

RAJALAKSHMI ENGINEERING COLLEGE

AN AUTONOMOUS INSTITUTION

THANDALAM-602105

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE



LAB MANUAL

AI23431 WEB TECHNOLOGIES AND MOBILE APPLICATIONS

NAME OF THE STUDENT : HARINI S

REGISTER NUMBER : 2116231801049

YEAR/SEMESTER : II Year / III Semester

INDEX

EX.NO	EXERCISE TITLE	PAGE NO.
1.	HTML & CSS a) Create a web page to embed a map along with hot spot, frames & links. Create a web page using an embedded, external and inline CSS file.	
2.	Write JavaScript to validate the following fields of the Registration page. a) First Name (Name should contains alphabets and the length Should not be less than 6 characters). b) Password (Password should not be less than 6 characters length). c) E-mail id (should not contain any invalid and must follow the standard pattern name@domain.com) d) Mobile Number (Phone number should contain 10 digits only). e) Last Name and Address (should not be Empty).	
3.	Write a Servlet program that prints "Hello, World!" when accessed through a browser.	
4.	Create a web form that accepts a user's name and age. Write a Servlet to process the form data and display it back on the browser.	
5.	Write a Servlet to demonstrate the difference between HTTP GET and POST methods by creating a form and handling requests accordingly.	
6.	Write a Servlet to demonstrate session tracking using HttpSession. Implement a simple login system where the user's session is tracked.	
7.	Develop an Android application using controls like Button, TextView, EditText for designing a calculator having basic functionality like Addition, Subtraction, multiplication, and Division.	
8.	Develop an application to change the font and color of the text and display toast message when the user presses the button.	
9.	Develop a mobile application to send an email.	

EXP.NO: 1	Web Page Creation using HTML
DATE: 29/01/2025	

AIM:

To create a web page that embeds a map with hotspots, frames, and links. The web page should be designed using HTML, incorporating embedded, external, and inline CSS for styling. This exercise aims to enhance understanding of integrating multimedia elements like maps, creating interactive hotspots, and organizing content using frames while applying different CSS techniques for styling.

ALGORITHM:

Create the HTML structure:

- Set up the basic HTML document with `<!DOCTYPE html>`, `<html>`, `<head>`, and `<body>` tags.
- In the `<head>`, link the external CSS file (`<link>` tag), define the title, and include any metadata.
- In the `<body>`, create the necessary elements:
 - **Frames:** Use the `<frameset>` tag to create a frame layout (for older browsers, use `<iframe>` instead).
 - **Map:** Use the `` tag to display a map.
 - **Hotspots:** Create clickable regions using the `<map>` and `<area>` tags.
 - **Links:** Add hyperlinks (`<a>`) to navigate to different parts of the page or external sites.
 -

Create CSS Styles:

- **External CSS:** In a separate file, define styles such as colors, fonts, and layout for the entire page.
- **Inline CSS:** Use the style attribute within specific tags to apply styles (e.g., in `<div>` or `<p>` tags).

- **Embedded CSS:** Inside the <style> tag in the <head>, define styles for specific elements like the map, frames, and text.

Map with Hotspots:

- Use the <map> tag to create an image map.
- Define the <area> tag for each hotspot, specifying the coordinates (e.g., coords="x,y,x2,y2") for clickable areas on the map.

Frames:

- Use the <iframe> tag for embedding external content like a map or another webpage, with attributes such as src for the URL, width, and height.

Add Links:

- Inside the <body>, use the <a> tag to create links for navigation (for example, to link to specific parts of the page or external sites).

Test the Page:

- Test the webpage to check the functionality of frames, hotspots, and links. Ensure that the map hotspots are clickable and the linked content displays correctly.
- Ensure that the styles are applied properly using inline, embedded, and external CSS.

CODE:

Index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Interactive Chennai Map</title>
  <!-- Link to External CSS -->
  <link rel="stylesheet" href="style.css">

  <!-- Embedded CSS -->
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #eef;
    }

    .header {
      text-align: center;
      padding: 20px;
      background-color: #4CAF50;
```

```

        color: white;
    }

    .map-container {
        text-align: center;
        margin: 20px;
    }

    iframe {
        border: 2px solid #444;
        width: 600px;
        height: 400px;
    }

    .nav {
        text-align: center;
        margin: 15px;
    }
}
</style>
</head>
<body>

<div class="header">
    <h1 style="font-size: 2.5rem;">Explore Chennai</h1> <!-- Inline CSS -->
</div>

<div class="nav">
    <a href="https://en.wikipedia.org/wiki/Chennai" target="_blank">About Chennai</a> |
    <a href="https://www.google.com/maps/place/Chennai" target="_blank">View on
Google Maps</a>
</div>

<div class="map-container">
    <!-- Image with Map Hotspots -->
    

    <map name="chennaiMap">
        <area shape="rect" coords="100,80,200,150"
href="https://en.wikipedia.org/wiki/Marina_Beach" alt="Marina Beach"
target="iframeDisplay">
        <area shape="circle" coords="300,200,40"
href="https://en.wikipedia.org/wiki/Fort_St._George,_India" alt="Fort St. George"
target="iframeDisplay">
        <area shape="poly" coords="400,250,450,300,420,350,380,320"
href="https://en.wikipedia.org/wiki/Kapaleeshwarar_Temple" alt="Kapaleeshwarar
Temple" target="iframeDisplay">
    </map>

```

</div>

```
<div class="map-container">
  <iframe name="iframeDisplay" src="about:blank"></iframe>
</div>
```

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

style.css

```
a {
  color: darkblue;
  font-weight: bold;
  text-decoration: none;
}

a:hover {
  color: crimson;
  text-decoration: underline;
}
```

OUTPUT:


Explore Chennai - Map with Hotspots

[View Map](#) | [About Chennai](#) | [Wikipedia](#)

About Chennai

Chennai, formerly known as Madras, is the capital city of Tamil Nadu. It is known for its rich culture, beaches, and historical landmarks.

CHENNAI MAP



The map illustrates the geographical layout of Chennai, highlighting its coastal location and the division into administrative districts. The districts are color-coded for easy identification: North Chennai (red), Central Chennai (blue), South Chennai (orange), West Chennai (purple), East Chennai (green), North West Chennai (yellow), North East Chennai (pink), South West Chennai (brown), South East Chennai (light blue), and Central West Chennai (dark blue). The map also shows the coastline of Chennai and the surrounding areas.

CHENNAI MAP



RESULT:

Html Page is created with embedded Chennai map along with hot spot, frames & links.

EXP.NO: 2	Write JavaScript Programme to validate the following fields of the HTML Registration page.
DATE: 06/02/2025	

AIM:

To write JavaScript code to validate user input in a registration form. The validation will ensure that the data entered by the user meets specific criteria for each field. This will help in maintaining data integrity and prevent invalid submissions.

ALGORITHM:

1. Input:

- The user fills in the following fields: **First Name, Password, E-mail, Mobile Number, Last Name, and Address.**

2. Validation Steps:

1. Validate First Name:

- Check if the first name contains **only alphabets**.
- Ensure the length of the **First Name** is **at least 6 characters**.
- If any of these conditions fail, display an error message indicating the required input.

2. Validate Password:

- Check if the password is **at least 6 characters long**.
- If it is less than 6 characters, display an error message.

3. Validate E-mail:

- Use a **regular expression** to check if the email follows the standard pattern: name@domain.com.
- Ensure that the email contains **no invalid characters** and matches the required pattern.
- If it doesn't match, display an error message indicating an invalid email format.

4. Validate Mobile Number:

- Check if the mobile number contains **exactly 10 digits**.

- Ensure there are **no alphabets or special characters**.
- If the number is invalid, display an error message indicating the correct format.

5. **Validate Last Name:**

- Ensure the **Last Name** field is **not empty**.
- If it is empty, display an error message.

6. **Validate Address:**

- Ensure the **Address** field is **not empty**.
- If it is empty, display an error message.

CODE:

Registration.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Registration Form</title>
</style>
  body {
    font-family: 'Segoe UI', sans-serif;
    background: linear-gradient(to right, #74ebd5, #9face6);
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
  }

  .form-container {
    background-color: white;
    padding: 30px;
    border-radius: 15px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    width: 350px;
  }

  .form-container h2 {
    text-align: center;
    color: #333;
  }
```

```

.form-group {
  margin-bottom: 15px;
}

.form-group label {
  display: block;
  margin-bottom: 5px;
  color: #555;
}

.form-group input, .form-group textarea {
  width: 100%;
  padding: 8px;
  border: 1px solid #aaa;
  border-radius: 8px;
  box-sizing: border-box;
}

.form-group .error {
  color: red;
  font-size: 0.8em;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #4CAF50;
  border: none;
  color: white;
  font-size: 16px;
  border-radius: 8px;
  cursor: pointer;
}

button:hover {
  background-color: #45a049;
}
</style>
</head>
<body>

<div class="form-container">
  <h2>Registration Form</h2>
  <form onsubmit="return validateForm();">
    <div class="form-group">
      <label for="fname">First Name</label>
      <input type="text" id="fname">
      <div class="error" id="fnameError"></div>
    </div>
  </form>
</div>

```

```

</div>

<div class="form-group">
  <label for="lname">Last Name</label>
  <input type="text" id="lname">
  <div class="error" id="lnameError"></div>
</div>

<div class="form-group">
  <label for="email">E-mail</label>
  <input type="text" id="email">
  <div class="error" id="emailError"></div>
</div>

<div class="form-group">
  <label for="password">Password</label>
  <input type="password" id="password">
  <div class="error" id="passwordError"></div>
</div>

<div class="form-group">
  <label for="mobile">Mobile Number</label>
  <input type="text" id="mobile">
  <div class="error" id="mobileError"></div>
</div>

<div class="form-group">
  <label for="address">Address</label>
  <textarea id="address" rows="3"></textarea>
  <div class="error" id="addressError"></div>
</div>

  <button type="submit">Register</button>
</form>
</div>

<script>
function validateForm() {
  let isValid = true;

  // Clear previous errors
  document.querySelectorAll('.error').forEach(e => e.innerText = "");

  // First Name Validation
  const fname = document.getElementById("fname").value.trim();
  if (!/^[A-Za-z]{6,}$/.test(fname)) {
    document.getElementById("fnameError").innerText = "First name must be at least 6
letters and only alphabets.";
  }
}

```

```

        isValid = false;
    }

    // Last Name Validation
    const lname = document.getElementById("lname").value.trim();
    if (lname === "") {
        document.getElementById("lnameError").innerText = "Last name cannot be empty.";
        isValid = false;
    }

    // Email Validation
    const email = document.getElementById("email").value.trim();
    const emailPattern = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-z]{2,}$/;
    if (!emailPattern.test(email)) {
        document.getElementById("emailError").innerText = "Please enter a valid email (e.g., name@domain.com).";
        isValid = false;
    }

    // Password Validation
    const password = document.getElementById("password").value.trim();
    if (password.length < 6) {
        document.getElementById("passwordError").innerText = "Password must be at least 6 characters.";
        isValid = false;
    }

    // Mobile Number Validation
    const mobile = document.getElementById("mobile").value.trim();
    if (!/^d{10}$/.test(mobile)) {
        document.getElementById("mobileError").innerText = "Mobile number must be exactly 10 digits.";
        isValid = false;
    }

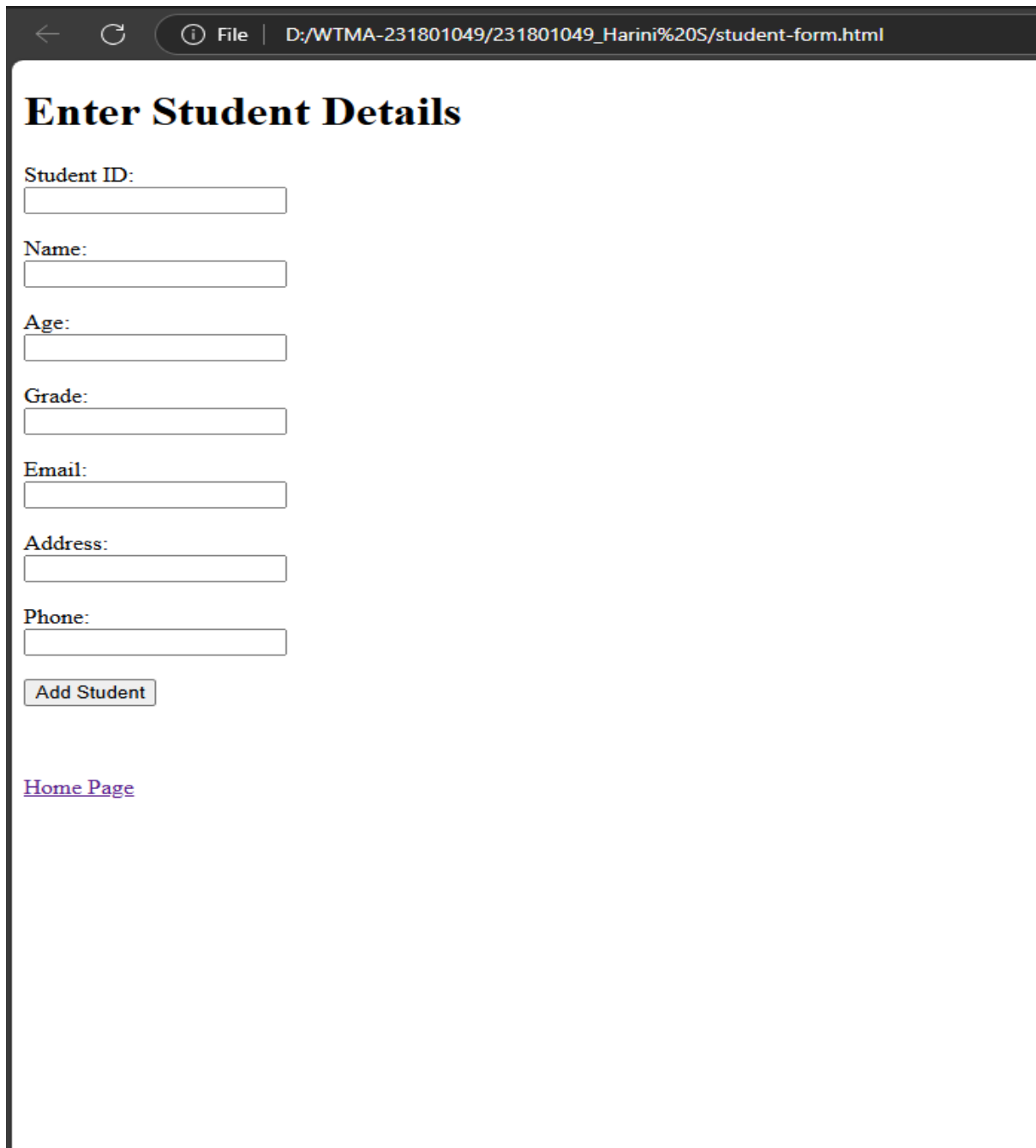
    // Address Validation
    const address = document.getElementById("address").value.trim();
    if (address === "") {
        document.getElementById("addressError").innerText = "Address cannot be empty.";
        isValid = false;
    }

    return isValid;
}
</script>

</body>
</html>

```

Output:



The screenshot shows a web browser window with the address bar displaying "D:/WTMA-231801049/231801049_Harini%20S/student-form.html". The page title is "Enter Student Details". The form contains the following fields and controls:

- Student ID:
- Name:
- Age:
- Grade:
- Email:
- Address:
- Phone:
-
- [Home Page](#)

RESULT:

HTML page with Form is created and is validated using Javascript.

EXP.NO: 3	Write a Servlet program that prints "Hello, World!" when accessed through a browser.
DATE:20/02/2025	

AIM:

To write a basic Servlet program that demonstrates the use of Java Servlets by printing "Hello, World!" when accessed through a browser.

ALGORITHM:

1. Input:

- The user makes an HTTP request (usually by entering a URL) through a browser.

2. Servlet Program Flow:

1. Create a Servlet Class:

- Define a Java class that extends the `HttpServlet` class.
- Override the `doGet()` method, which handles HTTP GET requests from the client (browser).

2. Handle HTTP Request:

- In the `doGet()` method:
 - Get the **PrintWriter** object to write the response to the browser.
 - Set the response content type to **text/html**.
 - Write the **"Hello, World!"** message to the response.

3. Compile the Servlet Class:

- Compile the Java Servlet file using the `javac` command to generate a `.class` file.

4. Deploy the Servlet:

- Place the compiled servlet class file in the appropriate directory of the web application (typically in the `/WEB-INF/classes` folder).
- Configure the servlet in the `web.xml` file of the web application or use annotations.

5. Access the Servlet via Browser:

- Start the **Apache Tomcat** or another servlet container.
- Access the servlet by entering the appropriate URL (e.g., `http://localhost:8080/your-webapp/hello`).

CODE:

HelloWorldServlet.java:

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {
```

```

        private static final long serialVersionUID = 1L;

        public HelloWorldServlet() {
            super();
        }

        @Override
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            response.setContentType("text/html");
            response.getWriter().println("<html><body><h1>Hello,
World!</h1></body></html>");
        }

        @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            doGet(request, response);
        }
    }

```

Output:



RESULT:

The browser will display the message "Hello, World!" returned by the Servlet.

EXP.NO: 4	Create a web form that accepts a user's name and age. Write a Servlet to process the form data and display it back on the browser.
DATE:26/02/2025	

AIM:

To create a simple web application that includes a form to accept a user's name and age, and then use a Servlet to process the submitted data. The Servlet will retrieve the form data, process it, and display the values back on the browser.

ALGORITHM:

1. Input:

- The user fills out a web form with their **name** and **age** and submits it.

2. Form Handling and Servlet Processing:

1. Create the Web Form:

- Create an HTML form with **two input fields**: one for the user's **name** and one for the user's **age**.
- Add a **submit button** to the form.
- Set the form's action attribute to the URL of the **Servlet** and the method to POST (or GET).

2. Create the Servlet Class:

- Write a Servlet that processes the **form data** submitted by the user.
- The Servlet will override the doPost() (or doGet() depending on the form method) to handle form submissions.
- The Servlet will **retrieve the name and age** parameters from the request object using request.getParameter("name") and request.getParameter("age").

3. Process the Data in the Servlet:

- Extract the name and age values submitted from the form.
- **Validate the input** (e.g., check if the age is a valid number).
- Prepare a **response** that will display the user's **name** and **age** back on the browser.

4. Send Response to the Browser:

- The Servlet will send a **response** to the browser containing a message like:
 - "Hello, [name]! You are [age] years old."
- The response will be displayed to the user in the browser.

5. Deploy the Servlet:

- Compile the Servlet and deploy it to a **Servlet container** (like **Apache Tomcat**).
- Ensure the web.xml file or Servlet annotations are used to map the Servlet to the URL defined in the form's action attribute.

6. Test the Application:

- Open the web browser, fill in the form with the name and age, and submit it.
- The Servlet will process the data and display the message back to the user in the browser.

CODE:

Index.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>User Info Form</title>
</head>
<body>
  <h2>Enter Your Details</h2>
  <form action="UserDataServlet" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

    <label for="age">Age:</label>
    <input type="number" id="age" name="age" required><br><br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

UserDataServlet.java:

```
package com.example.servlet;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
```

```

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/UserDataServlet")
public class UserDataServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

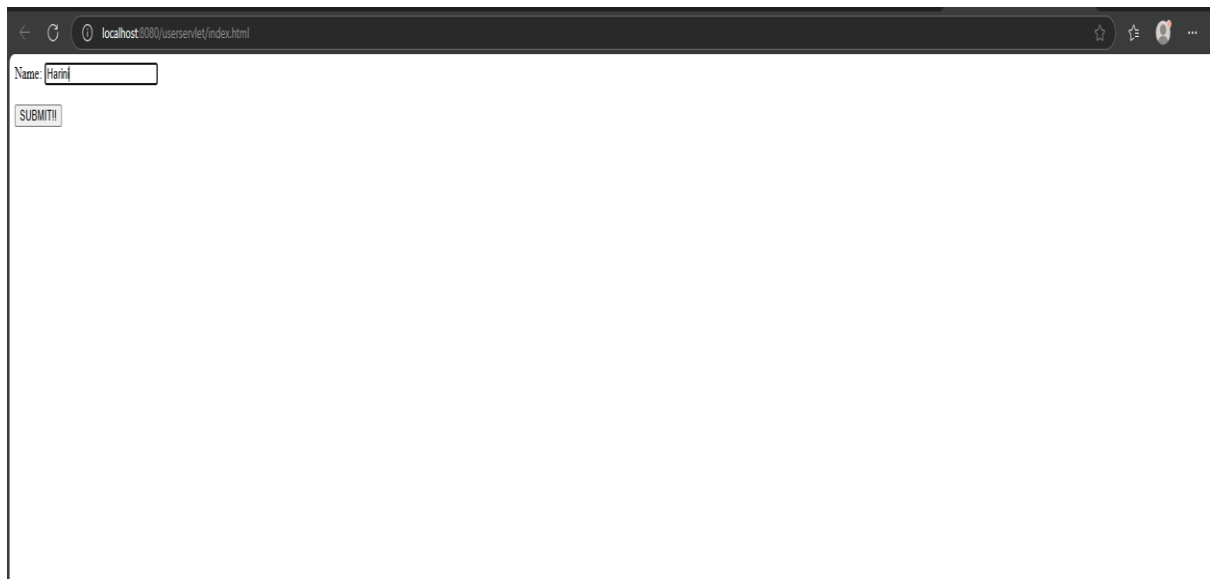
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("name");
        String ageStr = request.getParameter("age");
        try {
            int age = Integer.parseInt(ageStr); // Convert age to integer
            // Generate the response message
            out.println("<html><body>");
            out.println("<h1>Hello, " + name + "!</h1>");
            out.println("<p>You are " + age + " years old.</p>");
            out.println("</body></html>");
        } catch (NumberFormatException e) {

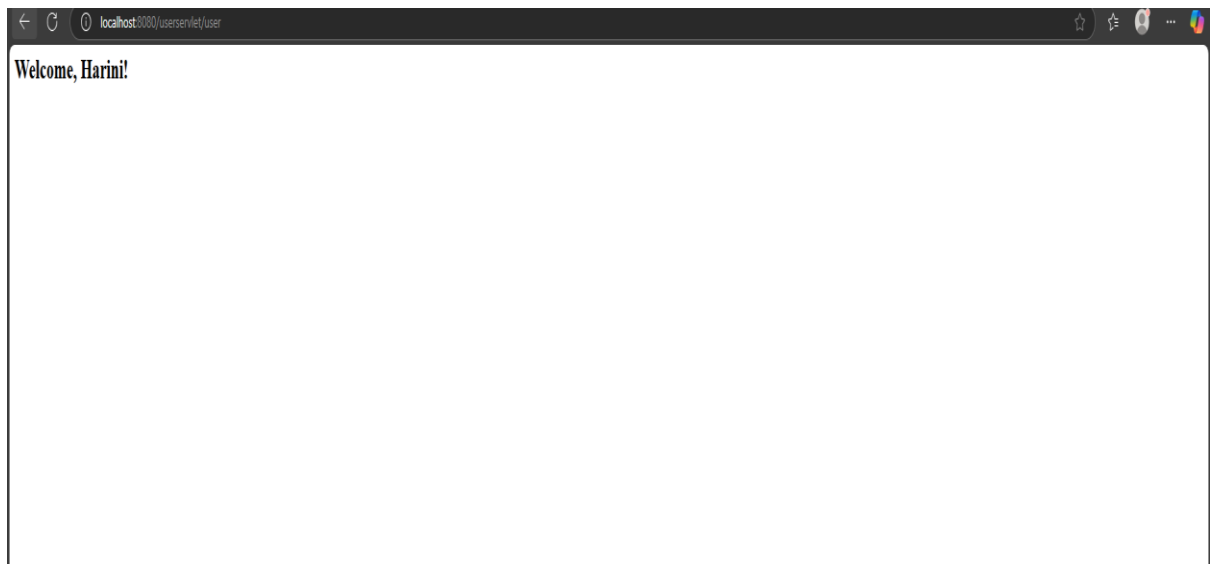
            out.println("<html><body>");
            out.println("<h2>Invalid age entered. Please enter a valid number.</h2>");
            out.println("</body></html>");
        }
    }
}

```

Output:



A screenshot of a web browser window. The address bar shows 'localhost:3080/userservlet/index.html'. The page content includes a form with a label 'Name:' followed by a text input field containing the text 'Harini'. Below the input field is a button labeled 'SUBMIT'.



A screenshot of a web browser window. The address bar shows 'localhost:3080/userservlet/user'. The page content displays the text 'Welcome, Harini!'.

RESULT:

A Servlet is created to process the HTML form data and display it back on the browser.

EXP.NO: 5	Write a Servlet to demonstrate the difference between HTTP GET and POST methods by creating a form and handling requests accordingly.
DATE:05/03/2025	

AIM:

To demonstrate the difference between the HTTP GET and POST methods by creating a form in an HTML page and handling the form submissions using two different HTTP methods in a Java Servlet.

ALGORITHM:

1. Input:

- The user fills out a web form with their **name** and **email**.
- The form will use both **GET** and **POST** methods for data submission.

2. Steps for Creating and Handling the HTTP GET and POST Requests:

1. Create an HTML Form:

- Design an HTML form with two input fields: one for the **name** and one for the **email**.
- Create two buttons to allow the user to submit the form using **GET** or **POST** methods.
- Use the action attribute to define the **Servlet URL** and specify the form method (either GET or POST).

2. Create the Servlet to Handle GET and POST Requests:

- Write a Java Servlet class that will handle **both GET and POST requests**.
- The servlet will have two methods:
 - doGet() to handle the GET requests.
 - doPost() to handle the POST requests.
- In both methods, retrieve the form data using request.getParameter().
- In the doGet() method, process and display the data in the URL.
- In the doPost() method, process and display the data in the HTTP request body.

3. Handle GET Requests in the Servlet:

- In the `doGet()` method, use `request.getParameter()` to retrieve the name and email submitted via GET.
- Output the data as a response, including the **name** and **email** in the URL (visible to the user).

4. Handle POST Requests in the Servlet:

- In the `doPost()` method, use `request.getParameter()` to retrieve the name and email submitted via POST.
- Output the data as a response, displaying the **name** and **email** without exposing them in the URL (more secure).

5. Return the Output:

- The Servlet should return an HTML page displaying the **name** and **email** entered by the user.
- For the **GET method**, the output should be visible in the URL.
- For the **POST method**, the output should be visible in the response body, not the URL.

6. Deploy and Test the Servlet:

- Compile the Servlet class and deploy it to a **Servlet container** (like **Apache Tomcat**).
- Test the form using **GET** and **POST** methods to see the difference in data submission.

7. Observe and Compare GET and POST Results:

- Test the **GET method**: The form data is sent in the URL.
- Test the **POST method**: The form data is sent in the HTTP request body, not visible in the URL.

CODE:

Index.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>GET vs POST Form</title>
</head>
<body>
  <h2>Enter Your Details</h2>

  <!-- Form for GET method -->
  <form action="FormServlet" method="get">
    <h3>GET Method</h3>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>

    <input type="submit" value="Submit using GET">
  </form>

  <br><br>

  <!-- Form for POST method -->
  <form action="FormServlet" method="post">
    <h3>POST Method</h3>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>

    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>

    <input type="submit" value="Submit using POST">
  </form>
</body>
</html>
```

FormServlet.java:

```
package com.example.servlet;
```

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
```

```
@WebServlet("/FormServlet")
public class FormServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // Set content type for the response
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Retrieve form data using GET method
    String name = request.getParameter("name");
    String email = request.getParameter("email");

    // Output the GET request data (visible in URL)
    out.println("<html><body>");
    out.println("<h2>GET Method</h2>");
    out.println("<p>Name: " + name + "</p>");
    out.println("<p>Email: " + email + "</p>");
    out.println("<p>Data is visible in the URL.</p>");
    out.println("</body></html>");
}

```

```

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Set content type for the response
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Retrieve form data using POST method
    String name = request.getParameter("name");
    String email = request.getParameter("email");

    // Output the POST request data (not visible in URL)
    out.println("<html><body>");
    out.println("<h2>POST Method</h2>");
    out.println("<p>Name: " + name + "</p>");
    out.println("<p>Email: " + email + "</p>");
    out.println("<p>Data is NOT visible in the URL.</p>");
    out.println("</body></html>");
}
}

```

Output:

The image displays two browser screenshots of a web application. The top screenshot shows the initial form titled "Enter Your Details" with sections for "GET Method" and "POST Method". The "GET Method" section has input fields for "Name" (containing "Hannah") and "Email" (containing "231801047@rajalakshmi.edu"), followed by a "Submit using GET" button. The "POST Method" section has empty input fields for "Name" and "Email", followed by a "Submit using POST" button. The bottom screenshot shows the result after a GET submission, with the "GET Method" section displaying the submitted values and a message: "Data is visible in the URL...". Both screenshots include a Windows watermark in the bottom right corner.

Enter Your Details

GET Method

Name:

Email:

POST Method

Name:

Email:

Activate Windows
Go to Settings to activate Windows.

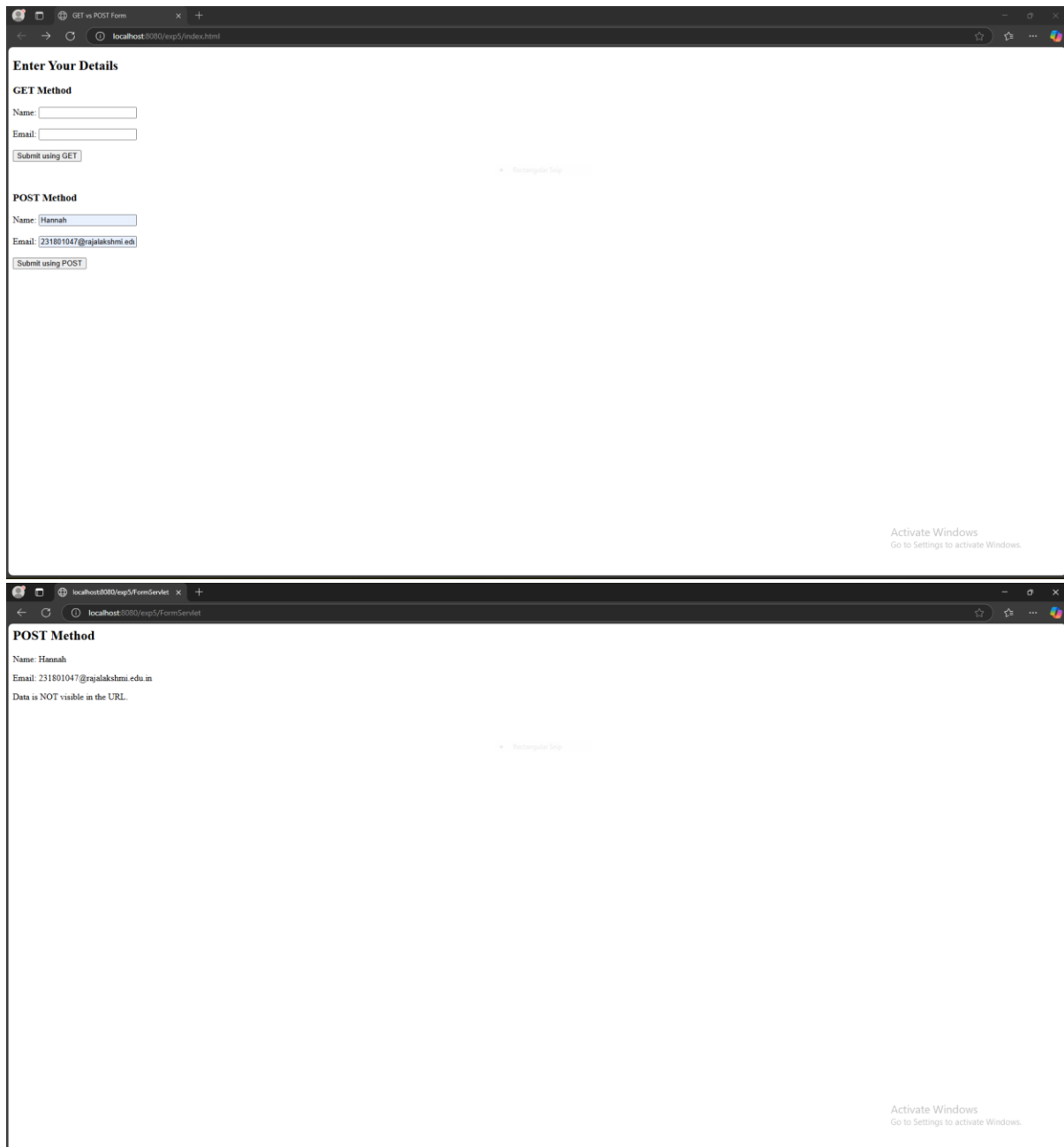
GET Method

Name: Hannah

Email: 231801047@rajalakshmi.edu.in

Data is visible in the URL...

Activate Windows
Go to Settings to activate Windows.



RESULT:

A Servlet is created to demonstrate the difference between HTTP GET and POST methods by creating a form and handling requests accordingly.

EXP.NO: 6	Write a Servlet to demonstrate session tracking using HttpSession.
DATE:26/03/2025	

AIM:

To demonstrate session tracking using HttpSession in a Servlet-based web application. The exercise involves implementing a simple login system where a user can log in with their username and password, and their session will be tracked throughout their interaction with the web application.

ALGORITHM:

1. Input:

- The user will enter their **username** and **password** in a **login form**.
- The servlet will validate the credentials and create an **HTTP session** for the user if authentication is successful.

2. Steps for Implementing Session Tracking with HttpSession:

1. Create the Login Form (HTML):

- Create an HTML form with two input fields for **username** and **password**.
- Provide a **submit button** to send the data to the servlet for authentication.

2. Create the Login Servlet:

- Write a Java servlet class to handle **login** requests.
- The servlet will:
 - Retrieve the **username** and **password** from the form.
 - Authenticate the user by comparing the credentials to a stored value (e.g., hardcoded or from a database).
 - Create a new **HTTP session** for the user if the credentials are valid.
 - Set session attributes such as the user's **username** to track the session.

3. Validate User Credentials:

- In the servlet, check if the **username** and **password** match the expected values.

- If valid, create a session and store the **username** in the session.
- If invalid, redirect the user back to the login page with an error message.

4. Create and Track User Session:

- Upon successful login, use the HttpSession object to store the session data.
- Use the session.setAttribute("username", username) method to save the **username**.
- Optionally, set session timeout using session.setMaxInactiveInterval(time_in_seconds) to manage session expiration.

5. Redirect to a Protected Page:

- After a successful login, the user will be redirected to a **protected page** (e.g., a dashboard or home page).
- The protected page should check if the user is authenticated by checking the **session attribute**.

6. Logout and Session Invalidation:

- Provide a **logout feature** that invalidates the session using session.invalidate() when the user logs out.
- After logout, redirect the user back to the login page.

7. Session Expiration:

- If the session expires or the user manually logs out, the session will be invalidated, and the user will be redirected to the login page.

CODE:

Login.jsp:

```
<!DOCTYPE html>
<html>
<head>
  <title>Login Form</title>
</head>
<body>
  <h2>Login</h2>
  <form action="LoginServlet" method="post">
    Username: <input type="text" name="username" required><br><br>
    Password: <input type="password" name="password" required><br><br>
    <input type="submit" value="Login">
  </form>
  <br>
  <p style="color:red;">${errorMessage}</p> <!-- Error message if invalid
credentials -->
```

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

LoginServlet.java:

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

import java.io.IOException;

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Hardcoded username and password for demonstration
    private static final String VALID_USERNAME = "validUser";
    private static final String VALID_PASSWORD = "validPassword";

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // Retrieve username and password from form
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Validate credentials
        if (VALID_USERNAME.equals(username) && VALID_PASSWORD.equals(password)) {
            // Create a session and store the username
            HttpSession session = request.getSession();
            session.setAttribute("username", username);

            // Redirect to the protected page (e.g., dashboard)
            response.sendRedirect("welcome.jsp");
        } else {
            // Invalid credentials, return an error message
            request.setAttribute("errorMessage", "Invalid username or password.");
            request.getRequestDispatcher("login.jsp").forward(request, response);
        }
    }
}
```

Welcome.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ page session="true" %>
```

```

<!DOCTYPE html>
<html>
<head>
    <title>Welcome Page</title>
</head>
<body>
    <h2>Welcome, ${sessionScope.username}!</h2>
    <form action="LogoutServlet" method="post">
        <input type="submit" value="Logout">
    </form>
</body>
</html>

```

LogoutServlet.java:

```
package com.example.servlet;
```

```

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

```

```
import java.io.IOException;
```

```
@WebServlet("/LogoutServlet")
```

```
public class LogoutServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
```

```

    // Use doGet() method for logout
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        // Invalidate the session
        HttpSession session = request.getSession();
        session.invalidate();

        // Redirect to login page
        response.sendRedirect("login.jsp");
    }

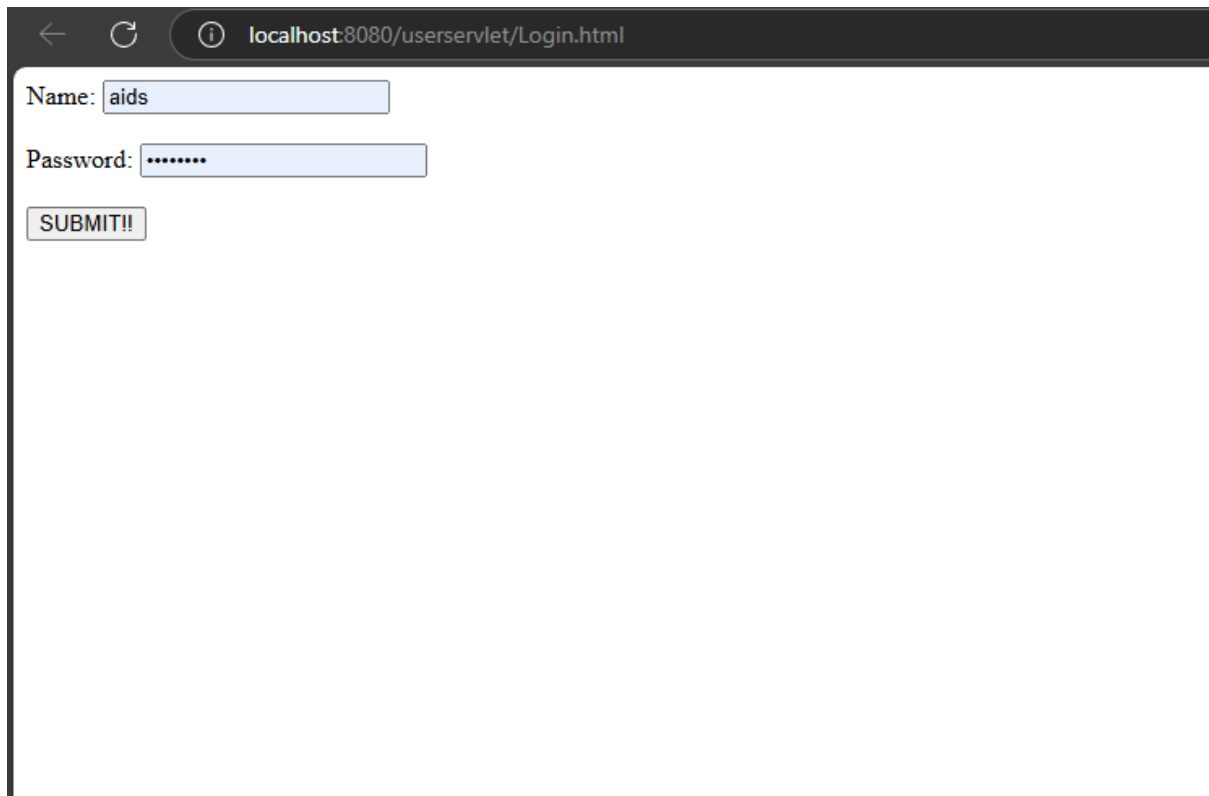
```

```

    // Optionally, if you want to handle POST requests as well, you can forward them to doGet()
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response); // Forward POST requests to doGet
    }
}

```

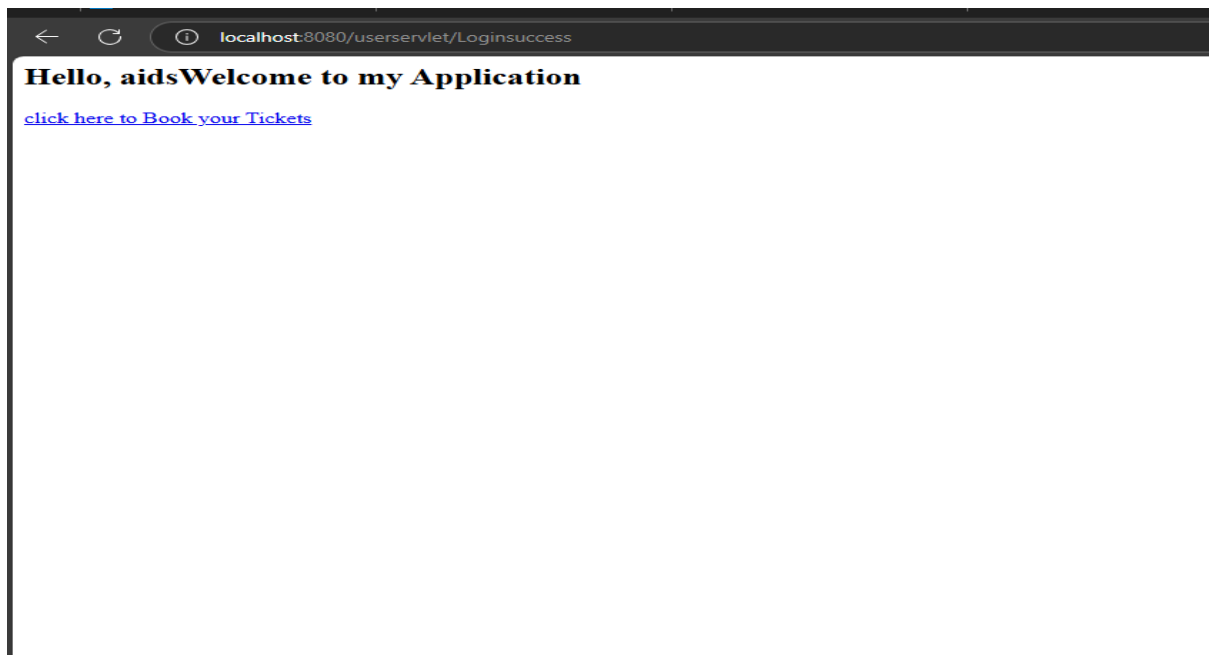
Output:



← ↻ ⓘ localhost:8080/userservlet/Login.html

Name:

Password:



RESULT:

A simple login system is created where the user's session is tracked by a servlet to demonstrate session tracking using HttpSession

EXP.NO: 7	Develop an Android Calculator Application using Android Studio
DATE:09/04/2025	

AIM:

To develop an Android application that implements a basic calculator using Android controls such as Button, TextView, and EditText. The calculator will support basic mathematical operations like Addition, Subtraction, Multiplication, and Division.

ALGORITHM:

1. Initialize the Android Application:

- Set up a new Android project in **Android Studio**.
- Create a **MainActivity** Java or Kotlin file for the main screen of the application.

2. Design the User Interface (UI):

- Create the following UI components in the **XML layout** file:
 - **EditText** for input fields where the user will enter numbers (two EditTexts for number1 and number2).
 - **Buttons** for each arithmetic operation (Addition, Subtraction, Multiplication, Division).
 - **TextView** to display the result of the operation.

3. Declare Variables:

- Create variables for the **EditText** fields, **Buttons**, and **TextView**.
- Define variables to hold the input numbers and the result (e.g., num1, num2, result).

4. Get Input from the User:

- Get the values entered by the user in **EditText** fields using `getText().toString()` and convert them to numerical values (like `Integer.parseInt()` or `Double.parseDouble()`).

5. Set Event Listeners for Buttons:

- For each button (Addition, Subtraction, Multiplication, and Division), set an **OnClickListener**.

- Inside the **onClick()** method for each button, perform the respective arithmetic operation:
 - **Addition:** $\text{result} = \text{num1} + \text{num2}$
 - **Subtraction:** $\text{result} = \text{num1} - \text{num2}$
 - **Multiplication:** $\text{result} = \text{num1} * \text{num2}$
 - **Division:** $\text{result} = \text{num1} / \text{num2}$ (add validation to prevent division by zero).

6. Perform Arithmetic Operation:

- Check which button was clicked (using `view.getId()`), and based on the button, perform the respective arithmetic operation.
- **Handle Division by Zero:** Check if `num2` is 0 before performing division and show an appropriate error message if true.

7. Display the Result:

- Use **TextView** to display the result of the operation.
- Convert the result to a string and set it using `setText()` method on the **TextView**.

8. Validate User Inputs:

- Before performing any operation, validate the input to ensure the user has entered valid numbers (not empty or non-numeric characters).

9. Handle Edge Cases:

- Ensure the app handles cases where users input invalid values or attempt division by zero. Display appropriate messages for errors.

10. Finish the Operation:

- Once the result is displayed, the user can either perform another calculation or exit the app.

Code

Step 1: Create the Project

- **Template:** Empty Views Activity
- **Project name** – CalculatorApp
- **Package** - com.example.calculatorapp
- **Language:** Java

- **Minimum SDK** – API 24
- **Build system:** Groovy DSL

Step 2: Under res → layout → Updated activity_main.xml using **ConstraintLayout**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <!-- First Number Input -->
    <EditText
        android:id="@+id/editTextNumber1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"
        android:hint="Enter first number"
        android:textSize="18sp"
        android:textColorHint="#000000"
        android:typeface="monospace"
        android:textStyle="bold"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_margin="16dp" />

    <!-- Second Number Input -->
    <EditText
        android:id="@+id/editTextNumber2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:inputType="numberDecimal"
        android:hint="Enter second number"
        android:textSize="18sp"
        android:textColorHint="#000000"
        android:typeface="monospace"
        android:textStyle="bold"
        app:layout_constraintTop_toBottomOf="@id/editTextNumber1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_margin="16dp" />
```

```

<!-- Row of Operation Buttons -->
<LinearLayout
    android:id="@+id/buttonRow"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="4"
    android:layout_margin="16dp"
    app:layout_constraintTop_toBottomOf="@id/editTextNumber2"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent">

    <Button
        android:id="@+id/buttonAdd"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Add" />

    <Button
        android:id="@+id/buttonSubtract"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Subtract" />

    <Button
        android:id="@+id/buttonMultiply"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Multiply" />

    <Button
        android:id="@+id/buttonDivide"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Divide" />
</LinearLayout>

<!-- Clear Button -->
<Button

```

```

        android:id="@+id/buttonClear"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clear"
        app:layout_constraintTop_toBottomOf="@id/buttonRow"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="12dp"/>

<!-- Result TextView -->
<TextView
    android:id="@+id/textViewResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Result will appear here"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintTop_toBottomOf="@id/buttonClear"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="24dp" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

Step 3 : Update the MainActivity.java file with this content

```

package com.example.calculatorapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

```

```
EditText editTextNumber1, editTextNumber2;  
Button buttonAdd, buttonSubtract, buttonMultiply, buttonDivide, buttonClear;  
TextView textViewResult;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    // Link XML elements with Java code  
    editTextNumber1 = findViewById(R.id.editTextNumber1);  
    editTextNumber2 = findViewById(R.id.editTextNumber2);  
    buttonAdd = findViewById(R.id.buttonAdd);  
    buttonSubtract = findViewById(R.id.buttonSubtract);  
    buttonMultiply = findViewById(R.id.buttonMultiply);  
    buttonDivide = findViewById(R.id.buttonDivide);  
    buttonClear = findViewById(R.id.buttonClear);  
    textViewResult = findViewById(R.id.textViewResult);
```

```
    // Add button logic  
    buttonAdd.setOnClickListener(view -> calculate('+'));
```

```
    // Subtract button logic  
    buttonSubtract.setOnClickListener(view -> calculate('-'));
```

```
    // Multiply button logic  
    buttonMultiply.setOnClickListener(view -> calculate('*'));
```

```
    // Divide button logic  
    buttonDivide.setOnClickListener(view -> calculate('/'));
```

```
    // Clear button logic  
    buttonClear.setOnClickListener(view -> {  
        editTextNumber1.setText("");  
        editTextNumber2.setText("");  
        textViewResult.setText("Result will appear here");  
    });  
}
```

```
private void calculate(char operator) {  
    String input1 = editTextNumber1.getText().toString();  
    String input2 = editTextNumber2.getText().toString();  
  
    if (input1.isEmpty() || input2.isEmpty()) {
```

```

        textViewResult.setText("Please enter both numbers.");
        return;
    }

    try {
        double num1 = Double.parseDouble(input1);
        double num2 = Double.parseDouble(input2);
        double result = 0;

        switch (operator) {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 * num2;
                break;
            case '/':
                if (num2 == 0) {
                    textViewResult.setText("Cannot divide by zero");
                    return;
                }
                result = num1 / num2;
                break;
        }

        textViewResult.setText("Result: " + result);
    } catch (NumberFormatException e) {
        textViewResult.setText("Invalid input");
    }
}

```

Step 4 – Under Gradle Scripts section → update the build.gradle(Module:app)

```

plugins {
    alias(libs.plugins.androidApplication)
}

android {
    namespace 'com.example.calculatorapp'
    compileSdk 34
}

```

```

defaultConfig {
    applicationId "com.example.calculatorapp"
    minSdk 26
    targetSdk 34
    versionCode 1
    versionName "1.0"

    testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
}

buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
    }
}

compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}

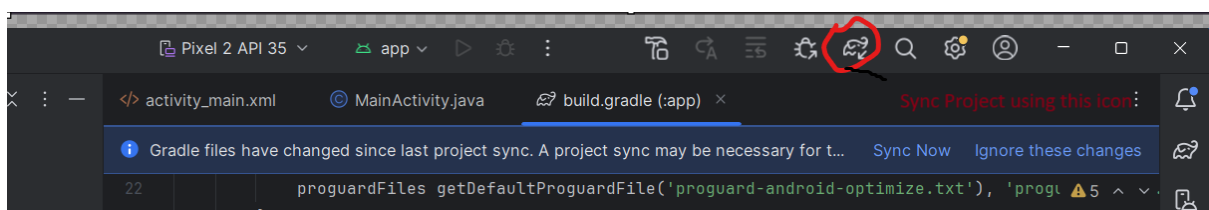
}

dependencies {
    // View/Layout
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

    // Core libraries
    implementation 'androidx.core:core:1.12.0'
    implementation libs.appcompat
    implementation libs.material
    implementation libs.activity

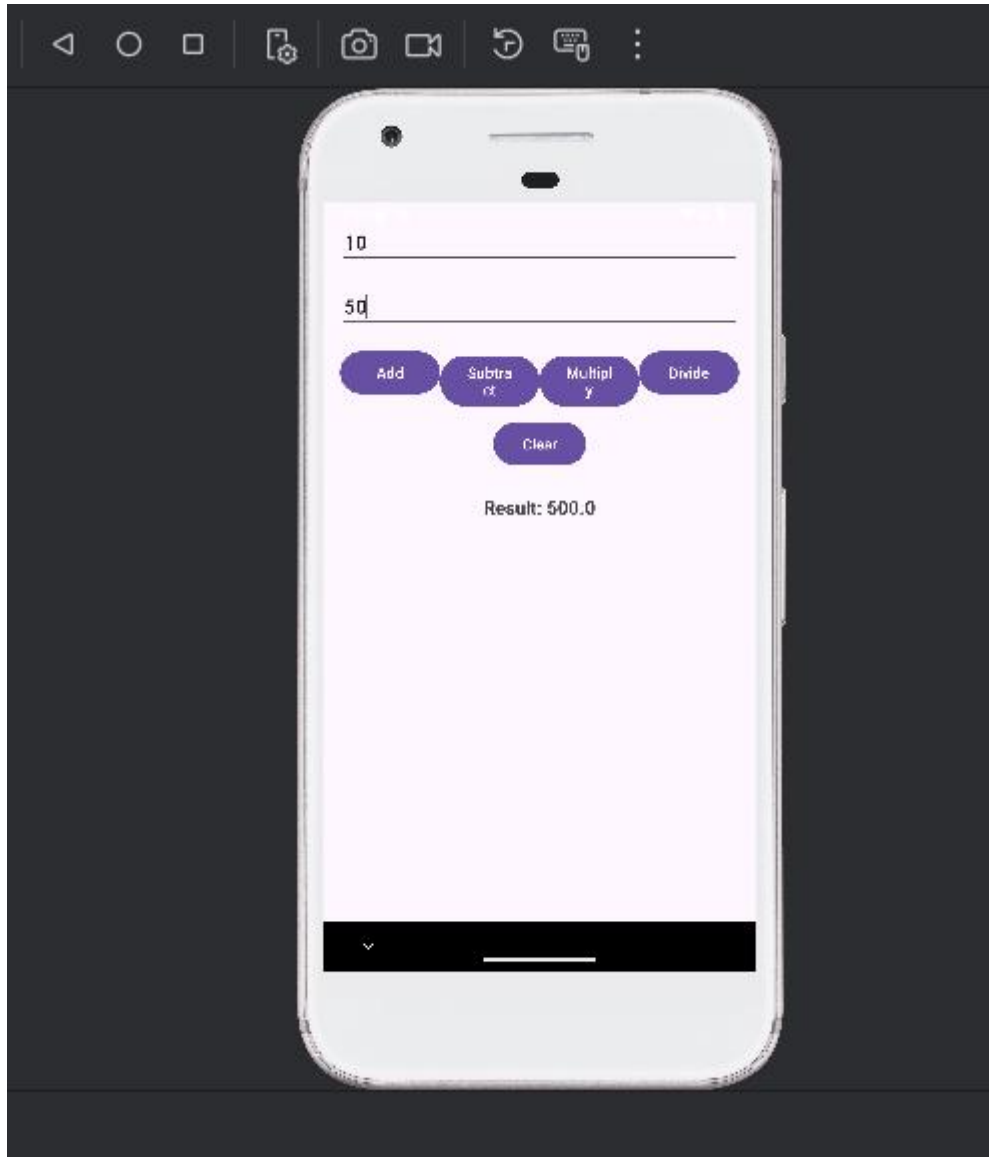
    // Testing
    testImplementation libs.junit
    androidTestImplementation libs.ext.junit
    androidTestImplementation libs.espresso.core
}

```



Step 5 – Click the Play Button to Run the project using the Emulator and verify the output as below

Output:



RESULT:

An calculator is designed using Android studio which uses controls like Button, TextView, EditText having basic functionality like Addition, Subtraction, multiplication, and Division.

EXP.NO: 8	Develop an application to change the font and color of the text and display toast message when the user presses the button.
DATE: 23/04/2025	

AIM:

To develop an Android application that allows the user to change the font and color of text displayed on the screen and display a Toast message when a button is pressed.

ALGORITHM:

1. Initialize the Android Application:

- Set up a new Android project in **Android Studio**.
- Create a **MainActivity** Java file for the main screen of the application.

2. Design the User Interface (UI):

- In the **XML layout file**, add the following UI components:
 - **TextView**: For displaying the text whose font and color will be changed.
 - **Button**: To trigger the change in font and color, and show a **Toast message**.
 - **Spinner or Color Picker**: To let the user choose a color (optional, or a simple button for a predefined color change).
 - **RadioButtons** or a **Dropdown Menu** to select different font styles (optional).

3. Declare Variables:

- Create references for the **TextView**, **Button**, **Color Picker** (if used), and **Radio Buttons** (if used to change font style).
- Define variables for the **font style**, **color**, and the **message** to display in the **Toast**.

4. Set Event Listener for the Button:

- Set an **OnClickListener** for the **Button**.
- Inside the **onClick()** method, write the logic to:
 - Change the **font** of the **TextView** based on the selected font style.

- Change the **color** of the **TextView** based on the selected color.
- Display a **Toast message** indicating the action has been performed.

5. Implement Font Style Changes:

- Change the **TextView's font** using `setTypeface()` to apply a selected font style (e.g., bold, italic).
- You can either use predefined styles (like `Typeface.BOLD` or `Typeface.ITALIC`) or let the user choose from a list of font styles.

6. Implement Color Change:

- Change the **TextView's text color** by using the `setTextColor()` method and applying the selected color (e.g., `Color.RED`, `Color.BLUE`, or a color from the **Color Picker**).
- If using a color picker or dropdown, get the selected color and set it on the **TextView**.

7. Show Toast Message:

- Display a **Toast** message to the user when the button is clicked, informing them that the text style or color has changed. You can also show a message like "Font and color changed!" or any other appropriate message.

8. End the Operation:

- After the changes are made, the application will display the updated **TextView** with the new font and color, and the **Toast message** will appear to provide feedback.

Code:

Step 1: Create the Project

- **Template:** Empty Views Activity
- **Project name** – Font_Change
- **Package** - com.example.font_change
- **Language:** Java
- **Minimum SDK** – API 24
- **Build system:** Groovy DSL

Step 2: Under res → layout → Updated activity_main.xml using **ConstraintLayout**

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/mainLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <TextView
        android:id="@+id/sampleText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!!!!!"
        android:textSize="24sp"
        android:textColor="#000000"
        android:fontFamily="@font/roboto_regular"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/changeStyleButton"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintVertical_bias="0.4"/>

    <Button
        android:id="@+id/changeStyleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Change Font Style"
        app:layout_constraintTop_toBottomOf="@id/sampleText"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_marginTop="20dp"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 3: Add a Custom Font in Android Studio

Add the Font Resource

1. In Android Studio:

- Right-click on res/ → **New** → **Android Resource Directory**
2. In the dialog:
- **Directory name:** font
 - **Resource type:** Select font from the dropdown
 - Click **OK**
3. Download the font:
- Open the link → <https://fonts.google.com/>
 - Select the font Roboto → Get Font → Download all
 - Open the downloaded zip file and go to the folder "static". Choose the font "Roboto-Regular.ttf" and rename to "roboto-regular.ttf". copy and paste this inside the font folder under drawable in Android studio.

Step 3 : Update the MainActivity.java file with this content

```
package com.example.font_change;
```

```
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.res.ResourcesCompat;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    TextView sampleText;
    Button changeStyleButton;
    boolean isChanged = false;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        sampleText = findViewById(R.id.sampleText);
        changeStyleButton = findViewById(R.id.changeStyleButton);
```

```

changeStyleButton.setOnClickListener(v -> {
    if (!isChanged) {
        sampleText.setTextColor(Color.parseColor("#FF5722")); // Orange
        Typeface typeface = ResourcesCompat.getFont(this, R.font.roboto_regular);
        sampleText.setTypeface(getResources().getFont(R.font.roboto_regular),
Typeface.BOLD_ITALIC);
        Toast.makeText(MainActivity.this, "Style Changed!",
Toast.LENGTH_SHORT).show();
    } else {
        sampleText.setTextColor(Color.BLACK);
        sampleText.setTypeface(Typeface.DEFAULT);
        Toast.makeText(MainActivity.this, "Style Reset!",
Toast.LENGTH_SHORT).show();
    }
    isChanged = !isChanged;
});
}
}

```

Step 4 – Under Gradle Scripts section → update the build.gradle(Module:app)

```

plugins {
    alias(libs.plugins.androidApplication)
}

android {
    namespace 'com.example.font_change'
    compileSdk 34

    defaultConfig {
        applicationId "com.example.font_change"
        minSdk 26
        targetSdk 34
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-

```

```

rules.pro'
    }
}

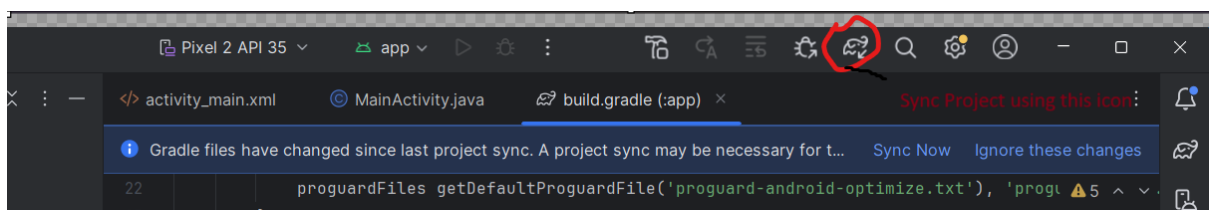
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
}

dependencies {
    // View/Layout
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

    // Core libraries
    implementation 'androidx.core:core:1.12.0'
    implementation libs.appcompat
    implementation libs.material
    implementation libs.activity

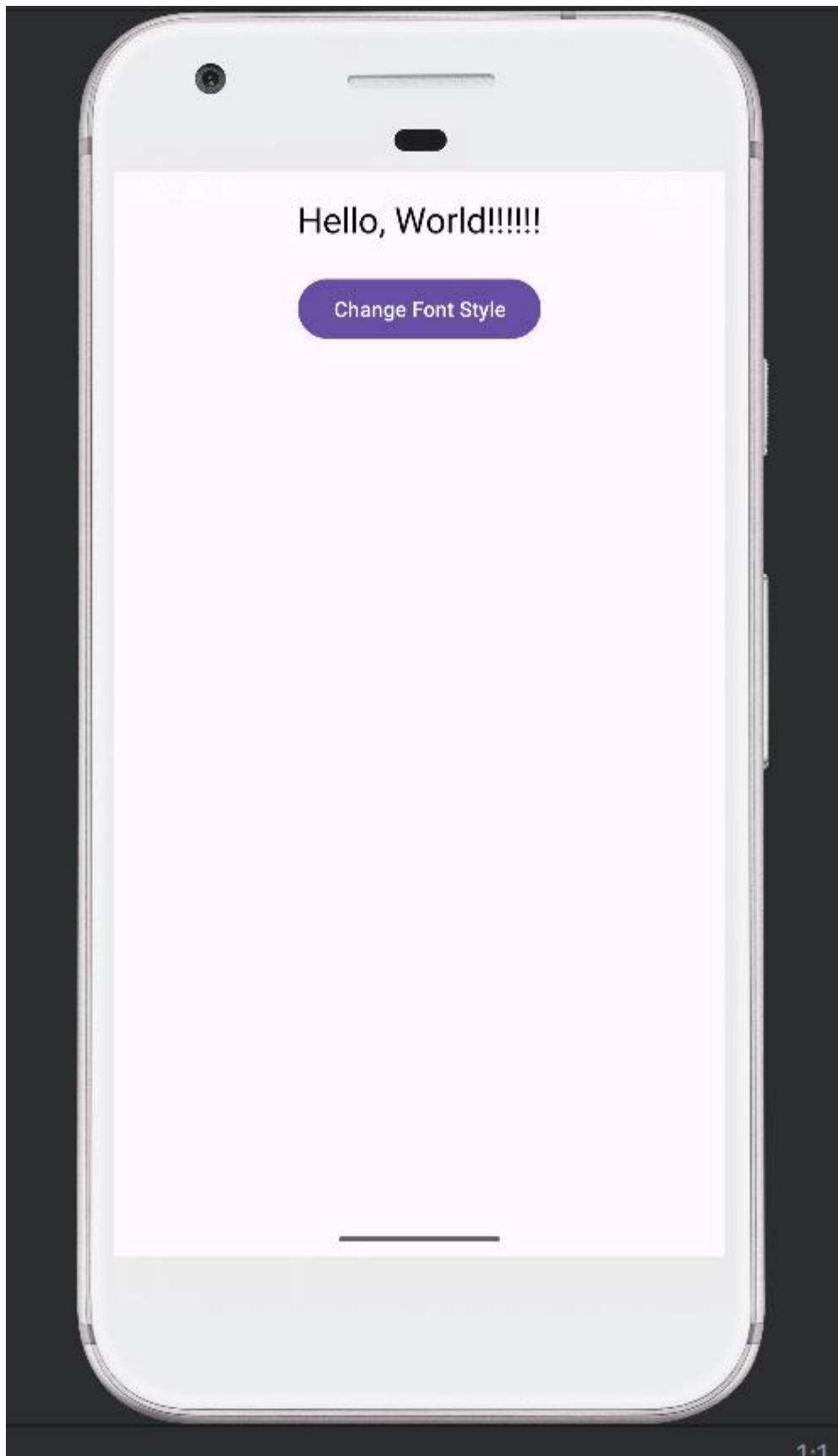
    // Testing
    testImplementation libs.junit
    androidTestImplementation libs.ext.junit
    androidTestImplementation libs.espresso.core
}

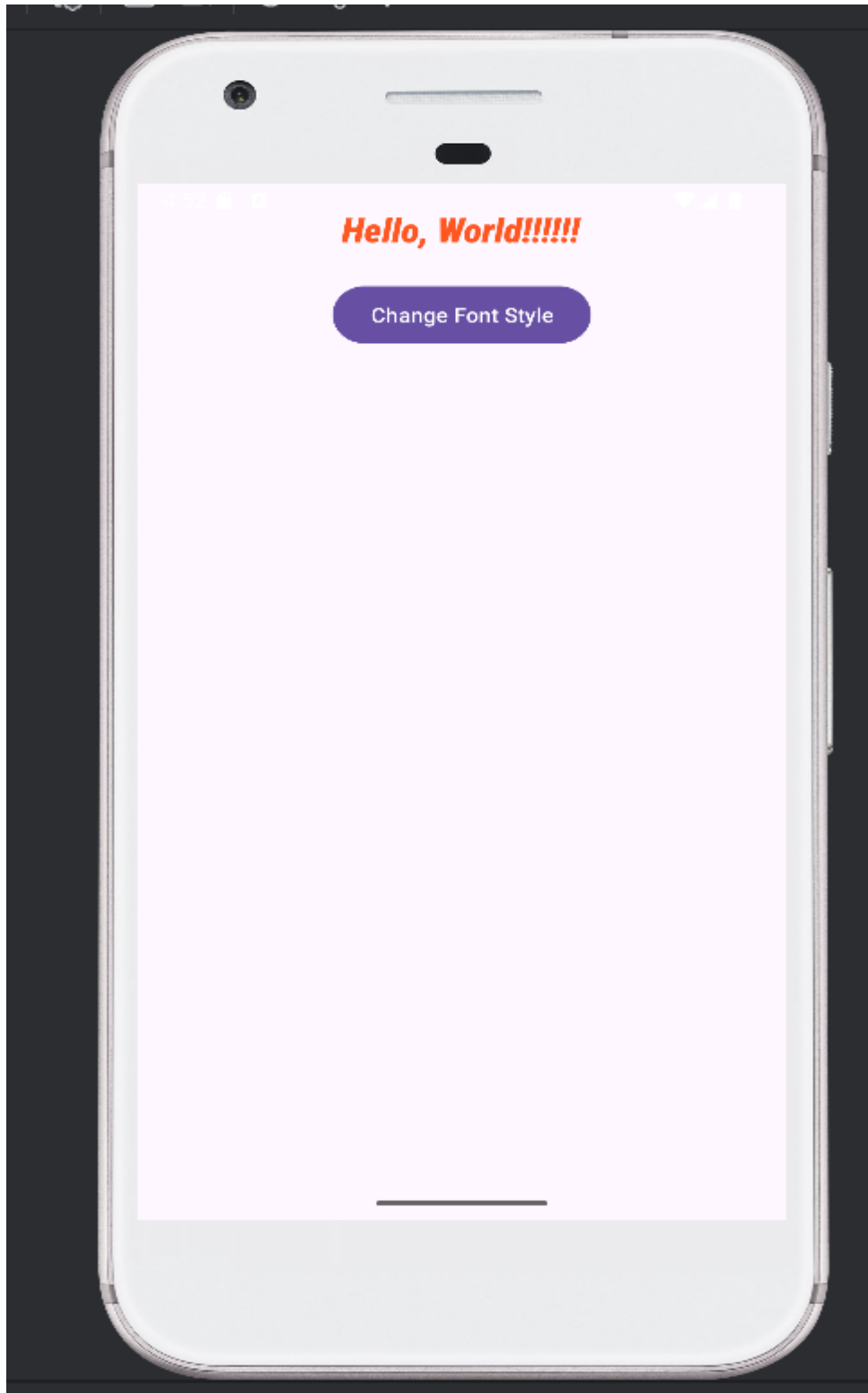
```



Step 5 – Click the Play Button to Run the project using the Emulator and verify the output as below

Output:





RESULT:

An mobile application is developed in Android studio to change the font and color of the text and display toast message when the user presses the button.

EXP.NO: 9	Develop a mobile application to send an email.
DATE:07/05/2025	

AIM:

To develop a mobile application that enables the user to send an email through the app. The application will provide an interface to input the recipient's email address, subject, message content, and optionally attach files. It will use Android's built-in email capabilities to send the email,

ALGORITHM:

Initialize the Android Application:

- Set up a new Android project in **Android Studio**.
- Create a **MainActivity** for the UI where the user will input email details (recipient address, subject, message, and attachment if any).

Design the User Interface (UI):

- In the **XML layout file**, create the following UI components:
 - **EditText** fields for entering:
 - Recipient's email address.
 - Subject of the email.
 - Message content.
 - **Button** to trigger the email sending process.
 - **Optional: Button** to select file attachments (if required).
 - **TextView** or **Toast** for displaying feedback to the user (e.g., success or failure).

Declare Variables:

- Declare variables for **EditText** fields (for email address, subject, and message).
- Declare a **Button** for sending the email and another **Button** for attachment (optional).
- Declare a **String** variable to hold the recipient email, subject, and message content.

Set OnClickListener for Send Button:

- Set an **OnClickListener** for the **Send Email Button**.

- Inside the `onClick()` method, retrieve the values entered by the user:
 - Get the email recipient, subject, and message using `getText().toString()` for the **EditText** fields.

Validate User Input:

- Ensure that the **email address** is in a valid format (e.g., `name@domain.com`).
- Check if all necessary fields (email, subject, message) are not empty.
- If any validation fails, show an error message (Toast) and do not proceed.

Create an Intent to Send the Email:

- Use an **Intent** to trigger the **email client** (e.g., Gmail, Outlook, etc.).
 - Set the action to `Intent.ACTION_SENDTO`.
 - Use `mailto:` URI scheme to trigger the email client and set the **recipient, subject, and message** as parameters.
 - Example: `mailto:recipient@example.com?subject=Subject&body=Message`
- Set the **email fields** (recipient, subject, message) in the intent.

Optional: Attach Files (if applicable):

- If attachments are required, use `Intent.EXTRA_STREAM` to attach a file (e.g., an image or document).
- Use a file picker to allow the user to select the attachment, then add the file URI to the intent.

Send the Email:

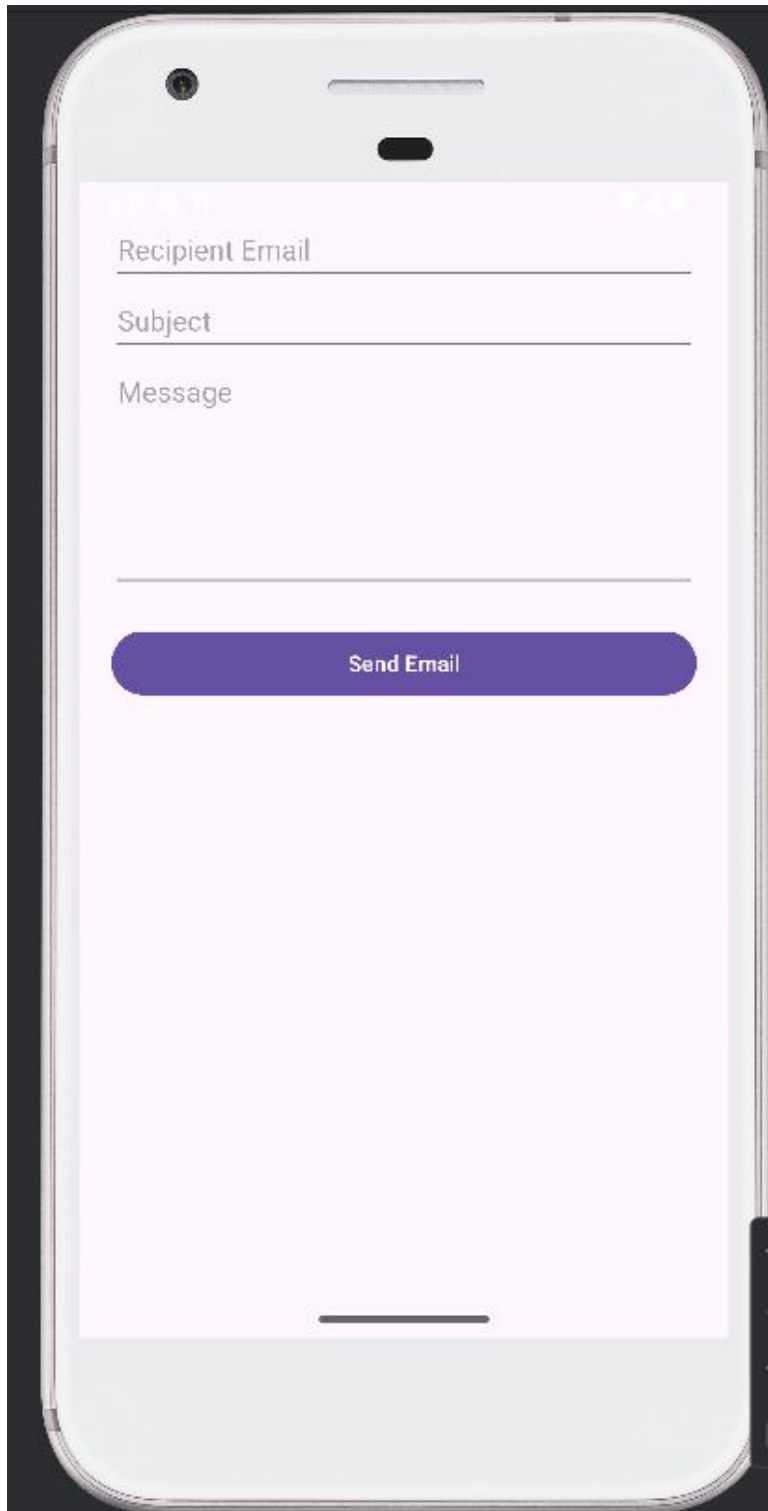
- Call `startActivity(intent)` to invoke the email client and send the email.
- If the email client is installed, it will open with the email details prefilled, and the user can press the send button.
- If the email client is not installed or an error occurs, display an appropriate **Toast message** informing the user.

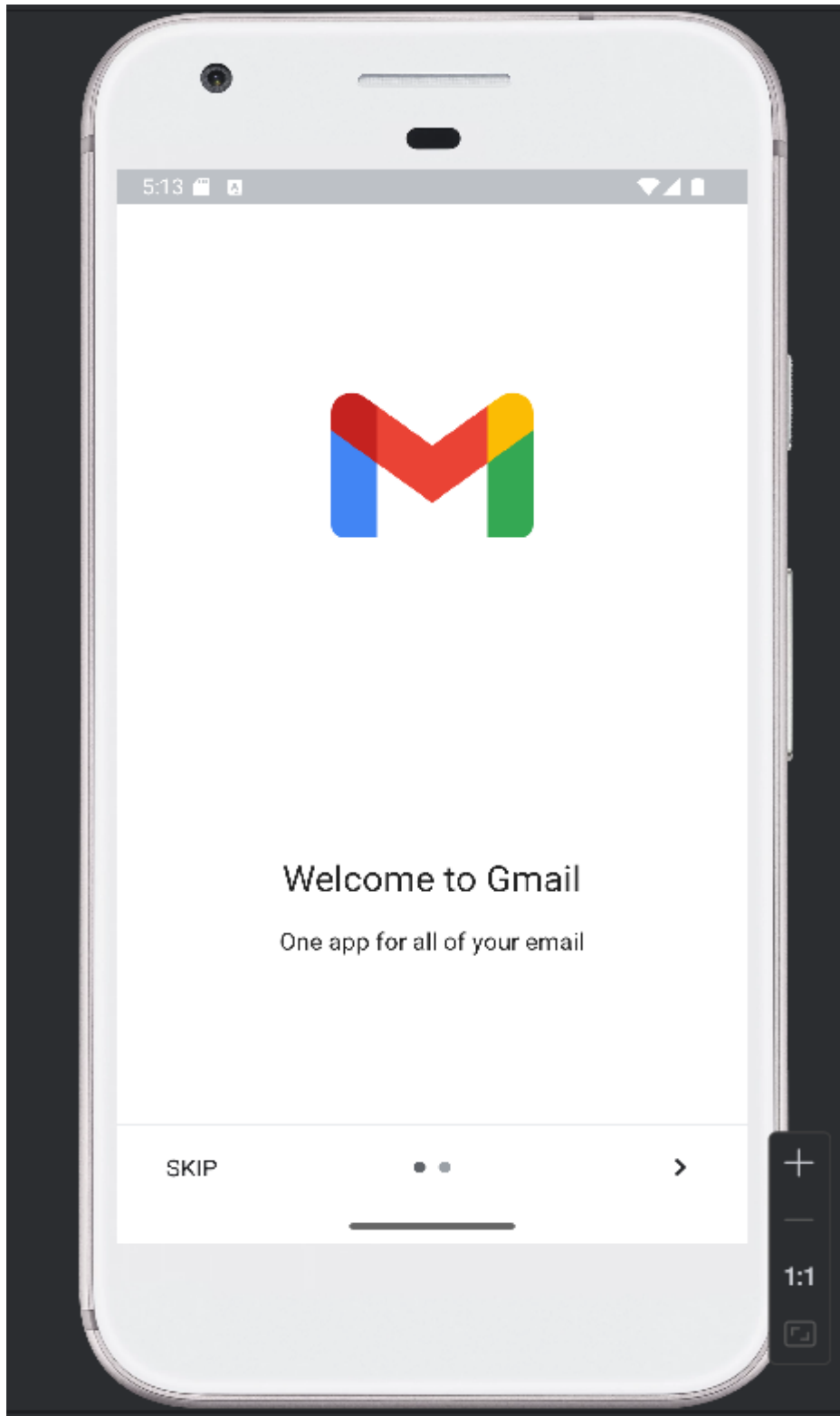
Display Feedback:

- If the email was successfully sent, show a **Toast** or update the UI with a success message.
- If there was an error (e.g., invalid email, missing email client), display an error **Toast** or show an error message in the **TextView**.

End the Operation:

- After the email has been sent or if there was an error, the operation is complete. The user can send another email or exit the app.

Output:



RESULT:

A mobile application is developed to send an email.