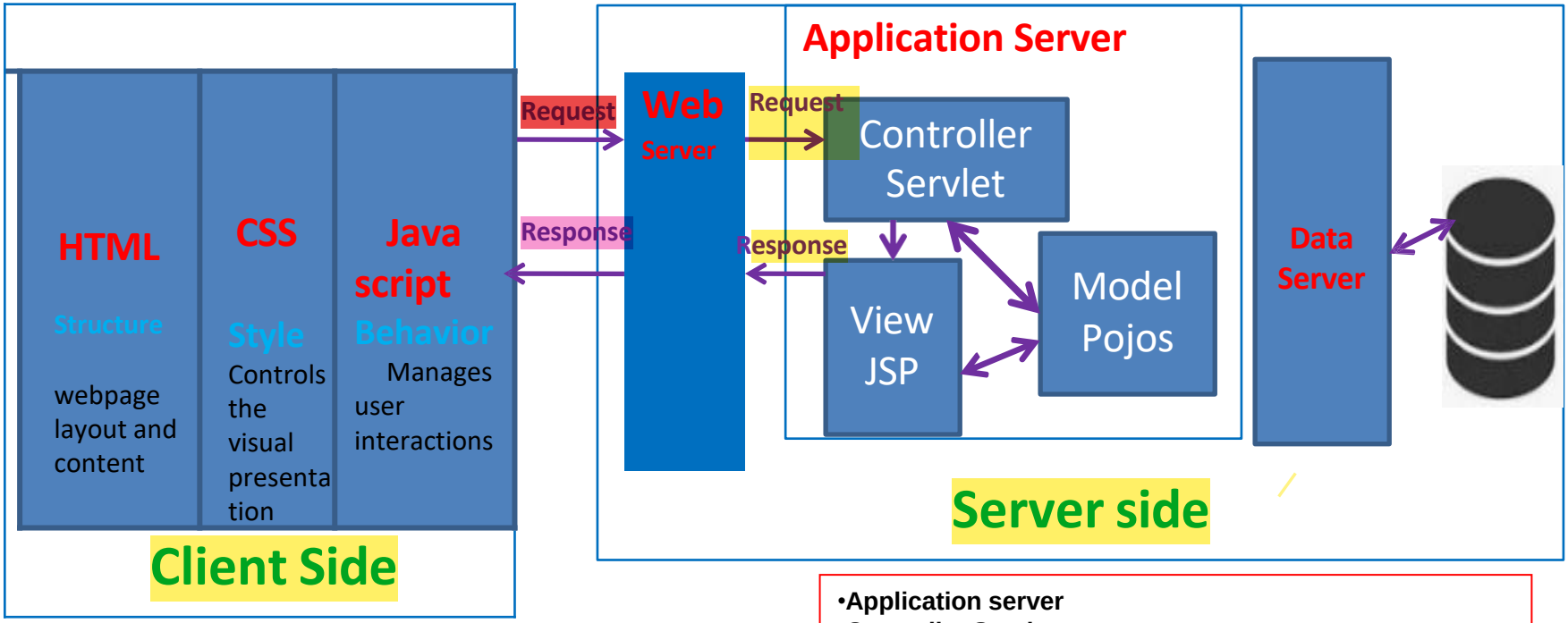
Code Reason

200 OK

**Description:** A 200 code in the response means everything worked!

| | | |
|---|---|---|
| 400 | Bad Request | The server could not understand the request. The request probably used incorrect syntax. |
| 401 | Unauthorized | The client was not authorized to access the resource and might need to authenticate. More on 401s and security in a later article. |
| 403 | Forbidden | The server refused access to the resource for an unspecified reason. |
| 404 | Not Found | A popular code meaning the resource was not found on the server. |
| 500 | Internal Server Error | The server encountered an error in processing the request. Commonly happens because of programming errors in a web application. |
| 503 | Service Unavailable | The server will currently not service the request. This status code can appear when a server is throttling requests because it is under heavy load. |

# Web Application Architecture Overview




int var=request.getParameter(“textName”);

- **Web Server**
- Receives initial HTTP requests from client
- Routes requests to appropriate application components
- Handles request/response communication

- **Application server**
- **Controller Servlet**
- Processes incoming requests
- Example code shown: int var=request.getParameter("textName");
- Handles form data and request parameters
- **View (JSP)**
- Generates dynamic HTML content
- Renders data for client presentation
- Separates presentation logic from business logic
- **Model POJOs**
- Contains business logic
- Interacts with database

A Servlet is a Java program that runs on a web server or application server and handles HTTP requests and responses. It is a key component of Java-based web applications and is used to generate dynamic web content.

# What Is a Servlet?

Types of Servlets
GenericServlet: A protocol-independent base class.
HttpServlet: A subclass of GenericServlet designed to handle HTTP-specific requests.

- A **servlet** is a Java class that runs on a web server and is designed to handle client requests.
- Servlet is a java program that runs inside JVM on the web server.
- Servlets operate within a **Servlet container** (also known as a **Servlet engine**), which is part of a web server or application server, such as Apache Tomcat or Jetty.
- It is used for developing dynamic web applications.
- For such applications, Java Servlet technology defines HTTP-specific servlet classes.

The servlet container (e.g., Tomcat) manages the servlet's lifecycle:

Loading and Instantiation: Servlet class is loaded into memory.
Initialization (init()): The servlet is initialized only once.
Service (service()): The servlet handles client requests using doGet() or doPost() methods.
Destruction (destroy()): The servlet is destroyed when no longer needed.

What is a Servlet?
Servlets are server-side programs written in Java.
They extend the capabilities of servers by processing requests (like HTTP requests) and producing responses.
Servlets run inside a Servlet Container (like Apache Tomcat) which manages their lifecycle.

# What Is a Servlet?

- The javax.servlet.Servlet and javax.servlet.http packages provide interfaces and classes for writing servlets.
- All servlets must implement the Servlet interface, which defines life-cycle methods.
- When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API.
- The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services. Web applications that use HTTP as the communication protocol, you will use the HttpServlet class

**Servlet Interface (javax.servlet.Servlet)**

```
public interface Servlet {
   void init(ServletConfig config) throws ServletException;   //perform initialization tasks
    //handles incoming client requests
   void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;
   void destroy();   //called when servlet to be destroyed
   String getServletInfo(); //provides information about the servlet
}
```

# Features of Servlet

- **Portable:**
  - For example, you can create a servlet on Windows operating system that users GlassFish as web server and later run it on any other operating system like Unix, Linux with Apache tomcat web server, this feature makes servlet portable.
- **Efficient and scalable:**
  - The web server invokes servlet using a lightweight thread so multiple client requests can be fulling by servlet at the same time using the multithreading feature of Java.

# Features of Servlet

- **Robust:**
  - the servlet is less prone to memory management issues and memory leaks (by inheriting the top features of Java (such as Garbage collection, Exception handling, Java Security Manager etc.))

# Servlet API

- Every Servlet must implement the java.servlet.Servlet interface, you can do it by extending one of the following two classes: javax.servlet.GenericServlet or javax.servlet.http.HttpServlet.
- The first one is for protocol independent Servlet and the second one for web applications that use HTTP as the communication protocol.

GnericServlet

HttpServlet

**javax.servlet.GenericServlet**
**javax.servlet.http.HttpServlet**

# Generic Servlet

**javax.servlet.GenericServlet class.**

- GenericServlet class has an abstract service() method. Which means the subclass of GenericServlet should always override the service() method.
  **Signature of service() method:**

```
public abstract void service(ServletRequest request, ServletResponse response)
    throws ServletException, java.io.IOException
```

- The service() method accepts two arguments ServletRequest object and ServletResponse object.
- The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

# Methods of Servlet interface

| S.No. | Method | Description |
|---|---|---|
| 1. | public void init(ServletConfigconfig) | It is used for initializing the servlet. It is invoked only once by the web container in a servlet life cycle. |
| 2. | public void service(ServletRequestreq, ServletResponse res) | It is used for providing a response to all the incoming request. It is invoked every time by the web container for each request. |
| 3. | public void destroy() | It is used for destroying the servlet. It is invoked only once in a life cycle of a servlet. |

# Key methods of HttpServlet class

- **doGet()** – This method is called by servlet service method to handle the HTTP GET request from client. **The Get method is used for getting information from the server**
- **doPost()** – **Used for posting information to the Server**
- **doPut()** – This method is **similar to doPost** method but unlike doPost method where we send information to the server, this method sends file to the server, this is **similar to the FTP operation from client to server**
- **doDelete()** – allows **a client to delete a document, webpage or information from the server**
- **init() and destroy()** – Used for managing resources that are held for the life of the servlet
- **getServletInfo()** – Returns information about the servlet, such as author, version, and copyright.

# Methods of HttpServlet interface

| S.No. | Method | Description |
|---|---|---|
| 1 | public void service(ServletRequest req,ServletResponse res) | It is used for securing the service method by creating objects of request and response. |
| 2 | protected void service(HttpServletRequest req, HttpServletResponse res) | It is used for receiving a service method. |
| 3 | protected void doGet(HttpServletRequest req, HttpServletResponse res) | It is invoked by the web container and it is used for handling the GET request. |
| 4 | protected void doPost(HttpServletRequest req, HttpServletResponse res) | It is invoked by the web container and it handles the POST request. |
| 5 | protected void doHead(HttpServletRequest req, HttpServletResponse res) | It is invoked by the web container and it handles the HEAD request. |

service() method of HttpServlet class listens to the Http methods (GET, POST etc) from request stream and invokes doGet() or doPost() methods based on Http Method

# Generic Servlet

- **Pros of using Generic Servlet:**
  1. Generic Servlet is easier to write
  2. Has simple lifecycle methods
  3. To write Generic Servlet you just need to extend javax.servlet.GenericServlet and override the service() method
- **Cons of using Generic Servlet:**

  Working with Generic Servlet is not that easy because methods such as doGet(), doPost(), doHead() etc does not exist in Generic Servlet that we can use in Http Servlet.
  - In Http Servlet we need to override particular convenience method for particular request,
  - for example to get information then override doGet(),
  - Similarly, to send information to server override doPost().
  - However in Generic Servlet only service() method need to be override for each type of request which is cumbersome.

INDEX.HTML

```
<html><body>
<form method="post" action="check">
   Name :<input type="text" name="user" >
    <input type="submit">
</form></body></html>
```

//Example for post!!!

**web.xml**

```
<!-- Define the servlet -->
<servlet>
   <servlet-name>check</servlet-name>
   <servlet-class>Servlet</servlet-class>
</servlet>
<!-- Map the servlet to a URL pattern -->
<servlet-mapping>
   <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
 </servlet-mapping>
```

map to class name

can be /abc

```
import javax.servlet.*;
import javax.servlet.http.*;

@WebServlet(urlPatterns = "/example")
public class Servlet extends HttpServlet {
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      try {
         String user = request.getParameter("user");
         out.println("<html><body>");
         out.println("<h2> Welcome "+user+"</h2>");
         out.println("</body></html>");
      } finally {
   }     out.close();
}     }
```

resposne is sets it's context!

response to be given!!

to get parameter as request from user!!

Program 1

http://<server>/check would invoke the check servlet.

**ExampleHttpServlet.java** //example for doGet()

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Creating Http Servlet by Extending HttpServlet class
public class ExampleHttpServlet extends HttpServlet
{
   private String mymsg;
   public void init() throws ServletException
   {
     msg = "Http Servlet Demo";
   }
   public void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException
   {
      // Setting up the content type of web page
      response.setContentType("text/html");
      // Writing the message on the web page
      PrintWriter out = response.getWriter();
      out.println("<h1>" + msg + "</h1>");
      out.println("<p>" + "Hello Students!" + "</p>");
   }
```

**index.html**

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Http Servlet Demo</title>
</head>
<body>
<a href="Demo">Click to call Servlet</a>
</body>
</html>
```

PrintWriter out=res.getWriter();

**web.xml** web.xml file is a **deployment descripter.**

```
<web-app>
<servlet>
<servlet-name>HttpServletDemo</servlet-name>
<servlet-class>ExampleHttpServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HttpServletDemo</servlet-name>
<url-pattern>/Demo</url-pattern>
</servlet-mapping>
</web-app>
```

Program 2

```
<!DOCTYPE html>
<html>
<body>
<form action="display" method="get">
<hr>
User name: <input type="text"
name="val1"> <br><br>
Password:   <input
type="password" name="val2" ><br><br>
<input type="submit" value="login">
</body>
</html>
```

//dispalying username and password after fetching!!

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class demo4 extends HttpServlet{
  public void doGet(HttpServletRequest
req,HttpServletResponse res)
  throws ServletException,IOException
  {
    res.setContentType("text/html");
    PrintWriter pwriter=res.getWriter();
    String uname=req.getParameter("val1");
    String pw=req.getParameter("val2");
    pwriter.println("User Details Page:");
    pwriter.println("Hello "+uname);
    pwriter.println("Your Password is **"+pw+"**");
    pwriter.close();   } }
```

Web.xml

```
<servlet>
 <servlet-name>abc3</servlet-name>
 <servlet-class>demo4</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>abc3</servlet-name>
 <url-pattern>/display</url-pattern>
 </servlet-mapping>
```

res.setContentType("text/html");
req.getParameter();

Program 4

//Deployment descriptor!!
web.xml

First line of any xml document

```
<?xml version="1.0" encoding="UTF-8"?>
```

root tag of wex.xml file. All other tag come inside it

```
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
```

this tag maps internal name to fully qualified class name

Give a internal name to your servlet

```
    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-class>MyServlet</servlet-class>
    </servlet>
```

servlet class that you have created

this tag maps internal name to public URL name

```
    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
```

URL name. This is what the user will see to get to the servlet

```
</web-app>
```

## Demo.html

Program 1

```
<!DOCTYPE html>
<html>
<head>
  <title>Student Grade Calculator</title>
  <style>
    form {
      max-width: 500px;
      margin: 0 auto;
      padding: 20px;
      border: 1px solid #ccc;
      border-radius: 10px;
    }
    input[type="text"] {
      width: 100%;
      padding: 8px;
      margin: 5px 0 15px 0;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    input[type="submit"] {
      width: 100%;
      padding: 10px;
      background-color: red;
      color: yellow;
    }
  </style>
</head>
```

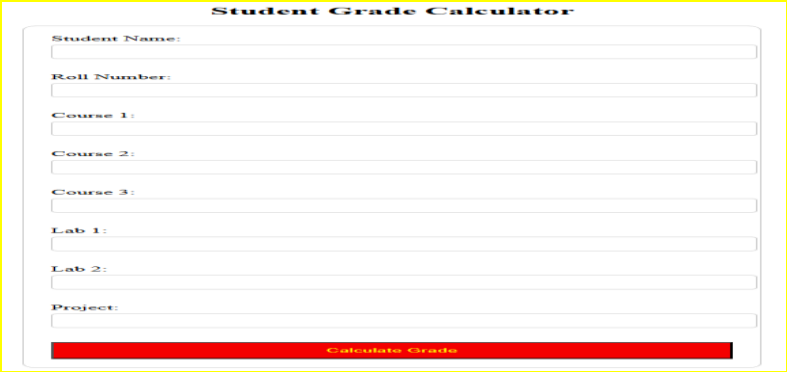
```
<body>
<h2 align="center">Student Grade Calculator</h2>
<form action="calculate" method="post">   url-pattern!!
  <label for="studentName">Student Name:</label>
  <input type="text" id="studentName" name="studentName" required>
  <br><br>                                   request.getParameter("studentName");
  <label for="rollNo">Roll Number:</label>
  <input type="text" id="rollNo" name="rollNo" required><br><br>
  <label for="subject1">Course 1:</label>
  <input type="text" id="subject1" name="subject1" required><br><br>
  <label for="subject2">Course 2:</label>
  <input type="text" id="subject2" name="subject2" required><br><br>
  <label for="subject3">Course 3:</label>
  <input type="text" id="subject3" name="subject3" required><br><br>
  <label for="lab1">Lab 1:</label>
  <input type="text" id="lab1" name="lab1" required><br><br>
  <label for="lab2">Lab 2:</label>
  <input type="text" id="lab2" name="lab2" required><br><br>
  <label for="project">Project:</label>
  <input type="text" id="project" name="project" required><br><br>
  <input type="submit" value="Calculate Grade">
</form>
</body>
</html>
```

## Student.java

```
public class Student {
  private String name, grade, rollNo;
  private int subject1,subject2,subject3,lab1,lab2,project;
  private double average;
  private String grade;

  public Student() {}

  // Getters and Setters
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getRollNo() { return rollNo; }
  public void setRollNo(String rollNo) { this.rollNo = rollNo; }
  public int getSubject1() { return subject1; }
  public void setSubject1(int subject1) { this.subject1 = subject1; }
  public int getSubject2() { return subject2; }
  public void setSubject2(int subject2) { this.subject2 = subject2; }
  public int getSubject3() { return subject3; }
  public void setSubject3(int subject3) { this.subject3 = subject3; }
  public int getLab1() { return lab1; }
  public void setLab1(int lab1) { this.lab1 = lab1; }
  public int getLab2() { return lab2; }
  public void setLab2(int lab2) { this.lab2 = lab2; }
  public int getProject() { return project; }
  public void setProject(int project) { this.project = project; }
  public double getAverage() { return average; }
  public void setAverage(double average) { this.average = average; }
   public String getGrade() { return grade; }
  public void setGrade(String grade) { this.grade = grade; }
```

## Program 2

```
  // calculate average
  public void calculateAverage() {
     this.average = (subject1 + subject2 + subject3 +
lab1 + lab2 + project) / 6.0;
  }

  // calculate grade
  public void calculateGrade() {
     if (average >= 90) grade = “O";
     else if (average >= 80) grade = "A+";
     else if (average >= 70) grade = “A";
     else if (average >= 60) grade = “B+";
     else if (average >= 50) grade = “B";
     else grade = “U";
  }
}
```



# Student.java

Program 2

```
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.WebServlet;

@WebServlet("/calculate")
public class GradeCalculatorServlet extends HttpServlet {
 public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      try {
         // Student object
         Student student = new Student();
         student.setName(request.getParameter("studentName"));
         student.setRollNo(request.getParameter("rollNo"));
         student.setSubject1(Integer.parseInt(request.getParameter("subject1")));
         student.setSubject2(Integer.parseInt(request.getParameter("subject2")));
         student.setSubject3(Integer.parseInt(request.getParameter("subject3")));
         student.setLab1(Integer.parseInt(request.getParameter("lab1")));
         student.setLab2(Integer.parseInt(request.getParameter("lab2")));
         student.setProject(Integer.parseInt(request.getParameter("project")));

         // Calculate average and grade
         student.calculateAverage();
         student.calculateGrade();
```

//setters used to setValue for student!!

//gettersUsed to Display!!

```
      // Display result
         out.println("<html><head><title>Grade Result</title></head>");
         out.println("<body>");
         out.println("<h2 align='center'>Student Grade Report</h2>");
         out.println("<table border='1' align='center'>");
         out.println("<tr><td>Name:</td><td>" + student.getName() + "</td></tr>");
         out.println("<tr><td>Roll Number:</td><td>" + student.getRollNo() +
"</td></tr>");
         out.println("<tr><td>Subject 1:</td><td>" + student.getSubject1() +
"</td></tr>");
         out.println("<tr><td>Subject 2:</td><td>" + student.getSubject2() +
"</td></tr>");
         out.println("<tr><td>Subject 3:</td><td>" + student.getSubject3() +
"</td></tr>");
         out.println("<tr><td>Lab 1:</td><td>" + student.getLab1() + "</td></tr>");
         out.println("<tr><td>Lab 2:</td><td>" + student.getLab2() + "</td></tr>");
         out.println("<tr><td>Project:</td><td>" + student.getProject() + "</td></tr>");
         out.println("<tr><td>Average:</td><td>" + String.format("%.2f",
student.getAverage()) + "</td></tr>");
         out.println("<tr><td>Final Grade:</td><td>" + student.getGrade() +
"</td></tr>");
         out.println("</table>");
         out.println("<p align='center'><a href='index.html'>Calculate Another
Grade</a></p>");
         out.println("</body></html>");

      } catch (NumberFormatException e) {
         out.println("<html><body>");
         out.println("<h3>Error: Please enter valid numbers</h3>");
         out.println("<a href='index.html'>Go Back</a>");
         out.println("</body></html>");
      }
   }
}
```

Program 4

## Web.xml

//Deployment descriptor!!

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <servlet>
  <servlet-name>GradeCalculator</servlet-name>
  <servlet-class>GradeServlet</servlet-class>
</servlet>

  <servlet-mapping>
    <servlet-name>GradeCalculator</servlet-name>
    <url-pattern>/calculate</url-pattern>
  </servlet-mapping>

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

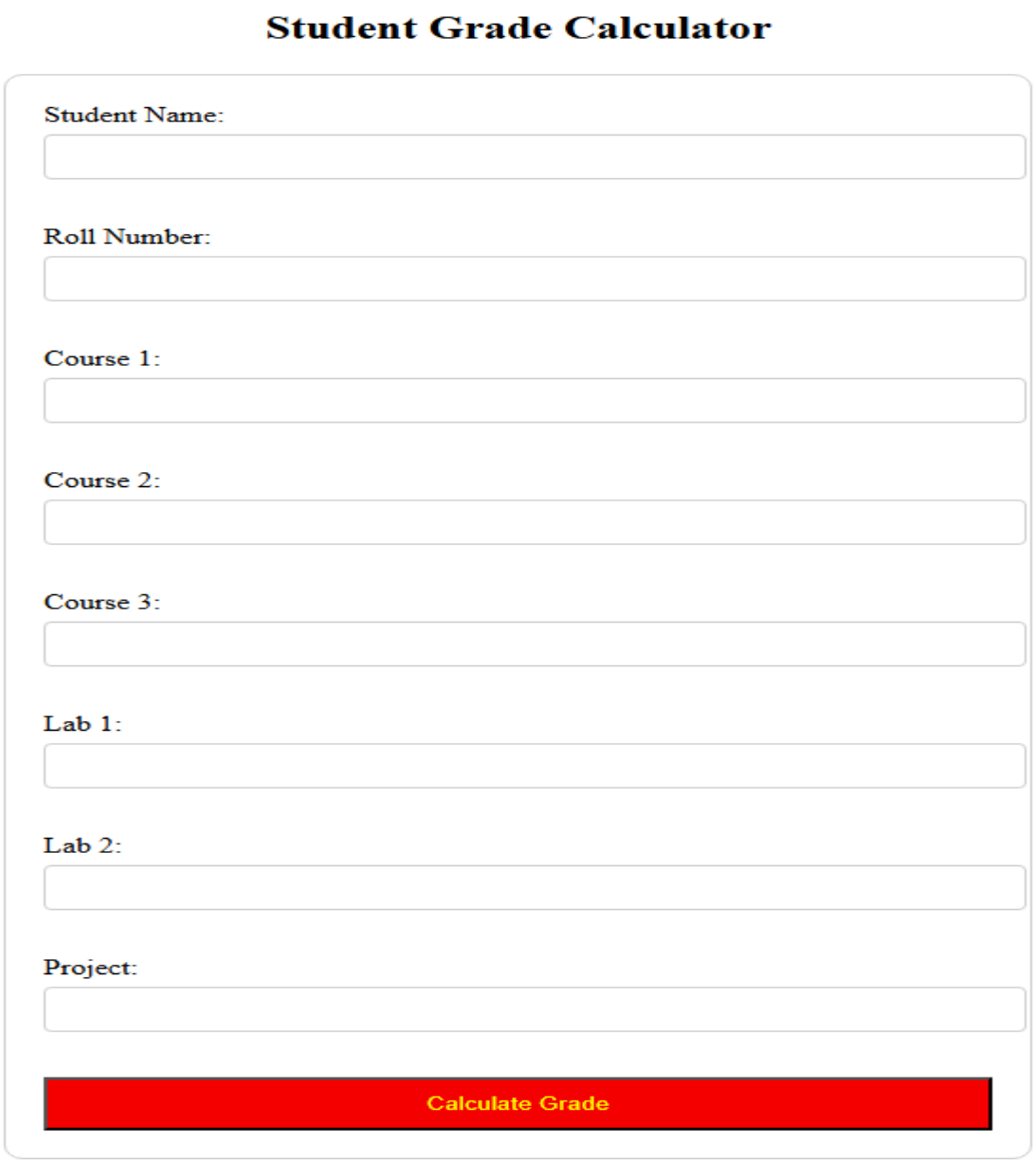
servlet name that link both SERVLET and SERVLET-MAPPING!!

Our created class!! GradeServlet!

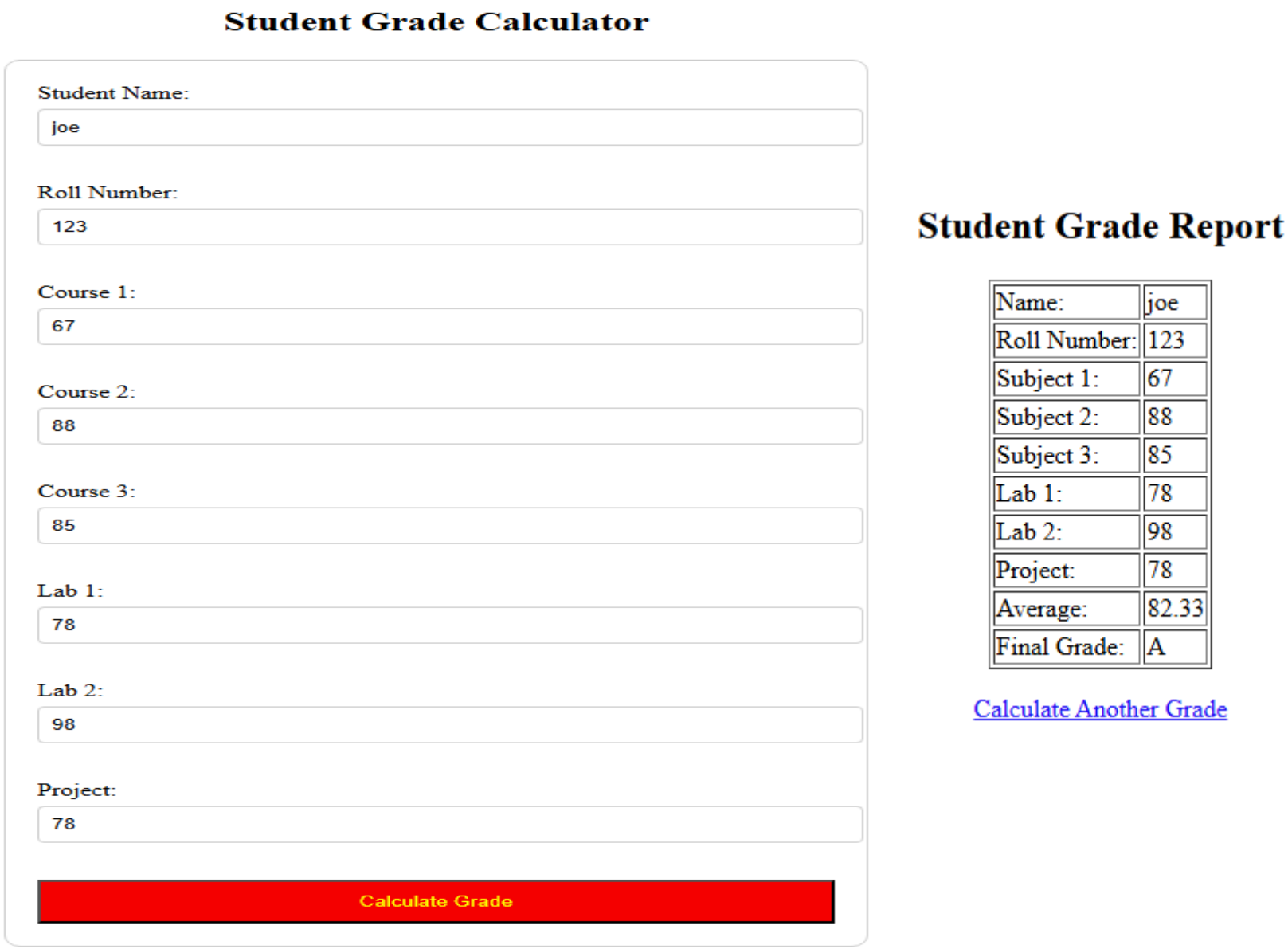
<form action="calculate" method="post">

pg26

# Output screenshot



# Output screenshot




## Student Grade Report

| | |
|---|---|
| Name: | joe |
| Roll Number: | 123 |
| Subject 1: | 67 |
| Subject 2: | 88 |
| Subject 3: | 85 |
| Lab 1: | 78 |
| Lab 2: | 98 |
| Project: | 78 |
| Average: | 82.33 |
| Final Grade: | A |

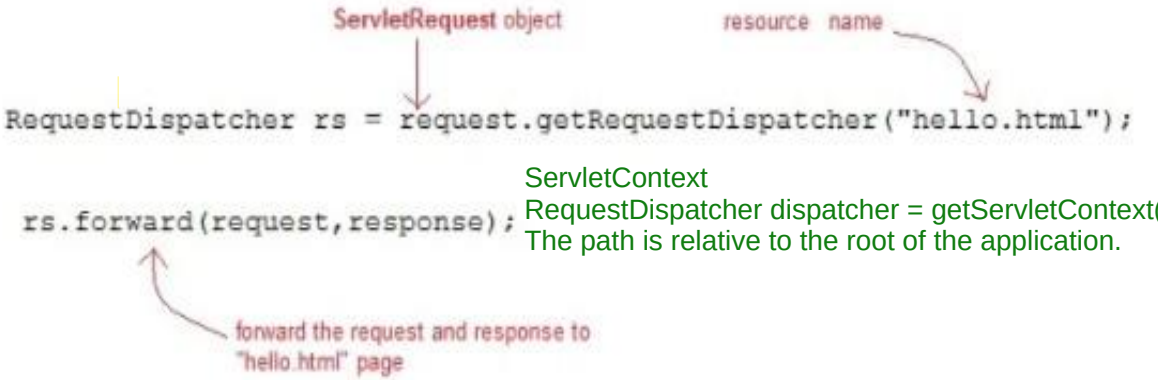
Calculate Another Grade

# How to get an Object of RequestDispatcher

`getRequestDispatcher()` method of **ServletRequest** returns the object of **RequestDispatcher**.

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");
rs.forward(request,response);
```

HttpServletRequest
RequestDispatcher dispatcher = request.getRequestDispatcher("/path");




ServletContext
RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/path");
The path is relative to the root of the application.

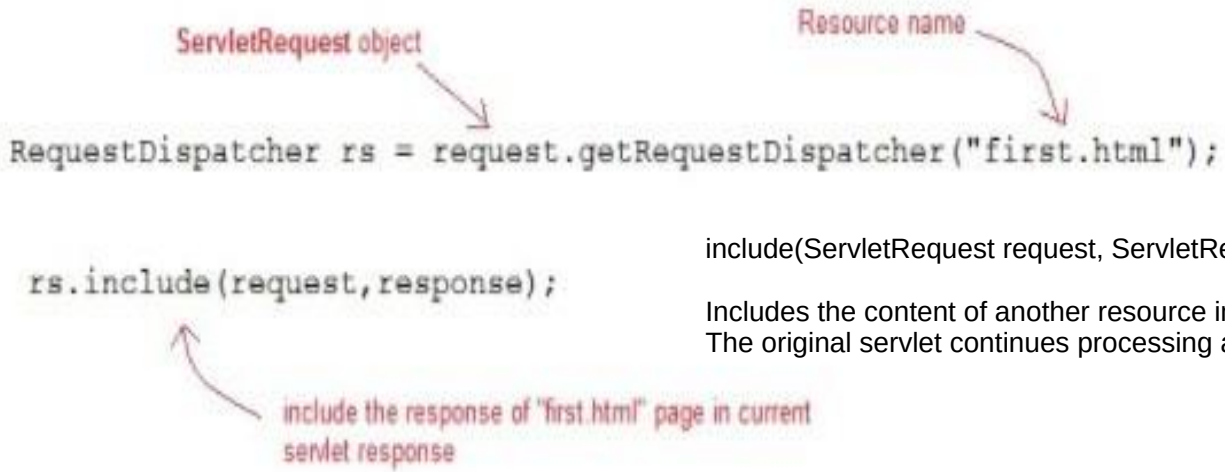
In Java, a RequestDispatcher is an interface in the javax.servlet package used in Servlet programming. It provides a way to forward a request from one servlet to another servlet or resource (like a JSP, HTML, or another servlet) within the same server context or to include the content of another resource in the response.

forward(ServletRequest request, ServletResponse response)

Forwards the request to another resource (servlet, JSP, etc.).
The control is transferred to the destination resource, and the original servlet stops processing after the forward.

OR

```
RequestDispatcher rs = request.getRequestDispatcher("hello.html");
rs.include(request,response);
```




include(ServletRequest request, ServletResponse response)

Includes the content of another resource in the response.
The original servlet continues processing after the include operation.

RequestDispatcher is used to **forward** or **include** response of a resource in a Servlet. Here we are using **index.html** to get username and password from the user, **Validate** Servlet will validate the password entered by the user, if the user has entered "studytonight" as password, then he will be forwarded to **Welcome** Servlet else the user will stay on the index.html page and an error message will be displayed.

**Files to be created :** //Sample programs for login!(welcome page)

- **index.html** will have form fields to get user information.
- **Validate.java** will validate the data entered by the user.
- **Welcome.java** will be the welcome page.
- **web.xml** , the deployment descriptor.

# Servlet: Methods of RequestDispatcher

**RequestDispatcher** interface provides two important methods

| Methods | Description |
|---|---|
| public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException | It is used for forwarding the request from one servlet to another servlet on a server. |
| public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException | It is used for including the content of the resource in the response. |

```
<form method="post" action="Validate">
Name:<input type="text" name="user"
/><br/>
Password:<input type="password"
name="pass" ><br/>
<input type="submit" value="submit">
</form>
```

getParameter()!

web.xml //Deployment discriptor

```
<web-app>
  <servlet>
    <servlet-name>Validate</servlet-name>
    <servlet-class>Validate</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>Welcome</servlet-name>
    <servlet-class>Welcome</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Validate</servlet-name>
    <url-pattern>/Validate</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

class name

action!

Validate.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Validate extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      try {
         String name = request.getParameter("user");
         String password = request.getParameter("pass");
         if(password.equals("hi"))
         {
                                                                    class name
            RequestDispatcher rd = request.getRequestDispatcher("Welcome");
            rd.forward(request, response);
         }
         else
         {
            out.println("<font color='red'><b>You have entered incorrect password</b></font>");
            RequestDispatcher rd = request.getRequestDispatcher("index.html");
            rd.include(request, response);
         }
      }
      finally {
         out.close();      }  }}
```

Welcome.java

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*;

public class Welcome extends HttpServlet {

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
         response.setContentType("text/html;charset=UTF-8");
         PrintWriter out = response.getWriter();
         try {
            out.println("<h2>Welcome user</h2>");
         }
         finally {
            out.close();
         }

     }
  }
```