WCNL

Experiment No 1 : HTML attributes and form tags for creating a webpage.

Code:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<header>
<h1>Page Heading</h1>
<nav>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
</header>
<main>
<article>
<h2>Article Heading</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque vel venenatis felis. Fusce eu
mollis orci.</p>
<section>
<h3>Section Heading</h3>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque vel venenatis felis. Fusce eu
mollis orci.</p>
</section>
```

```
</article>
<aside>
<h3>Related</h3>
<ul>
<li><a href="#">Related link 1</a></li>
<li><a href="#">Related link 2</a></li>
<li><a href="#">Related link 3</a></li>
</ul>
</aside>
</main>
<footer>
<p> Footer</p>
</footer>
</body>
</html>
```

Experiment No 2 . CSS3 Selectors for decorating the webpage.

Code:

```
body {
font-family: Arial, sans-serif;
}
h1, h2, h3 {
color: #333;
}
header {
background-color: #eee;
padding: 20px;
}
nav ul {
list-style-type: none;
margin: 0;
```

```
padding: 0;
}
nav li {
display: inline-block;
margin-right: 20px;
}
main {
max-width: 800px;
margin: 0 auto;
padding: 20px;
}
footer {
background-color: #eee;
text-align: center;
padding: 20px;
}
a {
color: #007bff;
text-decoration: none;
}
a:hover {
text-decoration: underline;
}
article {
margin-bottom: 40px;
}
aside {
border: 1px solid #ddd;
padding: 20px;
}
```

Experiment No 3 bootstrap based form for the validation process.

Code :

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Bootstrap Form with Validation</title>
 <!-- Bootstrap CSS -->
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet">
 <style>
  body {
   padding: 20px;
  }
 </style>
</head>
<body>

 <div class="container">
  <h1>Bootstrap Form Validation</h1>
  <form class="needs-validation" novalidate>
   <div class="mb-3">
    <label for="name" class="form-label">Name</label>
    <input type="text" class="form-control" id="name" required>
    <div class="invalid-feedback">
     Please enter your name.
    </div>
   </div>

   <div class="mb-3">
```

```
  <label for="email" class="form-label">Email address</label>
  <input type="email" class="form-control" id="email" required>
  <div class="invalid-feedback">
   Please enter a valid email.
  </div>
</div>

<div class="mb-3">
  <label for="password" class="form-label">Password</label>
  <input type="password" class="form-control" id="password" minlength="8" required>
  <div class="invalid-feedback">
   Password must be at least 8 characters long.
  </div>
</div>

<div class="mb-3">
  <label for="confirmPassword" class="form-label">Confirm Password</label>
  <input type="password" class="form-control" id="confirmPassword" required>
  <div class="invalid-feedback">
   Please confirm your password.
  </div>
</div>

<div class="mb-3 form-check">
  <input type="checkbox" class="form-check-input" id="terms" required>
  <label class="form-check-label" for="terms">I agree to the terms and conditions</label>
  <div class="invalid-feedback">
   You must agree before submitting.
  </div>
</div>
```

```
    <button type="submit" class="btn btn-primary">Submit</button>
   </form>
  </div>

  <!-- Bootstrap JS and Popper.js -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
  <script>
   // Bootstrap form validation
   (function () {
    'use strict'

    // Fetch all the forms we want to apply custom Bootstrap validation styles to
    var forms = document.querySelectorAll('.needs-validation')

    // Loop over them and prevent submission if they are invalid
    Array.prototype.slice.call(forms)
     .forEach(function (form) {
      form.addEventListener('submit', function (event) {
       if (!form.checkValidity()) {
        event.preventDefault()
        event.stopPropagation()
       }

       form.classList.add('was-validated')
      }, false)
     })
   })()
  </script>
</body>
</html>
```

Experiment No 4. Implement vanilla JavaScript for form validations with DOM elements.

Code:

STEP 1: create an html file named 'index.html'

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>JavaScript Demo with User Input</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   padding: 20px;
   background-color: #f4f4f9;
  }
  h2 {
   color: #333;
  }
  p, .output {
   font-size: 1.1em;
  }
  input, button {
   padding: 8px;
   margin: 5px;
  }
  button {
   background-color: #007bff;
   color: white;
```

```
    border: none;
    cursor: pointer;
    border-radius: 4px;
  }
  button:hover {
    background-color: #0056b3;
  }
  .output {
    font-weight: bold;
    color: #5a5a5a;
    margin-top: 10px;
  }
 </style>
</head>
<body>
 <h2>JavaScript User Input Demonstrations</h2>

 <!-- JavaScript Variables with User Input -->
 <h3>Variables</h3>
 <p>Enter two numbers to add:</p>
 <input type="number" id="var1">
 <input type="number" id="var2">
 <button onclick="displaySum()">Calculate Sum</button>
 <p id="output1" class="output"></p>

 <!-- JavaScript Function with User Input -->
 <h3>Function</h3>
 <p>Enter two numbers to multiply:</p>
 <input type="number" id="func1">
 <input type="number" id="func2">
 <button onclick="displayMultiplication()">Multiply</button>
```

```
 <p id="output2" class="output"></p>

 <!-- JavaScript Condition (if...else) with User Input -->
 <h3>Condition (if...else)</h3>
 <p>Enter a number to check if it's even or odd:</p>
 <input type="number" id="conditionInput">
 <button onclick="checkEvenOdd()">Check</button>
 <p id="output3" class="output"></p>

 <!-- JavaScript Loop with User Input -->
 <h3>Loop</h3>
 <p>Enter a list of items (comma separated):</p>
 <input type="text" id="loopInput">
 <button onclick="displayItems()">Display Items</button>
 <p id="output4" class="output"></p>

 <!-- JavaScript Input Validation -->
 <h3>Input Validation</h3>
 <p>Enter a number between 1 and 10:</p>
 <input type="number" id="validateInput">
 <button onclick="validateNumber()">Validate</button>
 <p id="output5" class="output"></p>

 <script src="userInput.js"></script>
</body>
</html>
```

JAVASCRIPT

STEP 2: CREATE A JS FILE NAMED 'userInput.js'

```
// Variables: Addition based on user input
```

```
function displaySum() {
  let x = parseFloat(document.getElementById("var1").value);
  let y = parseFloat(document.getElementById("var2").value);
  let sum = x + y;
  document.getElementById("output1").innerHTML = "The sum is: " + sum;
}

// Function: Multiplication based on user input
function displayMultiplication() {
  let num1 = parseFloat(document.getElementById("func1").value);
  let num2 = parseFloat(document.getElementById("func2").value);
  let product = num1 * num2;
  document.getElementById("output2").innerHTML = "The product is: " + product;
}

// Condition (if...else): Check if user input is even or odd
function checkEvenOdd() {
  let number = parseInt(document.getElementById("conditionInput").value);
  let result = (number % 2 === 0) ? "even" : "odd";
  document.getElementById("output3").innerHTML = "The number is: " + result;
}

// Loop: Display each item from user input
function displayItems() {
  let items = document.getElementById("loopInput").value.split(",");
  let output = "";
  for (let i = 0; i < items.length; i++) {
    output += items[i].trim() + "<br>";
  }
  document.getElementById("output4").innerHTML = "Items:<br>" + output;
}
```

```
// Input Validation: Check if the input is between 1 and 10
function validateNumber() {
  let num = parseInt(document.getElementById("validateInput").value);
  let message = (num >= 1 && num <= 10) ? "Input OK" : "Input not valid";
  document.getElementById("output5").innerHTML = message;
}
```

Exp No 5. ReactJS project initialization with some major functionalities.

Code:

STEP 1: npx create-react-app react-demo

cd react-demo

STEP 2: UPDATE THESE CODES

1) APP.JS

```
// src/App.js
import React from 'react';
import './App.css';
import Counter from './components/Counter';
import ItemList from './components/ItemList';
import NumberValidation from './components/NumberValidation';

function App() {
  return (
    <div className="App">
      <h1>ReactJS Functionalities Demo</h1>
      <Counter />
      <ItemList />
      <NumberValidation />
    </div>
  );
```

```
}

export default App;
```

2) APP.CSS

```
/* src/App.css */
.App {
  font-family: Arial, sans-serif;
  text-align: center;
  padding: 20px;
  background-color: #f4f4f9;
}
h2 {
  color: #333;
}
button {
  margin: 5px;
  padding: 8px;
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
  border-radius: 4px;
}
button:hover {
  background-color: #0056b3;
}
```

STEP 3: CREATE A COMPONENTS FOLDER IN SRC AND ADD THESE FILES

1) Counter.js

```
// src/components/Counter.js
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);

  return (
    <div>
      <h2>Counter</h2>
      <p>Current Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
    </div>
  );
}

export default Counter;
```

2) ItemList.js

```
// src/components/ItemList.js
import React, { useState } from 'react';

function ItemList() {
  const [items, setItems] = useState(['Apple', 'Banana', 'Cherry']);

  return (
    <div>
```

```
    <h2>Item List</h2>
    <ul>
     {items.map((item, index) => (
       <li key={index}>{item}</li>
     ))}
    </ul>
   </div>
  );
}

export default ItemList;
```

3) NumberValidation.js

```
// src/components/NumberValidation.js
import React, { useState } from 'react';

function NumberValidation() {
  const [number, setNumber] = useState('');
  const [message, setMessage] = useState('');

  const validateNumber = () => {
   const num = parseInt(number);
   if (num >= 1 && num <= 10) {
    setMessage('Input OK');
   } else {
    setMessage('Input not valid');
   }
  };

  return (
   <div>
```

```
    <h2>Number Validation</h2>
    <input
     type="number"
     value={number}
     onChange={(e) => setNumber(e.target.value)}
     placeholder="Enter a number"
    />
    <button onClick={validateNumber}>Validate</button>
    <p>{message}</p>
   </div>
  );
}

export default NumberValidation;
```

STEP 4: DELETE LOGO.SVG,APP.TEST.JS, SETUPTEST,JS, REPORTWEBVITALS.JS

STEP 5: CHANGE index.js

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
 <React.StrictMode>
  <App />
 </React.StrictMode>,
 document.getElementById('root')
);
```

STEP 6: npm start

CODE RUNNING Condition :
Step 1: Check Required Software

1. **Node.js and npm**:

   - Open your terminal or command prompt.

   - Run `node -v` and `npm -v` to check if Node.js and npm are installed.

   - If they are not installed, download and install Node.js from nodejs.org. This will also install npm.

2. **npx (comes with npm)**:

   - Run `npx -v` to verify the version. npx is typically included with npm 5.2+.

3. **Confirm React setup tools**:

   - You don’t need to install React separately. Instead, you'll use `npx` to create the React app, which will automatically set up the necessary files.

Step 2: Create the Project Folder and Setup React App

1. **Create a project folder**:

   - Choose or create a directory where you want to store your project. For example, create a folder called `react-demo` on your desktop or preferred location.

2. **Initialize the React project**:

   - In the terminal, navigate to the `react-demo` folder.

   - Run `npx create-react-app react-demo`. This command will set up the React application and install the necessary dependencies.

3. **Navigate into the project folder**:

   - Use `cd react-demo` to navigate into the new project directory.

Step 3: Project Folder Structure

After creating the React app, the following folder structure should be in place inside `react-demo`:

```
react-demo/
├── node_modules/
├── public/
│   ├── index.html
├── src/
│   ├── App.css
│   ├── App.js
│   ├── index.css
│   ├── index.js
└── package.json
```

Step 4: Update Files

1. **App.js** (located in `src/` folder):

   Update `App.js` to match your requirements for importing and displaying the components (Counter, ItemList, NumberValidation).

2. **App.css** (located in `src/` folder):

   Customize the CSS to style your app.

3. **Add Components Folder and Files**:

   - Create a new folder named `components` inside the `src` folder.

   - Add the following files to `components`:

     - **Counter.js**: Implements a counter with increment and decrement buttons.

     - **ItemList.js**: Displays a simple list of items.

- **NumberValidation.js**: Validates a number to check if it's between 1 and 10.

4. **Remove Unused Files**:

- Inside `src`, delete `logo.svg`, `App.test.js`, `setupTests.js`, and `reportWebVitals.js`.

Step 5: Modify `index.js`

- Update `index.js` in `src/` to render your `App` component using `React.StrictMode`.

Step 6: Running the Application

1. **Install Dependencies** (if required):

- Inside the project folder (`react-demo`), run `npm install` to ensure all dependencies are installed.

2. **Run the Application**:

- Run `npm start` to start the development server. This should automatically open your app in the default browser at `http://localhost:3000`.

Summary

- **Required Files**:

- `App.js`, `App.css` (inside `src/`)

- `Counter.js`, `ItemList.js`, `NumberValidation.js` (inside `src/components/`)

- `index.js` (inside `src/`)

- **Commands to Run**:

- `npx create-react-app react-demo`

- `cd react-demo`

- `npm install` (optional if dependencies aren’t automatically installed)

- `npm start` to launch the development server

Exp NO 6 a NodeJS routing for method driven CRUD operations.

STEP 1: npx create-react-app react-crud-demo

cd react-crud-demo

npm start

STEP 2: MODIFY THESE FILES

1) APP.JS

```
// src/App.js
import React, { useState } from 'react';
import './App.css';

function App() {
  const [items, setItems] = useState([]);
  const [newItem, setNewItem] = useState('');
  const [isEditing, setIsEditing] = useState(false);
  const [currentItem, setCurrentItem] = useState({});

  function addItem() {
    if (!newItem) return;
    const item = { id: items.length + 1, value: newItem };
    setItems([...items, item]);
```

```
  setNewItem('');
}

function editItem(item) {
  setIsEditing(true);
  setCurrentItem({ ...item });
}

function updateItem() {
  setItems(items.map((item) => (item.id === currentItem.id ? currentItem : item)));
  setIsEditing(false);
  setCurrentItem({});
}

function deleteItem(id) {
  setItems(items.filter((item) => item.id !== id));
}

return (
  <div className="App">
    <h1>Simple React CRUD App</h1>
    <input
      type="text"
      placeholder="Add a new item"
      value={newItem}
      onChange={(e) => setNewItem(e.target.value)}
    />
    <button onClick={addItem}>Add</button>

    {isEditing ? (
      <div>
```

```
        <input
          type="text"
          value={currentItem.value}
          onChange={(e) => setCurrentItem({ ...currentItem, value: e.target.value })}
        />
        <button onClick={updateItem}>Update</button>
      </div>
    ) : null}

    <ul>
      {items.map((item) => (
        <li key={item.id}>
          {item.value}
          <button onClick={() => editItem(item)}>Edit</button>
          <button onClick={() => deleteItem(item.id)}>Delete</button>
        </li>
      ))}
    </ul>
  </div>
 );
}

export default App;
```

2) APP.CSS

```
/* src/App.css */
.App {
 text-align: center;
 font-family: Arial, sans-serif;
 padding: 20px;
}
```

```
input {
  padding: 8px;
  margin: 5px;
}
button {
  padding: 8px;
  margin: 5px;
  cursor: pointer;
}
```

3) INDEX .JS

```
// src/index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

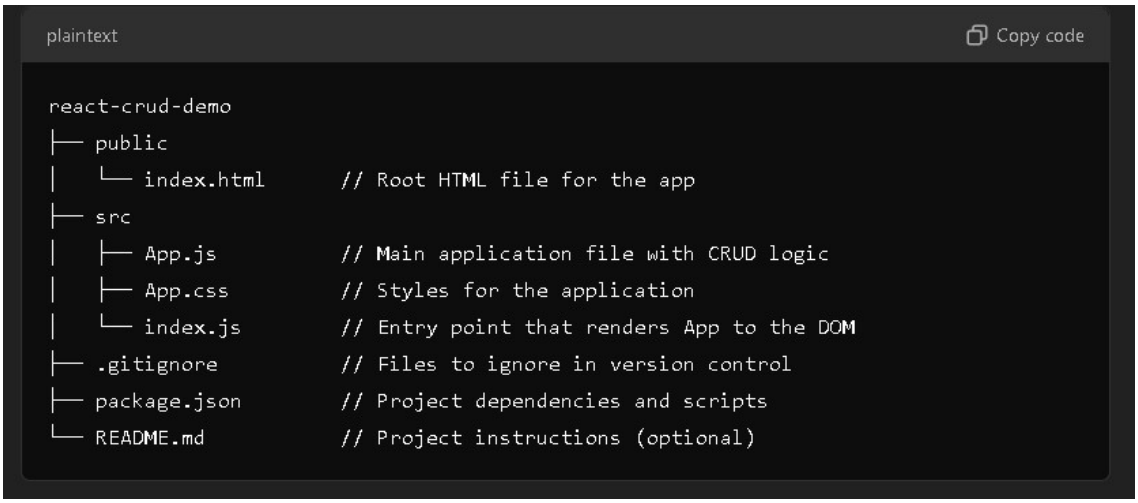
STEP 3: DELETE THESE FILES

App.tes.js

logo.svg

reportWebvitals.js

setupTests.js



How to run the following code :
Here's a step-by-step guide for setting up a React project, ensuring Node.js, npm, and React are installed, and running the application:

1. **Check Prerequisites:**

- **Node.js and npm** are required for working with React. To check if these are installed:

  - Open your terminal (or command prompt) and type:

    ```bash
    node -v
    ```

    ```bash
    npm -v
    ```

    If these return versions (e.g., `v16.14.0` for Node), they're installed. If not, download [Node.js](https://nodejs.org/) which comes with npm.

- **npx**: This is part of npm, so it should work once npm is installed. Check with:

  ```bash
  npx -v
  ```

2. **Setting up the Folder Structure for React Project:**

- Create a root folder for your project. You can name it something like `ReactProjects`:

```bash
mkdir ReactProjects
cd ReactProjects
```

- Inside this root folder, create your React application by running:

```bash
npx create-react-app react-crud-demo
```

This command will set up a `react-crud-demo` folder with all necessary configuration files and dependencies.

3. **React Project Folder Structure Overview:**

The created project will include the following important files:

- **`/src`**: Contains source code.
  - **App.js**: Main application file where you'll add the component code.
  - **App.css**: CSS file for styling the application.
  - **index.js**: Entry point that renders your `App` component to the DOM.
  - Other files (you'll delete some in the next steps).

- **`/public`**: Includes static assets like `index.html`, the root HTML file for the app.

4. **File Setup and Code Addition:**

- **Modify these files:**
  1. **App.js**: This file contains the main React component. Copy and paste the code provided for the `App.js` functionality.
  2. **App.css**: Contains the styles. Add the CSS code you were given for layout and button styling.
  3. **index.js**: Already configured, but it's the file where `App` is rendered into the DOM.

- **Delete unnecessary files** from `/src`:

- `App.test.js`
- `logo.svg`
- `reportWebVitals.js`
- `setupTests.js`

5. **Running the Application:**

- In the terminal, navigate into your project folder:

  ```bash
  cd react-crud-demo
  ```

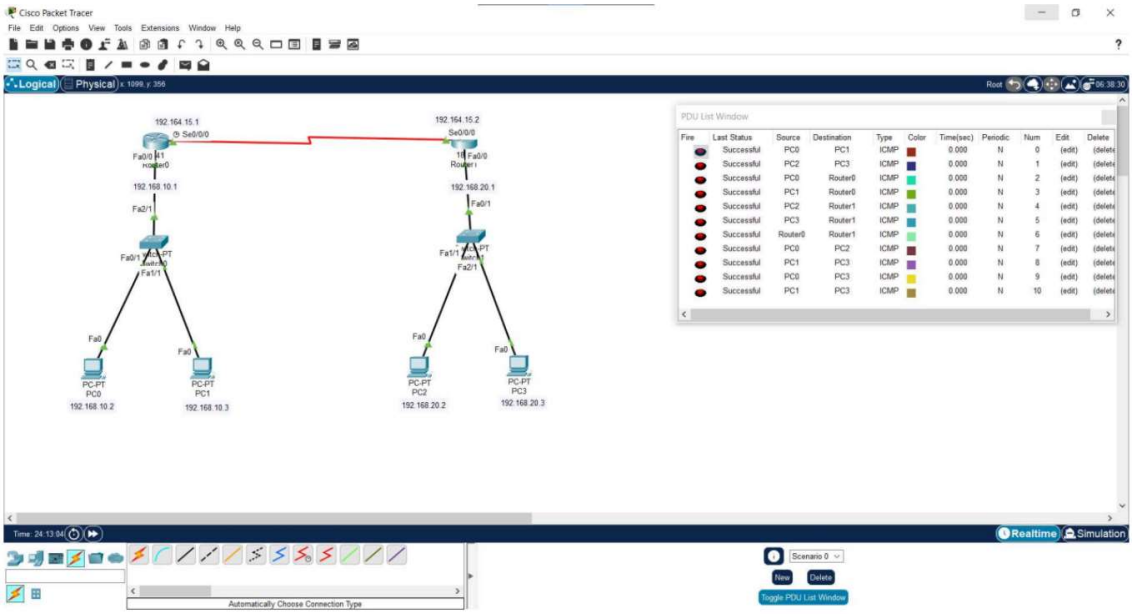
- Start the React app:

  ```bash
  npm start
  ```

  This command runs the application, typically launching it on `http://localhost:3000` in your default browser.

You should now see your React CRUD application running. You can edit files like `App.js` and `App.css` and see the changes update in real time.

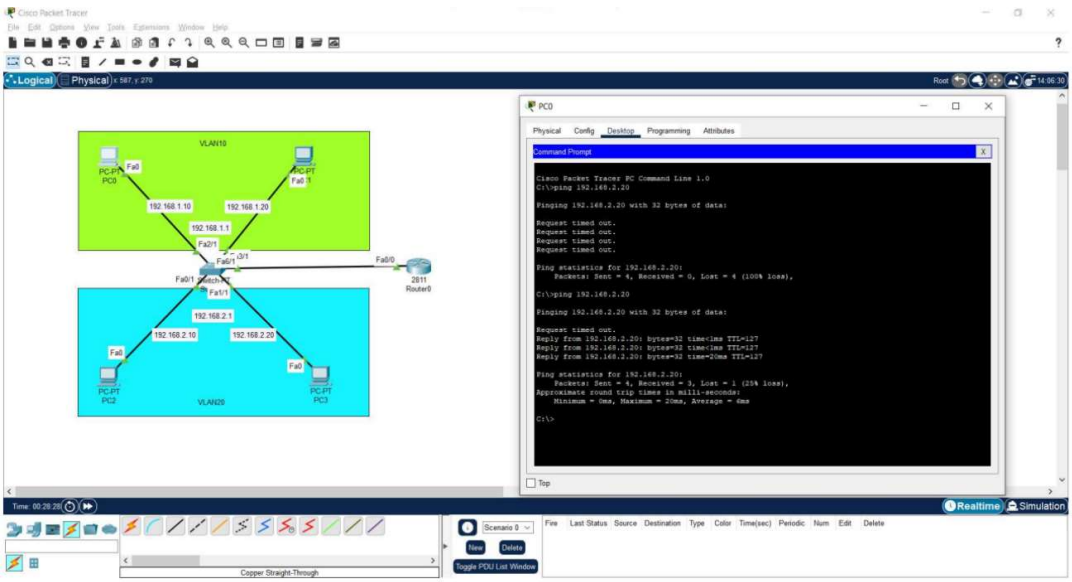
Exp NO 7 Dynamic Routing using Cisco packet tracer/GNS3



Download Link GNS3 : https://sourceforge.net/projects/gns-3/

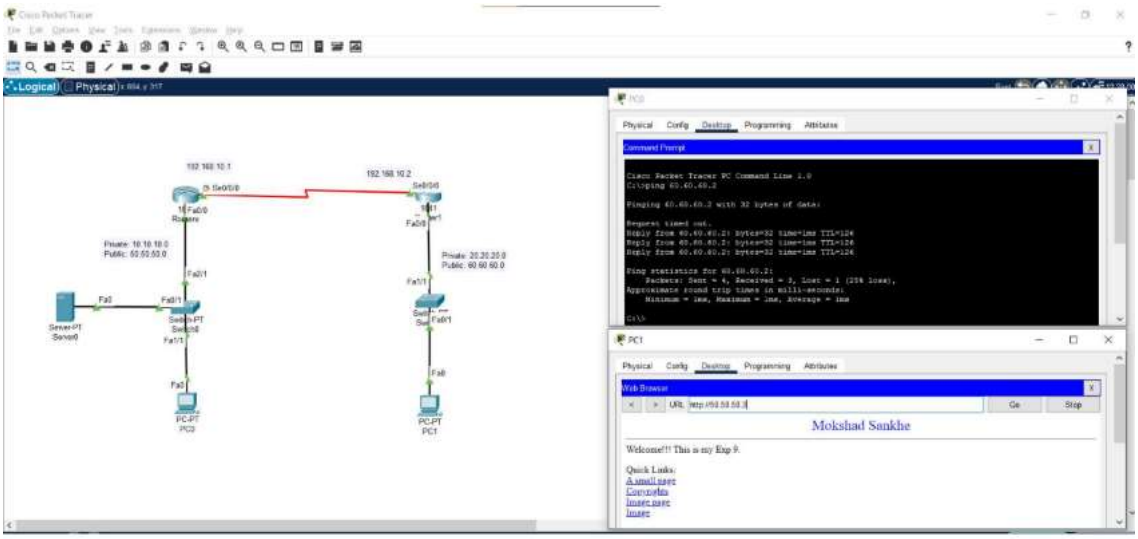
YouTube Link : Simple dynamic routing or ERP routing using CISCO packet tracer

Exp No 8 VLANs on the switch/router using Cisco packet tracer/GNS3



Youtube Video Link : Creating Virtual LAN (VLAN) using Packet Tracer

Exp No 9 simulate NAT on the router using Cisco packet tracer/GNS3



Youtube Link : NAT - Network Address Translation in Cisco Packet Tracer

Exp NO 10 simulation of Software Defined Network using Mininet.

Youtube Link : (70) Demo of generating SDN - Software Defined Networks using Mininet and Beacon Controller by Dharmik - YouTube