* **J2SE(Java Platform, Standard Edition)**

Also known as Core Java, this is the most basic and standard version of Java.It’s the purest form of Java, a basic foundation for all other editions.

It consists of a wide variety of general purpose API’s (like java.lang, java.util) as well as many special purpose APIs

J2SE is mainly used to create applications for Desktop environment.

It consist all the basics of Java the language, variables, primitive data types, Arrays, Streams, Strings Java Database Connectivity(JDBC) and much more. This is the standard, from which all other editions came out, according to the needs of the time.

The famous JVM of Java, the heart of Java development, was also given by this edition only.It’s because of this feature, that Java has such a wide usage.

* **J2ME(Java Platform, Micro Edition)**

This version of Java is mainly concentrated for the applications running on embedded systems, mobiles and small devices.(which was a constraint before it’s development)

Constraints included limited processing power, battery limitation, small display etc.

Also, the J2ME apps help in using web compression technologies, which in turn, reduce network usage, and hence cheap internet accessibility.

J2ME uses many libraries and API’s of J2SE, as well as, many of it’s own.

The basic aim of this edition was to work on mobiles, wireless devices, set top boxes etc.

Old Nokia phones, which used Symbian OS, used this technology.

Most of the apps, developed for the phones(prior to smartphones era), were built on J2ME platform only(the .jar apps on Nokia app store).

* **J2EE(Java Platform, Enterprise Edition)**

The Enterprise version of Java has a much larger usage of Java, like development of web services, networking, server side scripting and other various web based applications.

J2EE is a community driven edition, i.e. there is a lot of continuous contributions from industry experts, Java developers and other open source organizations.

J2EE uses many components of J2SE, as well as, has many new features of it’s own like Servlets, JavaBeans, Java Message Services, adding a whole new functionalities to the language.

J2EE uses HTML, CSS, JavaScript etc., so as to create web pages and web services. It’s also one of  the most widely accepted web development standard.

There are also many languages like .net and php, which can do that work, but what distinguishes it from other languages is the versatility, compatibility and security features, which are not that much prominent in other languages.

**Standalone application**

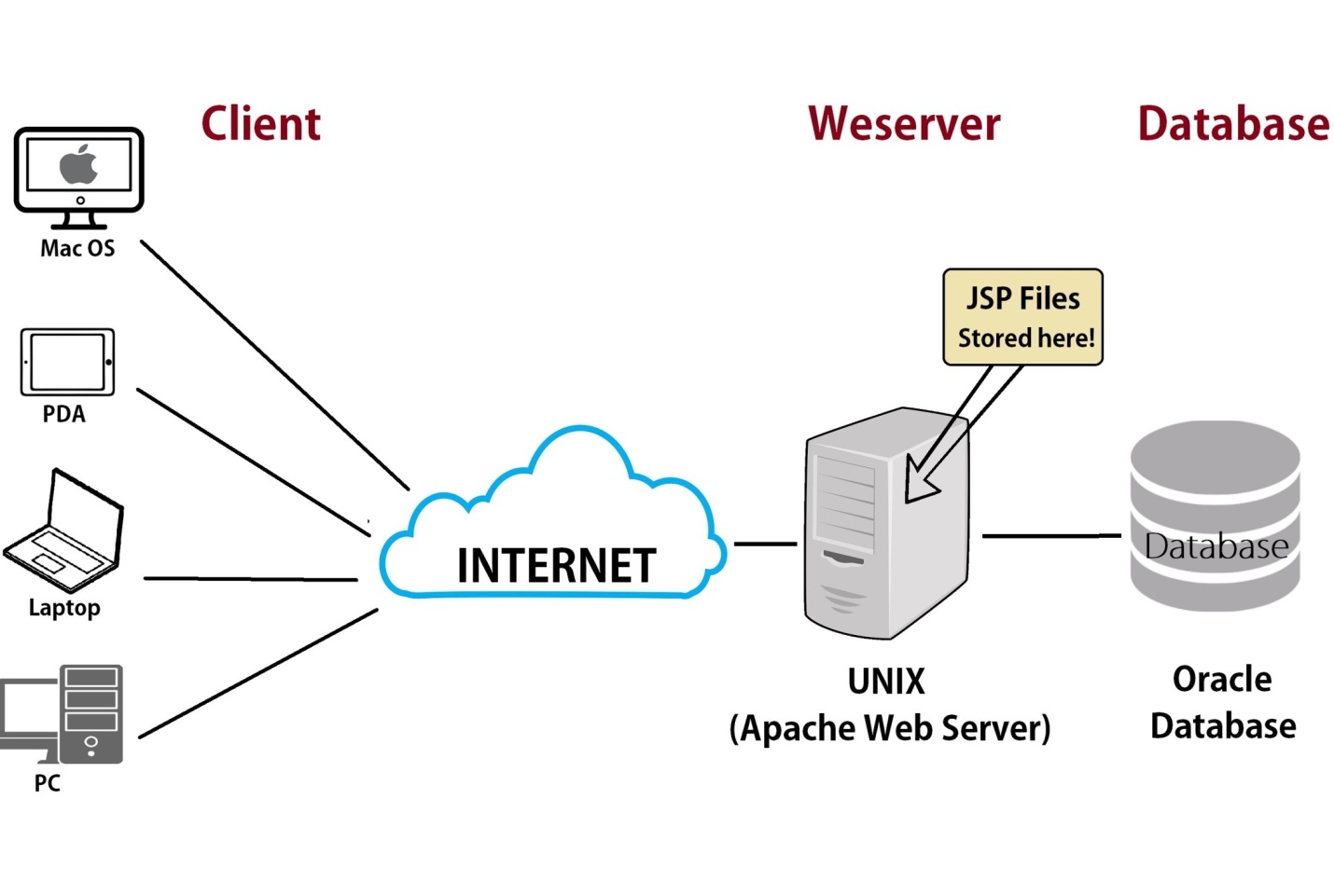
**Standalone applications** are traditional software that are installed on each client system. Essence Computing only develops platform-independent **applications**, so that the user can use any Operating System of their choice on the system

**Web Application**

A **Web application** (**Web app**) is an **application** program that is stored on a remote server and delivered over the Internet through a browser interface. **Web** services are **Web apps** by definition and many, although not all, websites contain **Web apps**.

**Web Server**

A Web [server](https://whatis.techtarget.com/definition/server) is a program that uses [HTTP](https://searchwindevelopment.techtarget.com/definition/HTTP) (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.



Application server

An application server is a type of server designed to install, operate and host applications and associated services for end users, IT services and organizations. It facilitates the hosting and delivery of high-end consumer or business applications, which are used by multiple and simultaneously connected local or remote users.

# JDBC

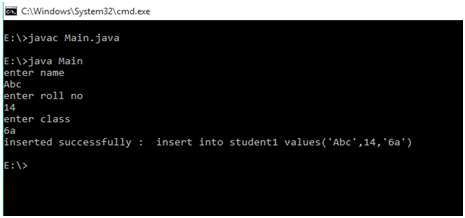
JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

To connect java application with the oracle database, we need to follow 5 following steps.

1. **Driver class:**The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:**The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
3. **Username:**The default username for the oracle database is **system**.
4. **Password:**It is the password given by the user at the time of installing the oracle database.

### Example to Connect Java Application with mysql database

1. **importjava.sql.\*;**
2. **importjava.util.\*;**
3. **class Main**
4. **{**
5. **public static void main(String a[])**
6. **{**
7. **//Creating the connection**
8. **String url = "jdbc:oracle:thin:@localhost:1521:xe";**
9. **String user = "system";**
10. **String pass = "12345";**
11. **//Entering the data**
12. **Scanner k = new Scanner(System.in);**
13. **System.out.println("enter name");**
14. **String name = k.next();**
15. **System.out.println("enter roll no");**
16. **int roll = k.nextInt();**
17. **System.out.println("enter class");**
18. **String cls = k.next();**
19. **//Inserting data using SQL query**
20. **String sql = "insert into student1 values('"+name+"',"+roll+",'"+cls+"')";**
21. **Connection con=null;**
22. **try**
23. **{**
24. **DriverManager.registerDriver(new oracle.jdbc.OracleDriver());**
25. **//Reference to connection interface**
26. **con = DriverManager.getConnection(url,user,pass);**
27. **Statement st = con.createStatement();**
28. **int m = st.executeUpdate(sql);**
29. **if (m == 1)**
30. **System.out.println("inserted successfully : "+sql);**
31. **else**
32. **System.out.println("insertion failed");**
33. **con.close();**
34. **}**
35. **catch(Exception ex)**
36. **{**
37. **System.err.println(ex);**
38. **}**
39. **}**
40. **}**



# Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

### Commonly used methods of Statement interface:

The important methods of Statement interface are as follows:

|  |
| --- |
| **1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet. |
| **2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc. |
| **3) public boolean execute(String sql):** is used to execute queries that may return multiple results. |
| **4) public int[] executeBatch():** is used to execute batch of commands. Example of Statement interface Let’s see the simple example of Statement interface to insert, update and delete the record. |

1. **import** java.sql.\*;
2. **class** FetchRecord{
3. **public** **static** **void** main(String args[])**throws** Exception{
4. Class.forName("oracle.jdbc.driver.OracleDriver");
5. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
6. Statement stmt=con.createStatement();
8. //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
9. //int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
10. **int** result=stmt.executeUpdate("delete from emp765 where id=33");
11. System.out.println(result+" records affected");
12. con.close();
13. }}

**next>><<pre**

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

### Example of PreparedStatement interface that inserts the record

First of all create table as given below:

1. create table emp(id number(10),name varchar2(50));

Now insert records in this table by the code given below:

1. **import** java.sql.\*;
2. **class** InsertPrepared{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
7. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
10. stmt.setInt(1,101);//1 specifies the first parameter in the query
11. stmt.setString(2,"Ratan");
13. **int** i=stmt.executeUpdate();
14. System.out.println(i+" records inserted");
16. con.close();
18. }**catch**(Exception e){ System.out.println(e);}
20. }
21. }

# Java CallableStatement Interface

CallableStatement interface is used to call the **stored procedures and functions**.

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

In this example, we are going to call the stored procedure INSERTR that receives id and name as the parameter and inserts it into the table user420. Note that you need to create the user420 table as well to run this application.

1. **import** java.sql.\*;
2. **public** **class** Proc {
3. **public** **static** **void** main(String[] args) **throws** Exception{
5. Class.forName("oracle.jdbc.driver.OracleDriver");
6. Connection con=DriverManager.getConnection(
7. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
9. CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
10. stmt.setInt(1,1011);
11. stmt.setString(2,"Amit");
12. stmt.execute();
14. System.out.println("success");
15. }
16. }

# Servlets

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

## What is a Servlet?

Servlet can be described in many ways, depending on the context.

* Servlet is a technology which is used to create a web application.
* Servlet is an API that provides many interfaces and classes including documentation.
* Servlet is an interface that must be implemented for creating any Servlet.
* Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
* Servlet is a web component that is deployed on the server to create a dynamic web page.



# Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



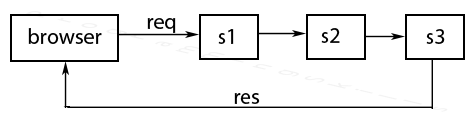
### Servlet Chaining:

If a client request is processed by group of servlets, then that servlets are known as servlet chaining or if the group of servlets process a single client request then those servlets are known as servlet chaining.

In order to process a client request by many number of servlets then we have two models, they are forward model and include model.

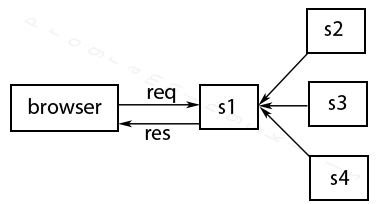
#### Forward model:

In this model when we forward a request to a group of servlets, finally we get the result of destination servlet as a response but not the result of intermediate servlets.



#### Include model:

If a single client request is passed to a servlet and that servlet makes use of other group of servlets to process a request by including the group of servlets into a single servlet.



In the above diagram client request goes to servlet s1 and s1 internally includes s2, s3 and s4 servlets and finally result of all these servlets given to the client by a source servlet s1.

**Note:** One servlet can include any number of servlets where as one servlet can forward to only one servlet at a time.

Introduction to JSP

* It stands for **Java Server Pages**.
* It is a server side technology.
* It is used for creating web application.
* It is used to create dynamic web content.
* In this JSP tags are used to insert JAVA code into HTML pages.
* It is an advanced version of Servlet Technology.
* It is a Web based technology helps us to create dynamic and platform independent web pages.
* In this, Java code can be inserted in HTML/ XML pages or both.
* JSP is first converted into servlet by JSP container before processing the client’s request

Syntax available in JSP are following

* 1. **Declaration Tag**:-It is used to declare variables.

**Syntax:-**

<%! Dec var %>

**Example:-**

<%! int var=10; %>

* 1. **Java Scriplets**:- It allows us to add any number of JAVA code, variables and expressions.

**Syntax:-**

<% java code %>

* 1. **JSP Expression**:- It evaluates and convert the expression to a string.

**Syntax:-**

<%= expression %>

**Example:-**

<% num1 = num1+num2 %>

* 1. **JAVA Comments**:- It contains the text that is added for information which has to be ignored.

**Syntax:-**

<% -- JSP Comments %>

## Paths Followed By JSP

The following are the paths followed by a JSP −

* Compilation
* Initialization
* Execution
* Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



# JSTL (JSP Standard Tag Library)

# The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

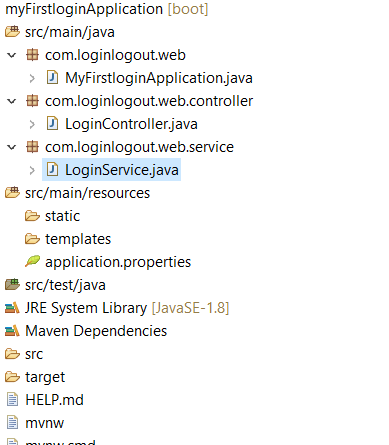
The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:

## JSTL Function Tags List

|  |  |
| --- | --- |
| **JSTL Functions** | **Description** |
| [fn:contains()](https://www.javatpoint.com/jstl-fn-contains-function) | It is used to test if an input string containing the specified substring in a program. |
| [fn:containsIgnoreCase()](https://www.javatpoint.com/jstl-fn-contains-ignorecase-function) | It is used to test if an input string contains the specified substring as a case insensitive way. |
| [fn:endsWith()](https://www.javatpoint.com/jstl-fn-endwidth-function) | It is used to test if an input string ends with the specified suffix. |
| [fn:escapeXml()](https://www.javatpoint.com/jstl-fn-escapexml-function) | It escapes the characters that would be interpreted as XML markup. |
| [fn:indexOf()](https://www.javatpoint.com/jstl-fn-indexof-function) | It returns an index within a string of first occurrence of a specified substring. |
| [fn:trim()](https://www.javatpoint.com/jstl-fn-trim-function) | It removes the blank spaces from both the ends of a string. |
| [fn:startsWith()](https://www.javatpoint.com/jstl-fn-startswith-function) | It is used for checking whether the given string is started with a particular string value. |
| [fn:split()](https://www.javatpoint.com/jstl-fn-split-function) | It splits the string into an array of substrings. |
| [fn:toLowerCase()](https://www.javatpoint.com/jstl-fn-tolowercase-function) | It converts all the characters of a string to lower case. |
| [fn:toUpperCase()](https://www.javatpoint.com/jstl-fn-touppercase-function) | It converts all the characters of a string to upper case. |
| [fn:substring()](https://www.javatpoint.com/jstl-fn-substring-function) | It returns the subset of a string according to the given start and end position. |
| [fn:substringAfter()](https://www.javatpoint.com/jstl-fn-substringafter-function) | It returns the subset of string after a specific substring. |
| [fn:substringBefore()](https://www.javatpoint.com/jstl-fn-substringbefore-function) | It returns the subset of string before a specific substring. |
| [fn:length()](https://www.javatpoint.com/jstl-fn-length-function) | It returns the number of characters inside a string, or the number of items in a collection. |
| [fn:replace()](https://www.javatpoint.com/jstl-fn-replace-function) | It replaces all the occurrence of a string with another string sequence. |

**Project in Spring boot**

**Project structure**

****

**MyApplication.java**

**package com.loginlogout.web;**

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication;**

**import org.springframework.context.annotation.ComponentScan;**

**@SpringBootApplication**

**@ComponentScan("com.loginlogout.web")**

**public class MyFirstloginApplication {**

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

**SpringApplication.run(MyFirstloginApplication.class, args);**

**}**

**}**

**LoginController.java**

**package com.loginlogout.web.controller;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.stereotype.Controller;**

**import org.springframework.ui.ModelMap;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestMethod;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.SessionAttributes;**

**import com.loginlogout.web.service.LoginService;**

**@Controller**

**@SessionAttributes("name")**

**public class LoginController {**

**@Autowired**

**LoginService service;**

**@RequestMapping(value="/login", method=RequestMethod.GET)**

**public String showLoginPage(ModelMap model) {**

**return"login";**

**}**

**@RequestMapping(value="/login",method=RequestMethod.POST)**

**public String showWelcomePage(ModelMap model,@RequestParam String name,@RequestParam String password) {**

**boolean isValidUser =service.validateUser(name,password);**

**if(!isValidUser) {**

**model.put("errorMessage", "InvalidCredentials");**

**return "login";**

**}**

**model.put("name", name);**

**model.put("password", password);**

**return "welcome";**

**}**

**}**

**LoginService.java**

**package** com.loginlogout.web.service;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** LoginService {

**public** **boolean** validateUser(String userid, String password) {

**return** userid.equalsIgnoreCase("hara")&&

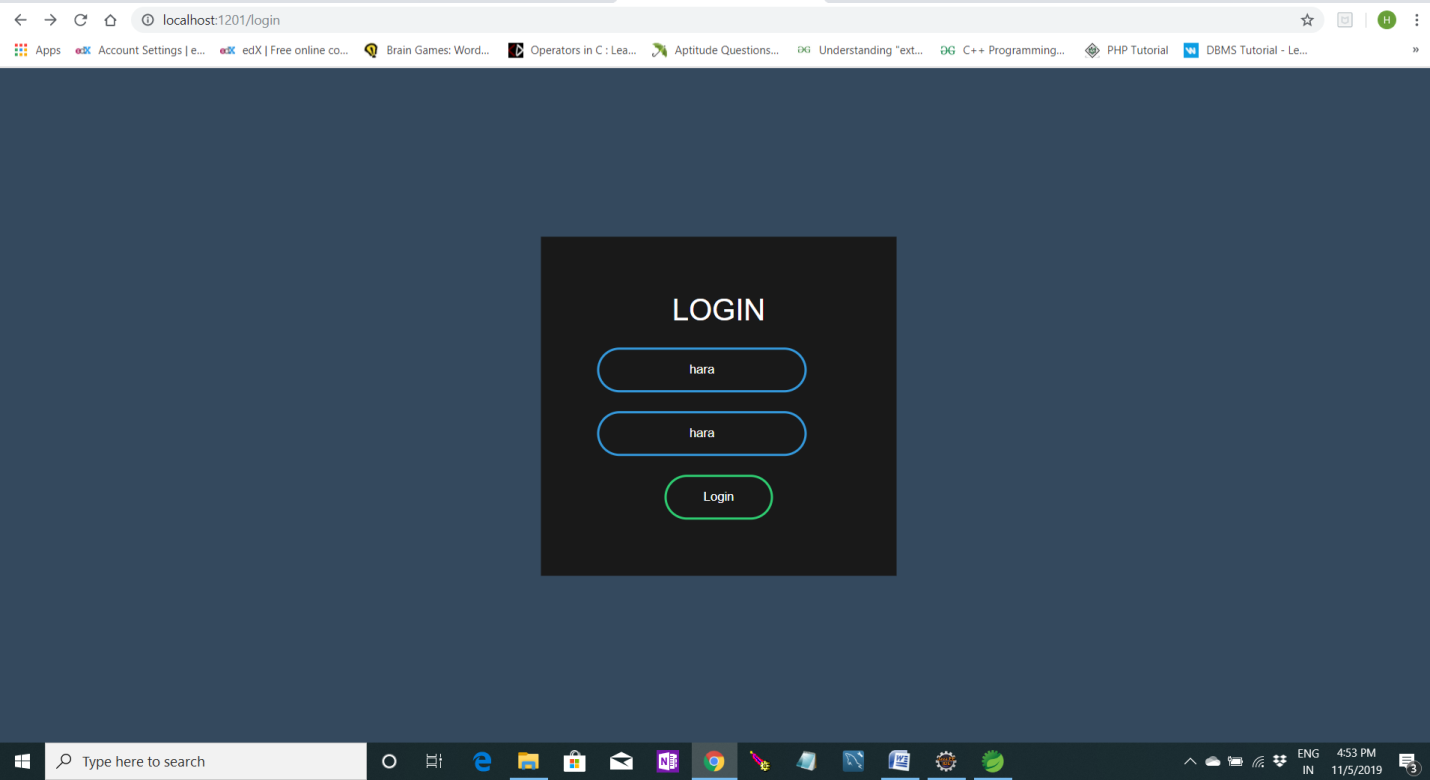
password.equalsIgnoreCase("hara");

}

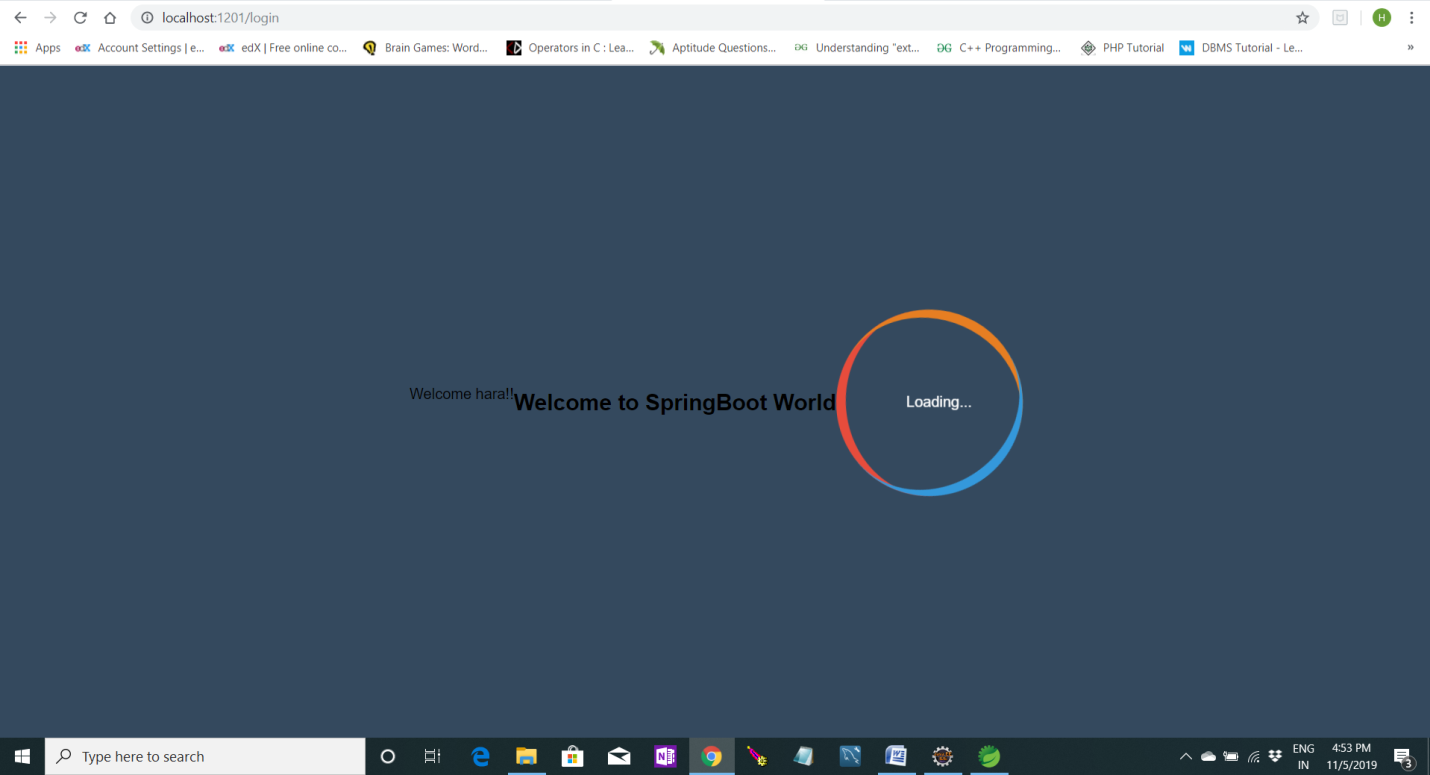
}

**Project output:**

**Before Login**

****

**After Login**

****