JAVA BACKEND DEVELOPMENT PROGRAM

Web Application & Servlet

OUTLINE

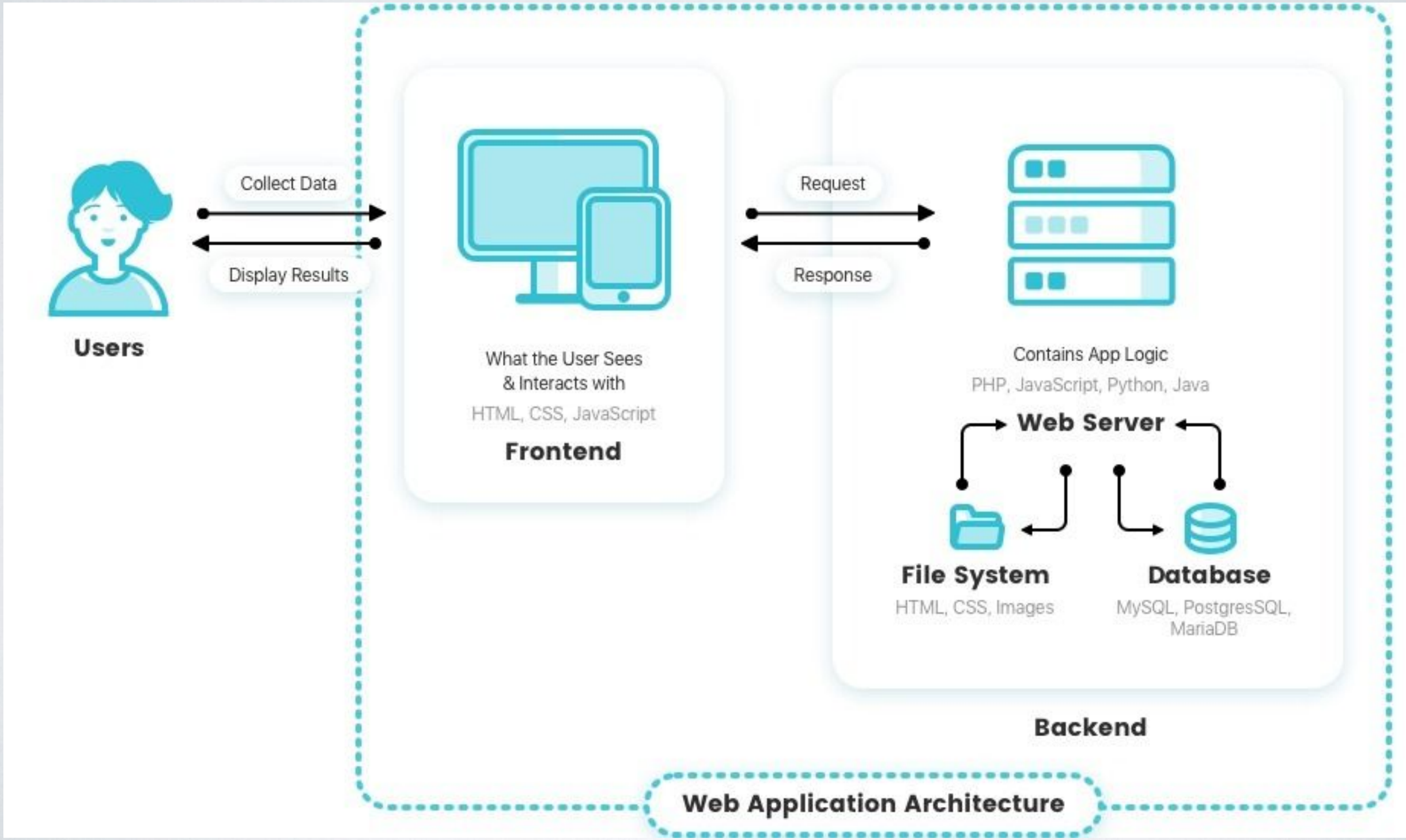
- Client
- Web application
    - HTTP Request
    - HTTP Response
- Web server
- Servlet

WEB APPLICATION

- A web application is a computer program that utilizes web browsers and web technology to perform tasks over the Internet.

- Web applications use a combination of *server-side* scripts (like Java) to handle the storage and retrieval of the information, and *client-side* scripts (JavaScript and HTML) to present information to users.

WEB APPLICATION



CLIENT

- A client in web application usually refers to a server that presents the user interface which a user actually interacts with, like browsers.
- Languages supported by web browsers:
  - HTML
  - CSS
  - JavaScript

HTML

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

CSS

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

JAVASCRIPT

- JavaScript is the programming language of HTML and the Web.
  - HTML to define the content of web pages
  - CSS to specify the layout of web pages
  - JavaScript to program the behavior of web pages
- HTML and CSS will only produce static web pages. JavaScript will make it dynamic by different events to respond to different user interactions

HTTP

- HTTP stands for Hypertext Transfer Protocol and is used to structure requests and responses over the internet.

- HTTP requires data to be transferred from one point to another over the network.

HTTP REQUEST

- ***HTTP Request*** is a packet of Information that one computer sends to another computer to communicate something

- To its core, *HTTP Request* is a packet of binary data sent by the Client to server.

- An *HTTP Request* contains following parts

  - Request Line

  - Headers, 0 or more Headers in the request

  - An optional Body of the Request

REQUEST LINE

- A ***Request Line*** specifies the Method Token (***GET, PUT*** …) followed by the Request URI and then the *HTTP Protocol* that is being used

```
----------------------Request--------------------------------
Request Line: GET /utilities/weatherfull/city/hyderabad HTTP/1.1
Request Method: GET
Request Time: 2017-08-27 13:30:30
Accept-Encoding: gzip, deflate
Host address: restapi.demoqa.com
Client IP: 84.245.10.101
Client Port: 42092
HTTP Protocol Version: HTTP/1.1
Connection: close
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Request body:
```

HTTP REQUEST METHOD

- HTTP defines a set of **request methods** to indicate the desired action to be performed for a given resource.
  - GET — The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- POST — The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- PUT — The PUT method replaces all current representations of the target resource with the request payload.
- DELETE — The DELETE method deletes the specified resource.
- HEAD, CONNECT, OPTIONS, TRACE, PATCH

POST VS. PUT

- POST — The POST method is used to request that the origin server accept the entity enclosed in the request as *a new subordinate* of the resource identified by the Request-URI in the Request-Line
  - In other word, POST is used to create/add
- PUT — The PUT method requests that the enclosed entity be stored under the supplied Request-URI.
  - If the Request-URI refers to an already *existing resource*, the enclosed entity SHOULD be considered as a *modified* version of the one residing on the origin server.
  - If the Request-URI does *not* point to an existing resource, and that URI is capable of being defined as a *new resource* by the requesting user agent, the origin server can create the resource with that URI.
  - In other word, PUT is used to create/add and update

POST VS. PUT

- When to use
  - Do you name your URL objects you create explicitly, or let the server decide? If you name them then use PUT. If you let the server decide then use POST.
- PUT is idempotent, so if you PUT an object twice, it has no effect.
- You can update or create a resource with PUT with the same object URL
- POST twice with the same data means that create two identical users with different ids

POST VS. PUT

- Pragmatic Advice
  - To save an existing user, or one where the ***client generates the id*** and it's been verified that the id is unique
    - PUT /user/12345 HTTP/1.1 <-- create the user providing the id 12345
    - GET /user/12345 HTTP/1.1 <-- return that user
  - Otherwise, use POST to initially create the object, and PUT to update the object:
    - POST /user HTTP/1.1 <--- create the user, server returns 12345
    - PUT /user/12345 HTTP/1.1 <--- update the user
- When it comes to system design, we only need to choose one either POST or PUT and support it well in the application.

URL VS. URI

- URL — uniform resource ***locator***
- URI — uniform resource ***identifier***
- Look at an example

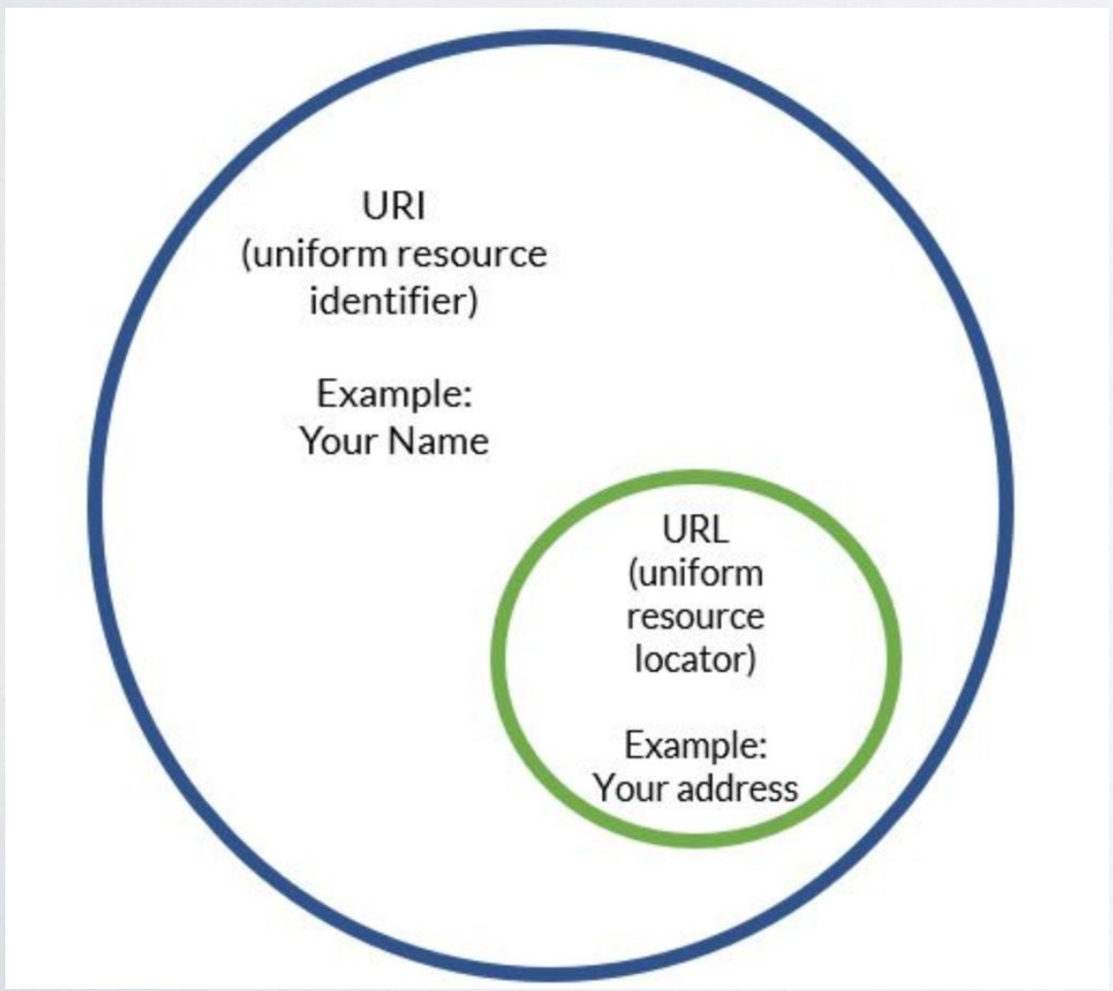
```
http://beginwithjava.com:80/servlet-jsp/introduction/url.html
|---| |----------------||--||--------------------------------|
  1         2              3           4
```

  - 1 — Protocol name (HTTP)
  - 2 — Host name
  - 3 — Port number(Optional)
  - 4 — Path

URL VS. URI

- URI can be a name, locator, or both for an online resource where a URL is just the locator

- URLs are a subset of URIs. That means all URLs are URIs.



- Your name could be a URI because it identifies you, but it couldn't be a URL because it doesn't help anyone find your location

- On the other hand, your address is both a URI and a URL because it both identifies you *and* it provides a location for you.

REQUEST HEADER

- In the request section, whatever follows ***Request Line*** till before ***Request Body*** everything is a *Header*

- *Headers are used to pass additional information about the request to the server*

```
----------------------Request--------------------------------

Request Line: GET /utilities/weatherfull/city/hyderabad HTTP/1.1

Request Method: GET

Request Time: 2017-08-27 13:30:30

Accept-Encoding: gzip, deflate

Host address: restapi.demoqa.com

Client IP: 84.245.10.101

Client Port: 42092

HTTP Protocol Version: HTTP/1.1

Connection: close

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

Request body:
```

REQUEST BODY

- ***Request Body*** is the part of the *HTTP Request* where additional content can be sent to the server.
  - eg. a file type of JSON or XML.
- It is optional.

```
----------------------Request-------------------------------

Request Line: GET /utilities/weatherfull/city/hyderabad HTTP/1.1

Request Method: GET

Request Time: 2017-08-27 13:30:30

Accept-Encoding: gzip, deflate

Host address: restapi.demoqa.com

Client IP: 84.245.10.101

Client Port: 42092

HTTP Protocol Version: HTTP/1.1

Connection: close

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.101 Safari/537.36

Accept-Language: en-GB,en-US;q=0.8,en;q=0.6

Request body:
```

HTTP RESPONSE

- ***HTTP Response*** is the packet of information sent by *Server* to the *Client* in response to an earlier *Request* made by *Client*

- *HTTP Response* contains the information requested by the Client

- Just like HTTP Request, HTTP Response also has the same structure:

  - Status Line

  - Headers, 0 or more Headers in the request

  - An optional Body of the Request

STATUS LINE

- A ***Status Line*** consists of three parts:
  - HTTP Protocol Version
  - Status Code
  - Reason Phrase

```
Status Line: HTTP/1.1 200 OK

Response status code -> 200 OK

Server: openresty

Date: Sun, 27 Aug 2017 13:30:30 GMT

Content-Type: application/json; charset=utf-8

Content-Length: 536

Connection: close

X-Cache-Key: /data/2.5/weather?APPID=199c0c704dcd69ff1a88fc99385dae08&q=hyderabad

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST

Response Body:

{
    "City": "Hyderabad",
    "Temperature": "24.51 Degree celsius",
    "Humidity": "94 Percent",
    "Weather Description": "thunderstorm with light rain haze",
    "Wind Speed": "4.92 Km per hour",
    "Wind Direction degree": "279.501 Degree"
}
```

HTTP STATUS CODE

- There are five classes of status code
  - 1xx Informational – the request was received, continuing process
  - 2xx Successful – the request was successfully received, understood and accepted
  - 3xx Redirection – further action needs to be taken in order to complete the request
  - 4xx Client Error – the request contains bad syntax or cannot be fulfilled
  - 5xx Server Error – the server failed to fulfill an apparently valid request
- Full status code reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

RESPONSE HEADER

- Just like a *Request Header*, *Response Header* also contains zero or more *Header lines*. However, it is very uncommon to have zero Headers in the response.

- Lines just after the ***Status Line*** and before the ***Response Body*** are all ***Response Headers*** lines

```
Status Line: HTTP/1.1 200 OK

Response status code -> 200 OK

Server: openresty

Date: Sun, 27 Aug 2017 13:30:30 GMT

Content-Type: application/json; charset=utf-8

Content-Length: 536

Connection: close

X-Cache-Key: /data/2.5/weather?APPID=199c0c704dcd69ff1a88fc99385dae08&q=hyderabad

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST

Response Body:

{
```

RESPONSE BODY

- ***Response Body*** contains the resource data that was requested by the client.

```
Status Line: HTTP/1.1 200 OK

Response status code -> 200 OK

Server: openresty

Date: Sun, 27 Aug 2017 13:30:30 GMT

Content-Type: application/json; charset=utf-8

Content-Length: 536

Connection: close

X-Cache-Key: /data/2.5/weather?APPID=199c0c704dcd69ff1a88fc99385dae08&q=hyderabad

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

Access-Control-Allow-Methods: GET, POST

Response Body:

{
    "City": "Hyderabad",
    "Temperature": "24.51 Degree celsius",
    "Humidity": "94 Percent",
    "Weather Description": "thunderstorm with light rain haze",
    "Wind Speed": "4.92 Km per hour",
    "Wind Direction degree": "279.501 Degree"
}
```

WEB SERVER

- A program that uses HTTP for serving files that create web pages for users in response to their requests that are sent by the HTTP clients of their computer is called as a web server.

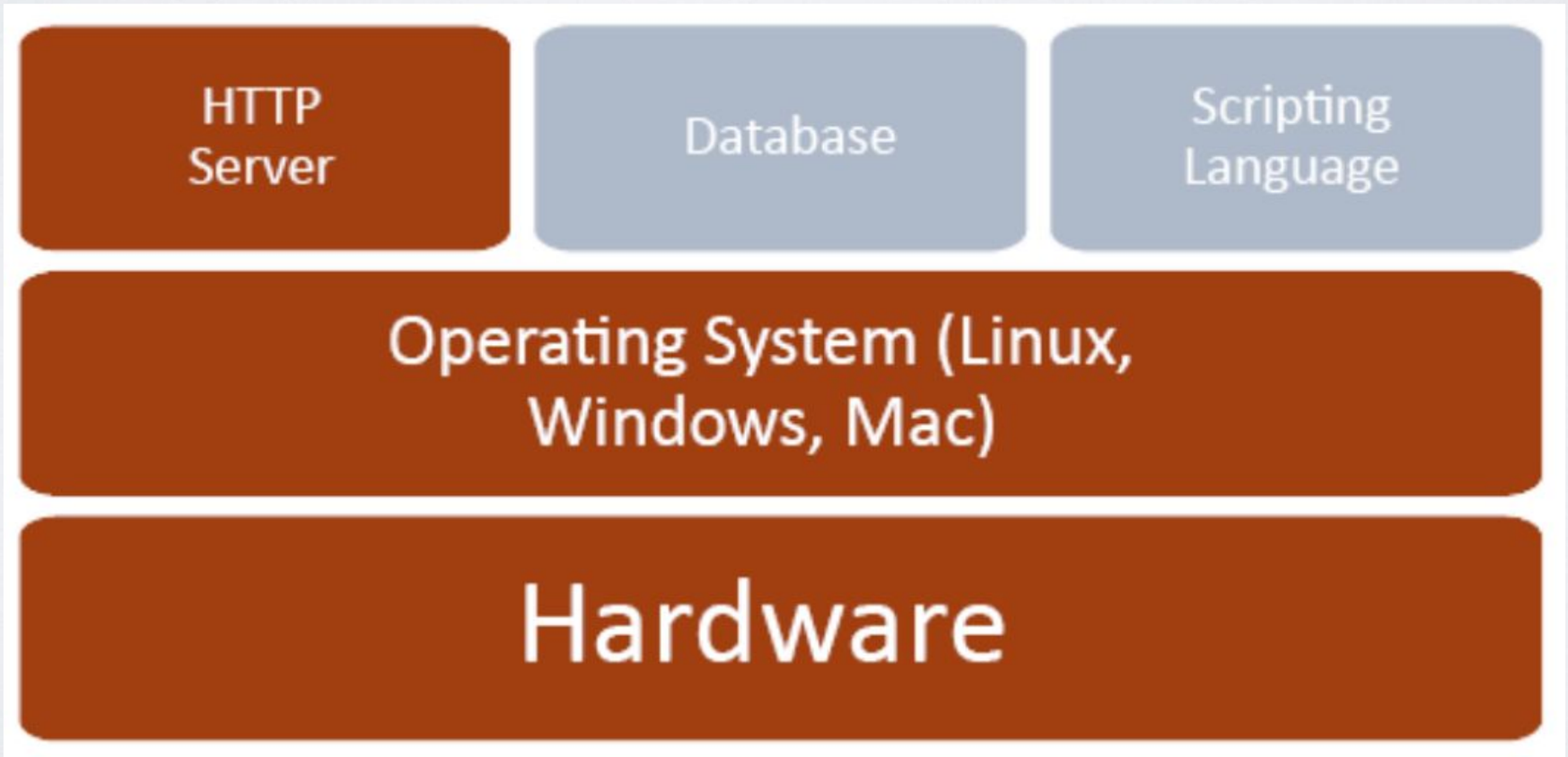
- This server is always connected to the internet.

- Every Web server that is connected to the Internet is given a unique address made up of a series of four numbers between 0 and 256 separated by periods.

  - eg. 68.178.157.132

WEB SERVER

- Computers hosting websites are web servers
- The diagram below represents the basic elements of a web server



WEB SERVER

- When you register a web address, also known as a domain name, such as google.com you have to specify the IP address of the Web server that will host the site
- Then you can load up with Dedicated Servers that can support your web-based operations
- There are four leading web servers
  - Apache HTTP Server (like Tomcat) — widely used
  - Nginx Web Server — high performance, stability, simple configuration and low resource usage.
  - Google Web Server (GWS)
- Except for those, there are several other web servers in market like Bea's Web Logic and IBM's WebSphere

WEB SERVER

- Other terminologies
  - SMTP Server — **S**imple **M**ail **T**ransfer **P**rotocol Server. This server takes care of delivering emails from one server to another server
  - FTP — **F**ile **T**ransfer **P**rotocol. a standard network protocol used for the transfer of computer files between a client and server on a computer network
- ISP — **I**nternet **S**ervice **P**rovider. They are the companies who provide you service in terms of internet connection to connect to the internet.
  - eg. You will buy space on a Web Server from any Internet Service Provider. This space will be used to host your Website.
- DNS — **D**omain **N**ame **S**ystem.
  - When someone types in your domain name, www.example.com, your browser will ask the Domain Name System to find the IP that hosts your site. When you register your domain name, your IP address should be put in a DNS along with your domain name

WEB APPLICATION

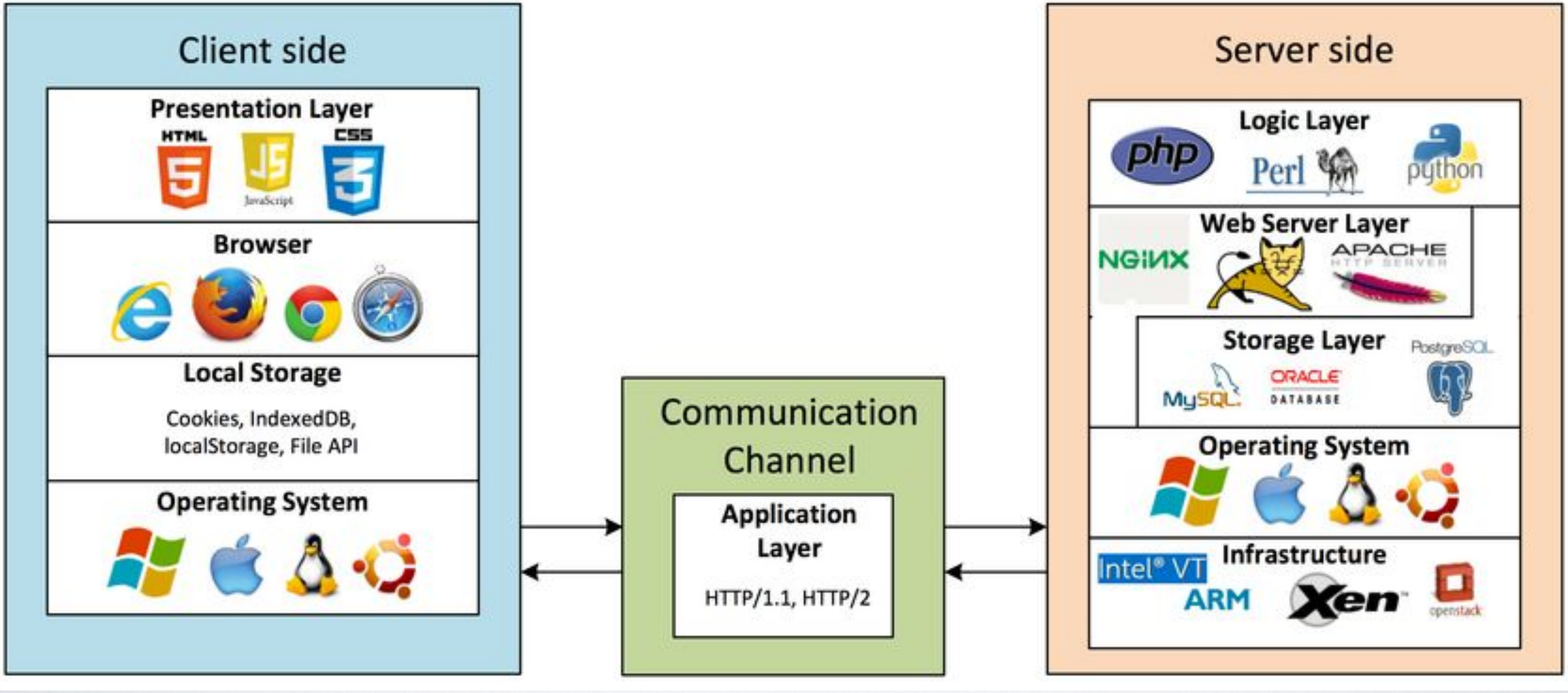
- Until now, we have already covered all basics in web application.

- So what happens when I type google.com in the browser? Or how does the web application works?

HOW IT WORKS

- A user enters a URL into a browser (for example, Google.com. This request is passed to a domain name server.
- The domain name server returns an IP address for the server that hosts the Website (for example, 68.178.157.132).
- The browser requests the page from the Web server using the IP address specified by the domain name server. (URL)
- The Web server returns the page to the IP address specified by the browser requesting the page. The page may also contain links to other files on the same server, such as images, which the browser will also request.
- The browser collects all the information and displays to your computer in the form of Web page.

WEB APPLICATION



HOW IT WORKS

- How does web application know how to process a request?
- We need to implement the logic, so web application knows what should be returned as a response. This is called **business logic**.
- We use **Servlet** to implement the business logic.

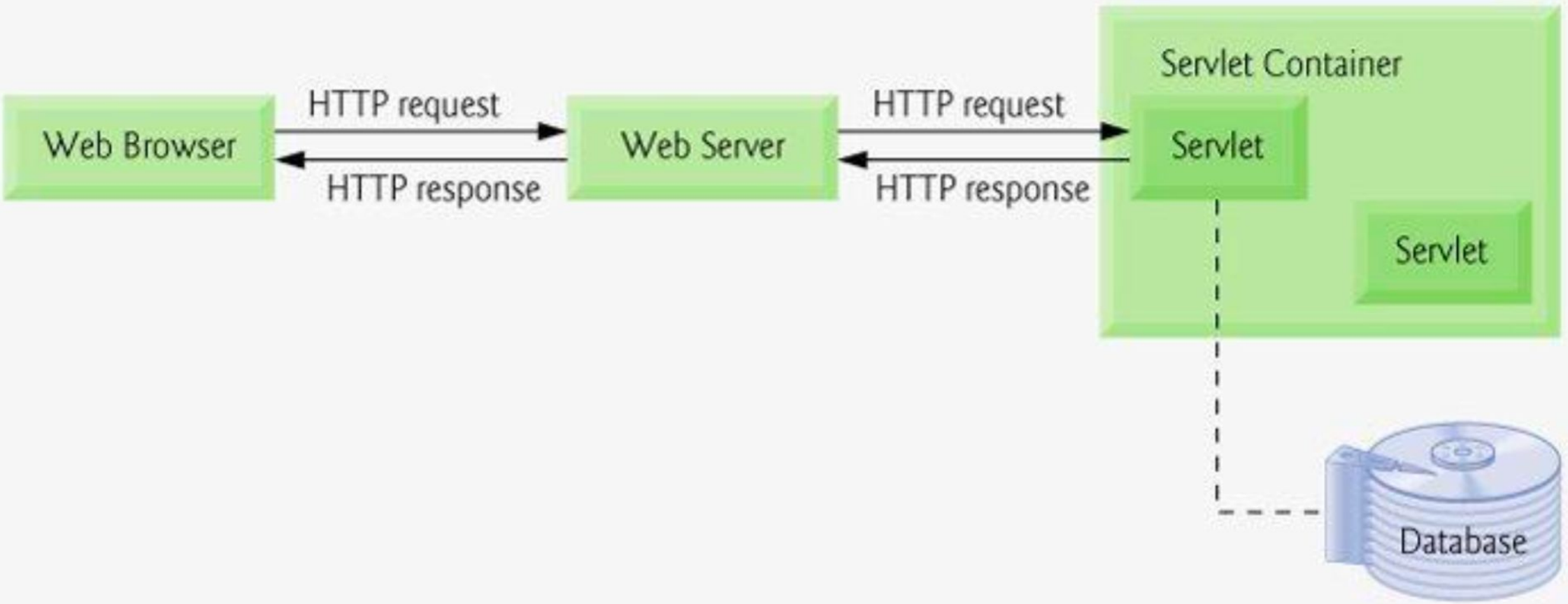
WHAT IS SERVLET

- Servlet can be described in many ways, depending on the context.
  - Servlet is a technology i.e. used to create web application.
  - Servlet is an API that provides many interfaces and classes include documentations,
  - Servlet is class that extend the capabilities of the servers and respond to the incoming request. It can respond to any types of requests.
  - Servlet is a web component that is deployed on the server to create dynamic web page.

WHAT IS SERVLET

- Servlet is a Java Interface. It provides standards for process HTTP requests.
  1. what should you do when initialized
  2. what should you do when destroyed
  3. what should you do when receiving request

SERVLET ARCHITECTURE



SERVLET CONTAINER

- Servlet container (also known as a web container) is the component of a **web server** that interacts with **Java servlets**.
- A servlet container is responsible for managing the ***lifecycle*** of servlets, ***mapping*** a URL to a particular servlet and ensuring that the URL requester has the correct *access-rights*.
- Common web container
  - Open Source
    - **Apache Tomcat**, GlassFish, WildFly
  - Commercial
    - JBoss , WebLogic Application Serve, IBM WebSphere Application Server,

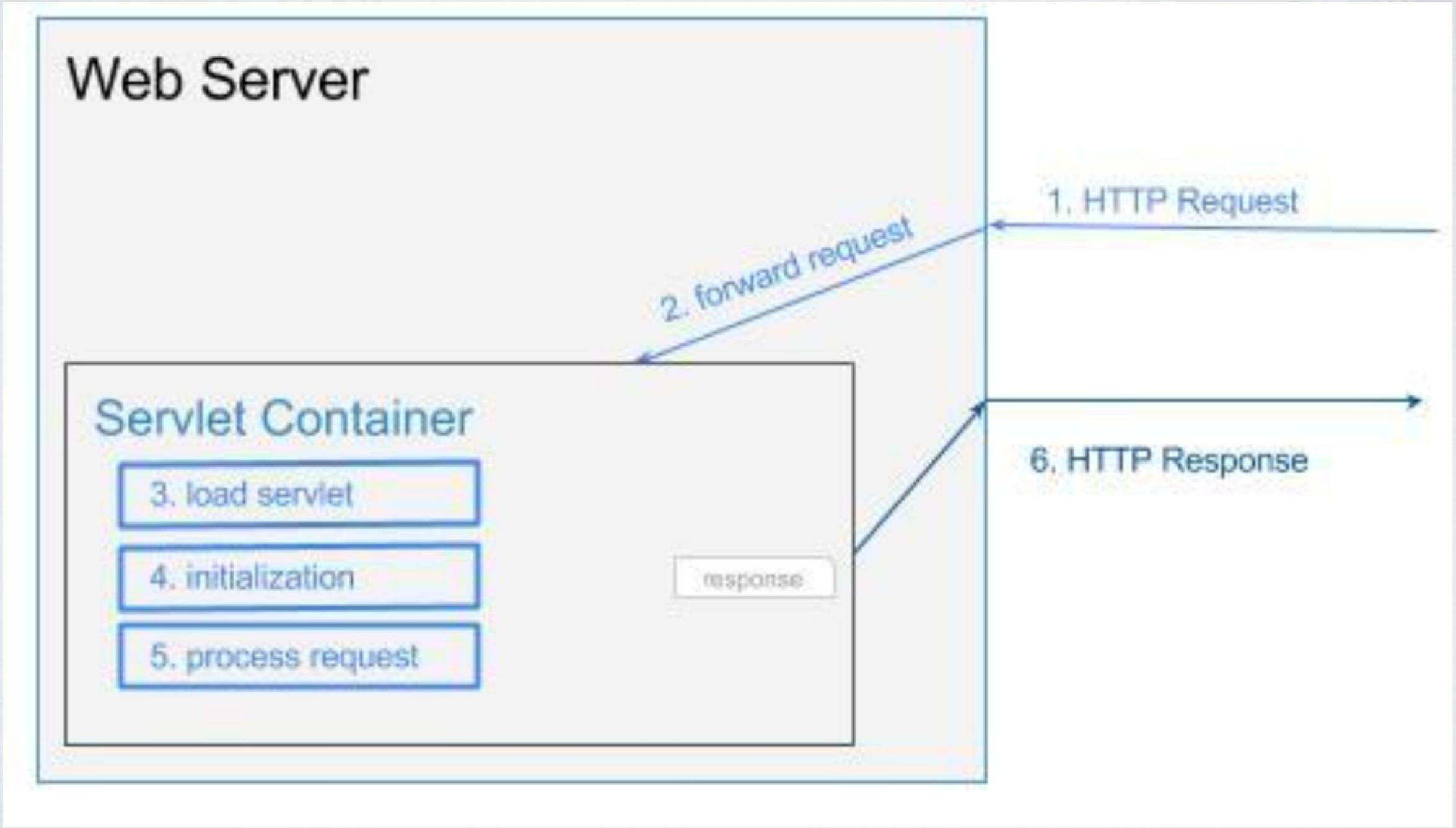
APACHE TOMCAT

- **Apache Tomcat** is a popular open source web server and **Servlet container** for Java code.
- Tomcat provides a "pure Java" HTTP web server environment in which Java code can run.
- Tomcat provides **dynamic content** by employing Java-based logic, while the Apache web server's primary purpose is to simply serve up **static content** such as HTML, images, audio and text.

HOW SERVLET PROCESS A REQUEST

- Web server receives HTTP request
- Web server forwards the request to servlet container
- If the required servlet is not in the container, it will be dynamically retrieved and loaded into address space
- The container invokes the init() method of the servlet for initialization (invoked once when the servlet is loaded first time)
- The container invokes the service() method of the servlet to process the HTTP request, i.e., read data in the request and build a response. The servlet remains in the container's address space and can process other HTTP requests.
- Web server return the dynamically generated results to the correct location

HOW SERVLET PROCESS A REQUEST



SERVLET LIFE CYCLE

- A servlet life cycle can be defined as the entire process from its creation till the destruction.
- The following are the paths followed by a servlet
  - The servlet is initialized by calling the init() method.
  - The servlet calls service() method to process a client's request.
  - The servlet is terminated by calling the destroy() method.
  - Finally, servlet is garbage collected by the garbage collector of the JVM.

INIT()

- The init method is designed to be called only once.
- It is called when the servlet is first initialized. It will not be called again for other requests. So, it is used for one-time initializations.
- The initialization of a servlet is normally at time when user first make request call to the URL corresponding to the servlet, but we can also change the configuration and load the servlet when the server is activated
- When user make a request to a servlet, an instance of this servlet will be generated. After that, each request will be handled by a separate thread, and call the corresponding method.
- If we want to load some data, we should also do that in init() method, these data will be used through entire lifecycle

SERVICE()

- The service() method is the *main* method to perform the actual task. Once web server receives a request method, it will call the service() method of corresponding servlet. After processing, it will return the formatted response to the client
- Each time the server receives a servlet request, server will generate a new thread and invoke the service() method. Service() method will detect request type (GET, POST...) and call corresponding method from the servlet (doGET, doPOST...)
- The service method is invoked by the server. Then the service() method will invoke doGET, doPOST or other methods based on request type. Thus, we do not need to make any changes to service() method. All we need is to override doGET or doPOST method based on business requirement.
- The doGet() and doPost() are most frequently used methods within each service request.

DESTROY()

- The destroy() method is called only *once* at the end of the life cycle of a servlet.
- This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection.

SERVLET CONFIGURATION

- Two ways of servlet configuration
  - XML Configuration — web.xml (Commonly used in Enterprise Applications)
    - Also called a deployment descriptor; the full list for what can be included is here:
    - https://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml. html
- Annotation (Easy for coding)

XML CONFIGURATION

- web.xml
  - <display-name> — The name of the project
  - <welcome-file-list>/<welcome-file> — The welcome-file element contains file name to use as a default welcome file, such as index.html
- <servlet> — Register servlets in the application
  - <servlet-name> — The name of servlet, which can be used by others
  - <servlet-class> — The location of the servlet
  - <load-on-startup> — This servlet should be loaded (instantiated and have its init() called) on the startup of the web application or not
    - Value: Must be Integers(Lower value takes the priority)
      - Negative — The container is free to load the servlet whenever it chooses
      - Positive or 0 — The container must load and initialize the servlet as the application is deployed

XML CONFIGURATION

- web.xml
  - <servlet-mapping> — Map the servlet to one URI
    - <servlet-name> — The name of the servlet which is defined at <servlet> tag
    - <url-pattern> — The URI mapping to the servlet
- <context-param> — The declaration of a web application's servlet context initialization parameters
  - What’s the difference between servletContext and servletConfig?
  - <param-name> — The param-name element contains the name of a parameter
  - <param-value> — The param-value element contains the value of a parameter

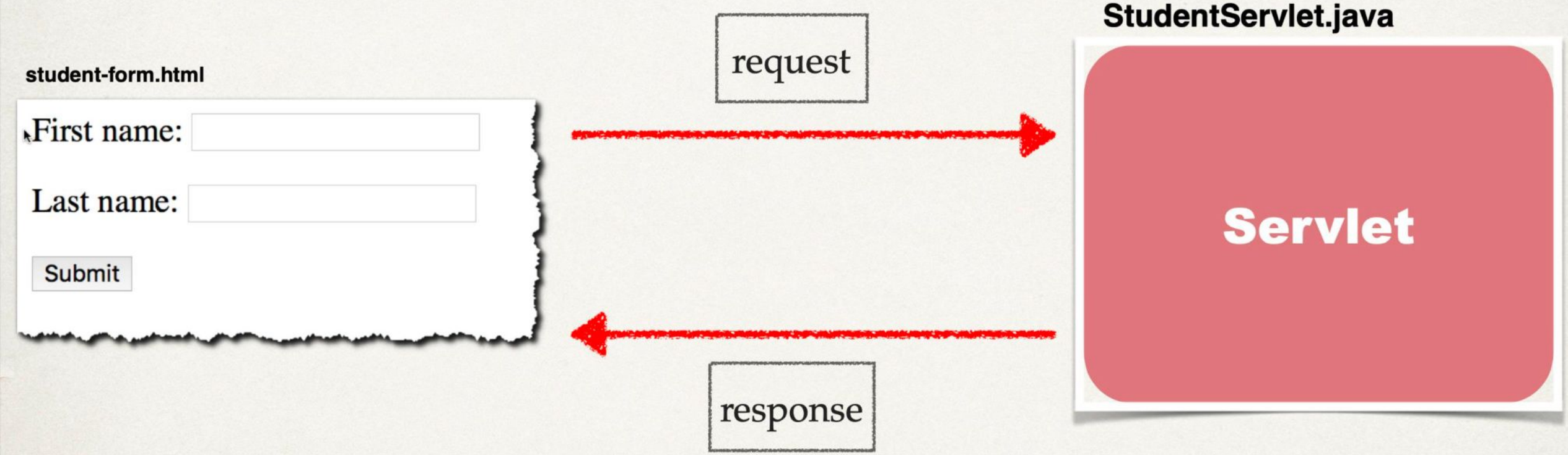
ANNOTATION CONFIGURATION

- @WebServlet() — on servlet class
  - name
  - urlPattern
  - loadOnStartUp • displayName
  - initParams

ACCESS SERVLET

- http(s)://<host: port>/<project-name>/<servlet-mapping>

INTERACT WITH HTML



INTERACT WITH HTML

```
<form action="StudentServlet" method="GET">

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

  Last name: <input type="text" name="lastName" />

  <input type="submit" value="Submit" />

</form>
```

First name:

Last name:

Submit

INTERACT WITH HTML

```
<form action="StudentServlet" method="GET">
   ...
</form>
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws ServletException, IOException {

    ...
}
```

INTERACT WITH HTML

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

Last name: <input type="text" name="lastName" />
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {

    ...
    out.println("The student is confirmed: " + request.getParameter("firstName")
            + " " + request.getParameter("lastName"));

}
```

ANY QUESTIONS?