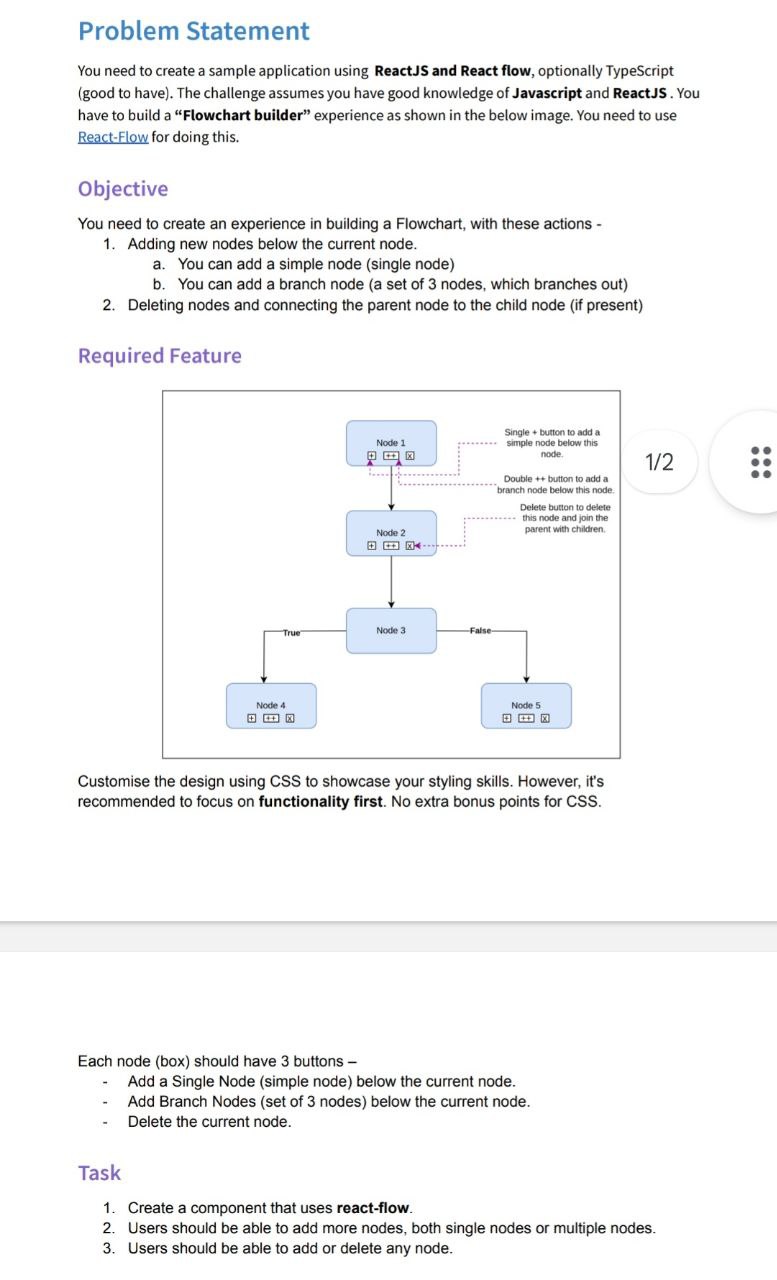
**PROBLEM STATEMENT -1 USING REACT JS AND REACTFLOW FRONTEND**

**Set-UP:**

**1)Download Visual Studio code**

**2)Open Terminal in VSC**

**3)We can Add extensions like React or React-Native snippets etc…**

**3)install node js to your PC and create react app**

**Commands:**

**1)npx create-react-app newapp**

**2)npm install react-flow-render**

**3)npm install react-flow**

**4)cd newapp**

**5)To run npm start.**

**process :**

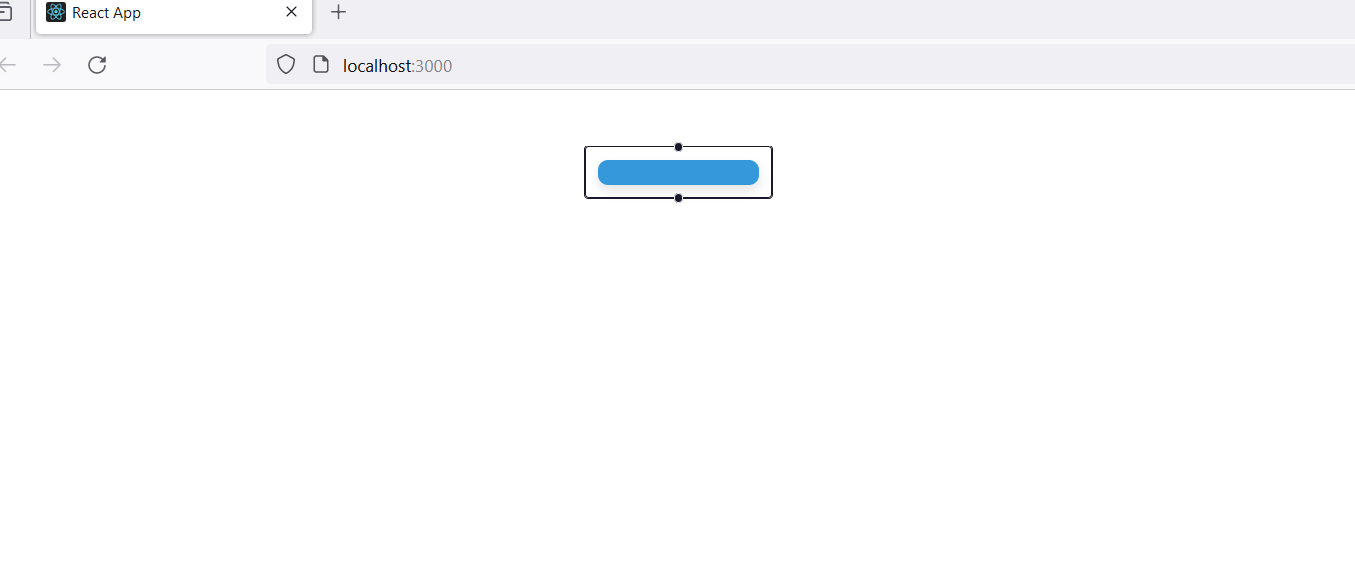
**1)Go to App.js in the newapp and remove the code in it so first we need to create a Intial single node with Id,data,label,type,position.**

**2)Import :**

|  |
| --- |
| import React, { useState, useEffect } from 'react'; |
| import ReactFlow from 'react-flow-renderer'; |
| import './App.css'; |

**3)Every node will contain id,data,label,type,position**

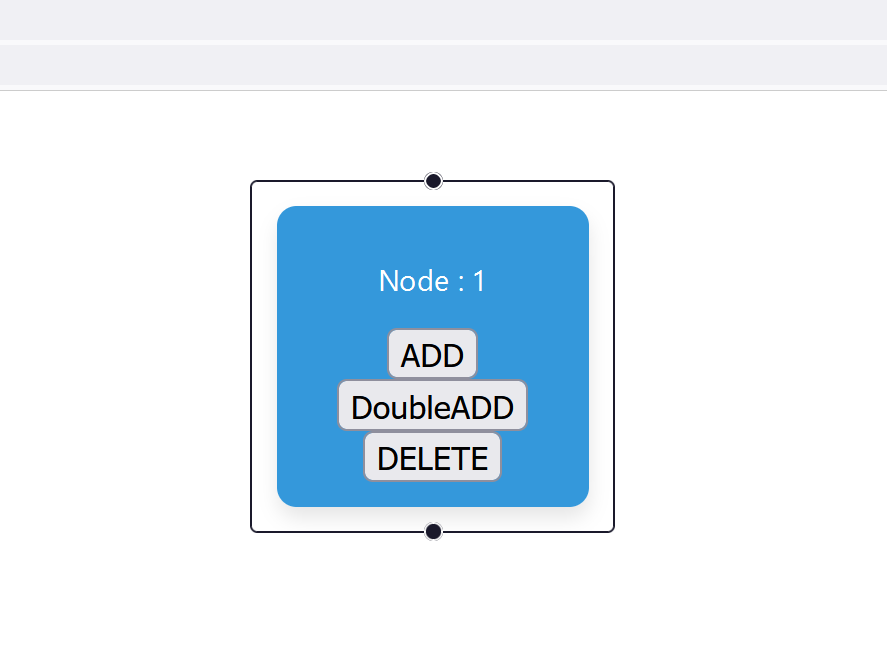
|  |
| --- |
| **Code:** |
| const App = () => {    const [nodes, setNodes] = useState([        {        id: "1",        data: {          label: (            <div className='box'>            </div>          ),        },        type: 'default',        position: { x: 650, y: 50 },      },    ]);      return (      <div style={{ height: '100vh', width: '100%' }}>        <ReactFlow  nodes={nodes}  />      </div>    );  };  export default App; |

**4)Output: **

**5)Add Node ID and Buttons = {Add,Double Add , Delete }**

|  |
| --- |
| **Code:** |
| const App = () => {    const [nodes, setNodes] = useState([      {        id: '1',        data: {          label: (            <div className='box'>              <p>Node : 1</p>              <button onClick={() => handleAddNode('1')}>ADD</button>              <br />              <button onClick={() => handleDoubleAddNode('1')}>DoubleADD</button>              <br />              <button onClick={() => handleDeleteNode('1')}>DELETE</button>            </div>          ),        },        type: 'default',        position: { x: 650, y: 50 },      },    ]); |

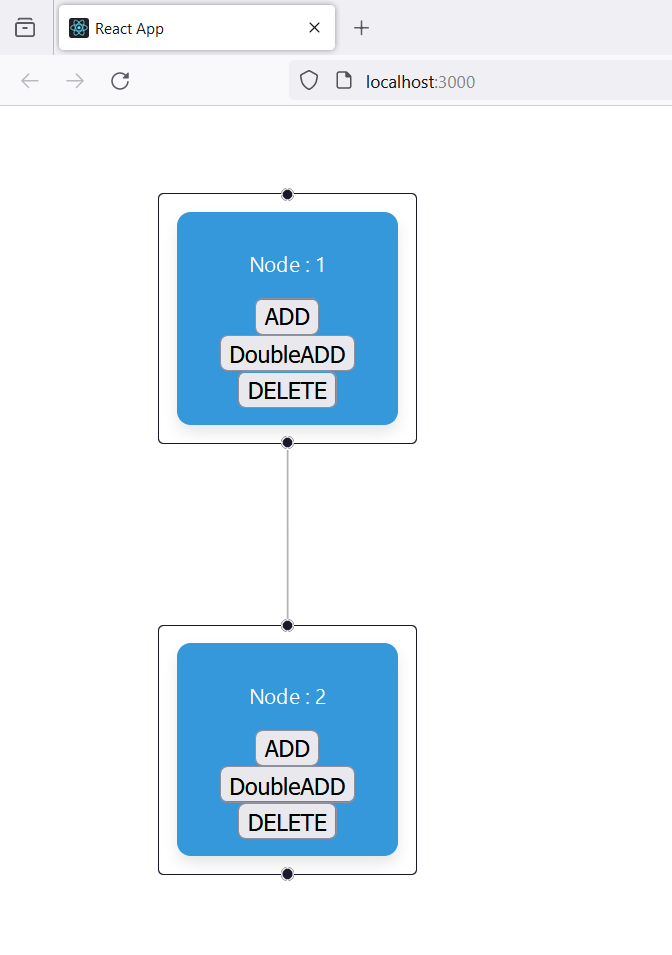
**6)Output:**

****

**7)Write a function for Button (ADD) so When we click on the button ADD it should create a New Node as Node : 2 And Create a edge between the two Nodes.**

|  |
| --- |
| **Code:** |
| const [edges, setEdges] = useState([]);  const handleAddNode = (clickedNodeId) => {      setNodes((prevNodes) => {        const clickedNode = prevNodes.find((node) => node.id === clickedNodeId);        const newNodeId = `${prevNodes.length + 1}`;        const newNode = {          id: newNodeId,          data: {            label: (              <div className='box'>                <p>Node : {newNodeId}</p>                <button onClick={() => handleAddNode(newNodeId)}>ADD</button>                <br />                <button onClick={() => handleDoubleAddNode(newNodeId)}>DoubleADD</button>                <br />                <button onClick={() => handleDeleteNode(newNodeId)}>DELETE</button>              </div>            ),          },          type: 'default',          position: { x: clickedNode.position.x, y: clickedNode.position.y + 250 },        };        const edge = createEdge(clickedNodeId, newNodeId);        setEdges((prevEdges) => [...prevEdges, edge]);        return [...prevNodes, newNode];      });    };    const createEdge = (source, target) => ({      id: `edge-${source}-${target}`,      source,      target,      type: 'step',    }); |

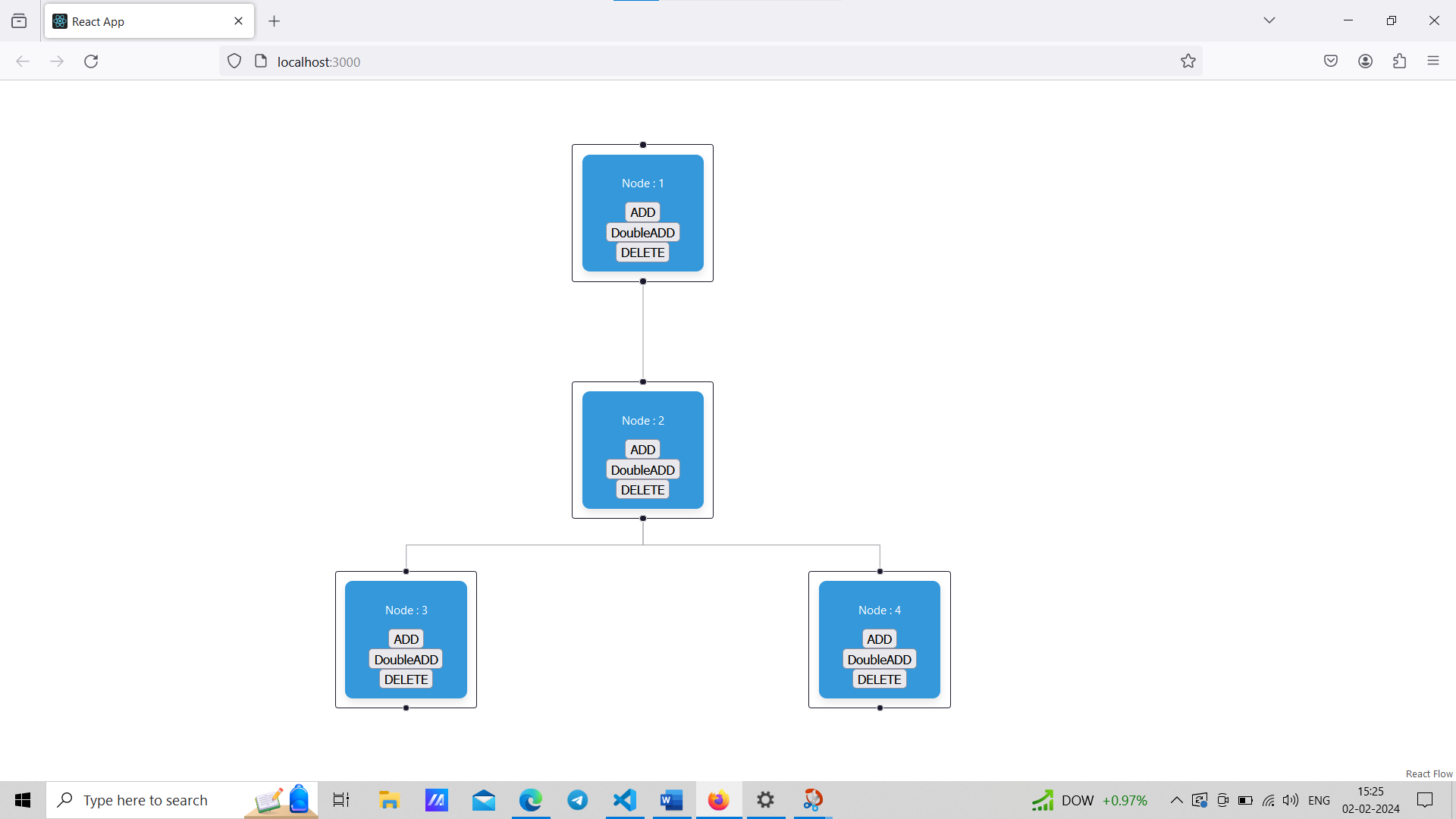
**8)Output:**

****

**9)Write a function for Button (Double ADD) When we click on the button DoubleAdd it should create two New Nodes with In-Line Position. And Connect the edges between the Node and newly Created Nodes.**

|  |
| --- |
| **Code:** |
| const handleDoubleAddNode = (clickedNodeId) => {      setNodes((prevNodes) => {        const clickedNode = prevNodes.find((node) => node.id === clickedNodeId);        const newNodeId = `${prevNodes.length + 1}`;        const newNodeId2 = `${prevNodes.length + 2}`;        const createNode = (id, positionOffsetX) => ({          id,          data: {            label: (              <div className='box'>                <p>Node : {id}</p>                <button onClick={() => handleAddNode(id)}>ADD</button>                <br />                <button onClick={() => handleDoubleAddNode(id)}>DoubleADD</button>                <br />                <button onClick={() => handleDeleteNode(id)}>DELETE</button>              </div>            ),          },          type: 'default',          position: { x: clickedNode.position.x + positionOffsetX, y: clickedNode.position.y + 200 },        });        const newNode1 = createNode(newNodeId, -250);        const newNode2 = createNode(newNodeId2, 250);        const edge1 = createEdge(clickedNodeId, newNodeId);        const edge2 = createEdge(clickedNodeId, newNodeId2);        setEdges((prevEdges) => [...prevEdges, edge1, edge2]);        return [...prevNodes, newNode1, newNode2];      });    }; |

**10)Output:**

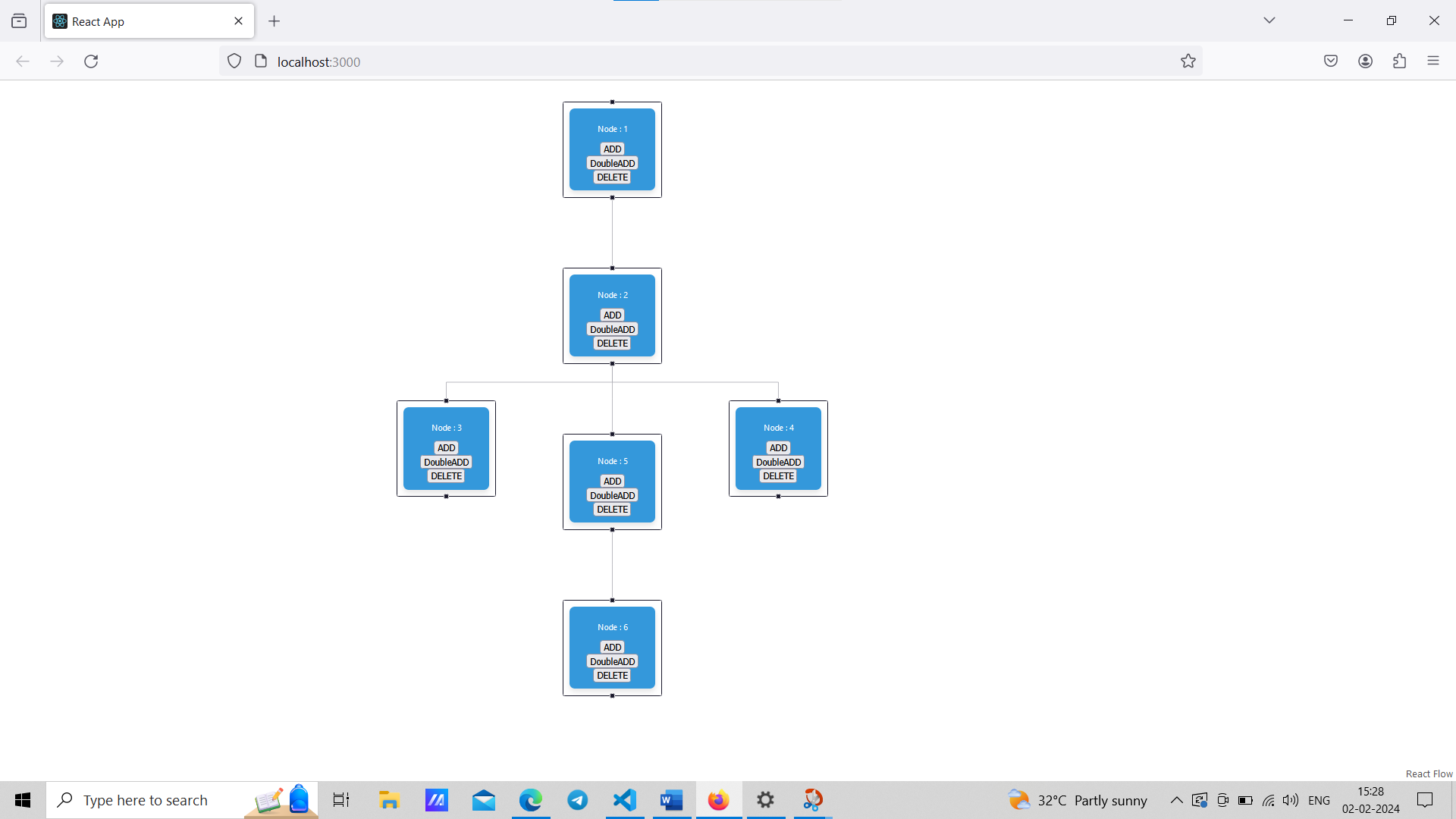
****

**11)Write a function for Button (DELETE) . When we click on the button DELETE then it should delete that Node and Connect the Edges with parent and child Nodes.**

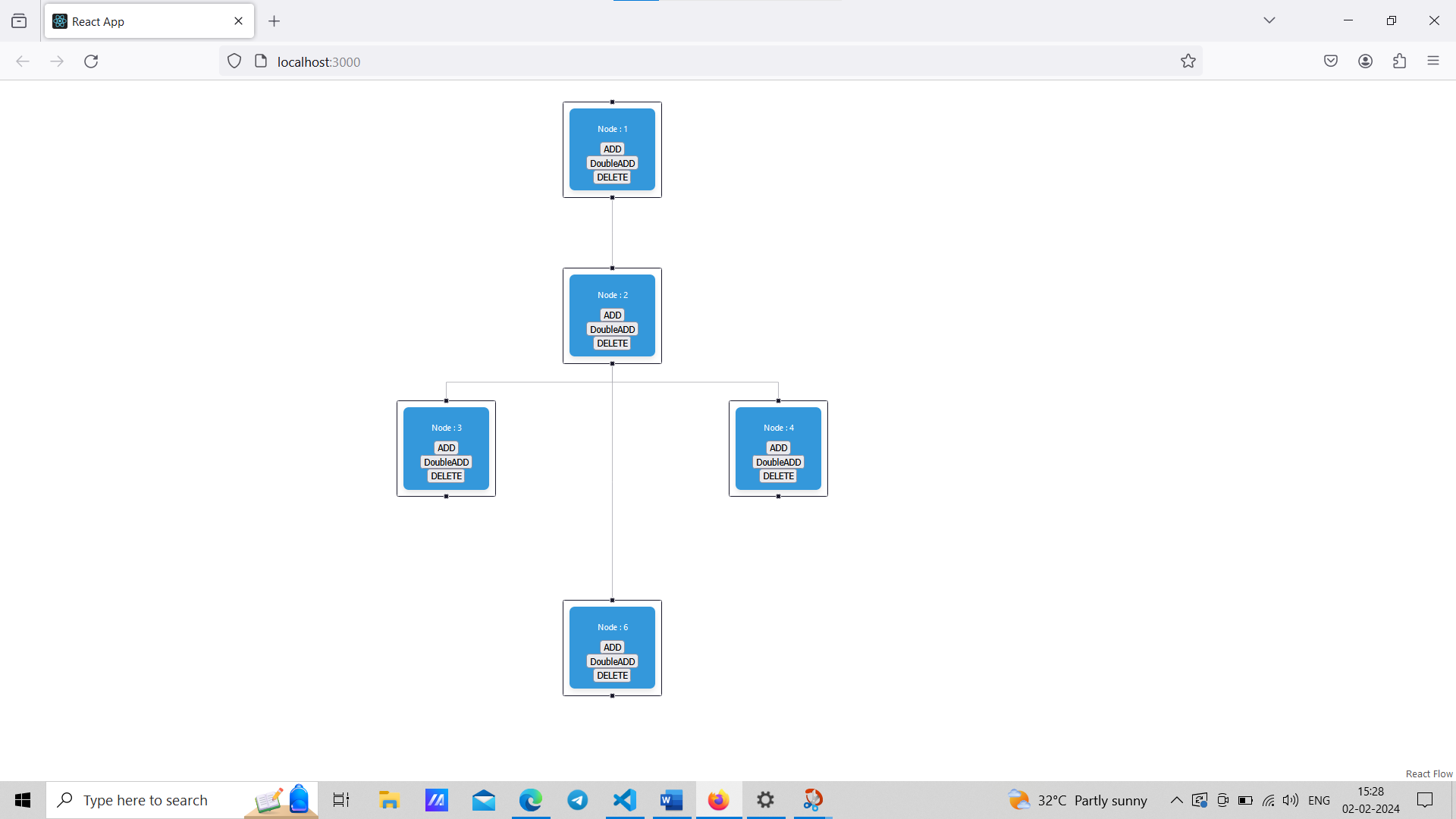
|  |
| --- |
| **Code:** |
| const handleDeleteNode = (deletedNodeId) => {      setNodes((prevNodes) => {        const deletedNode = prevNodes.find((node) => node.id === deletedNodeId);          if (!deletedNode) {          return prevNodes;        }          const updatedNodes = prevNodes.filter((node) => node.id !== deletedNodeId);          setEdges((prevEdges) => {          const incomingEdges = prevEdges.filter((edge) => edge.target === deletedNodeId);          const outgoingEdges = prevEdges.filter((edge) => edge.source === deletedNodeId);            // Filter out edges that are not connected to the deleted node          const otherEdges = prevEdges.filter(            (edge) => edge.source !== deletedNodeId && edge.target !== deletedNodeId          );            const updatedEdges = incomingEdges.concat(            outgoingEdges.map((edge) => ({              ...edge,              source: incomingEdges.length === 1 ? incomingEdges[0].source : edge.source,              target: outgoingEdges.length === 1 ? outgoingEdges[0].target : edge.target,            }))          );            // Combine updated edges with other edges          const finalEdges = updatedEdges.concat(otherEdges);            return finalEdges;        });          return updatedNodes;      });    }; |

**12)OutPut:**

**Before Deleting Node 5 :**

****

**After Deleting Node 5 :**

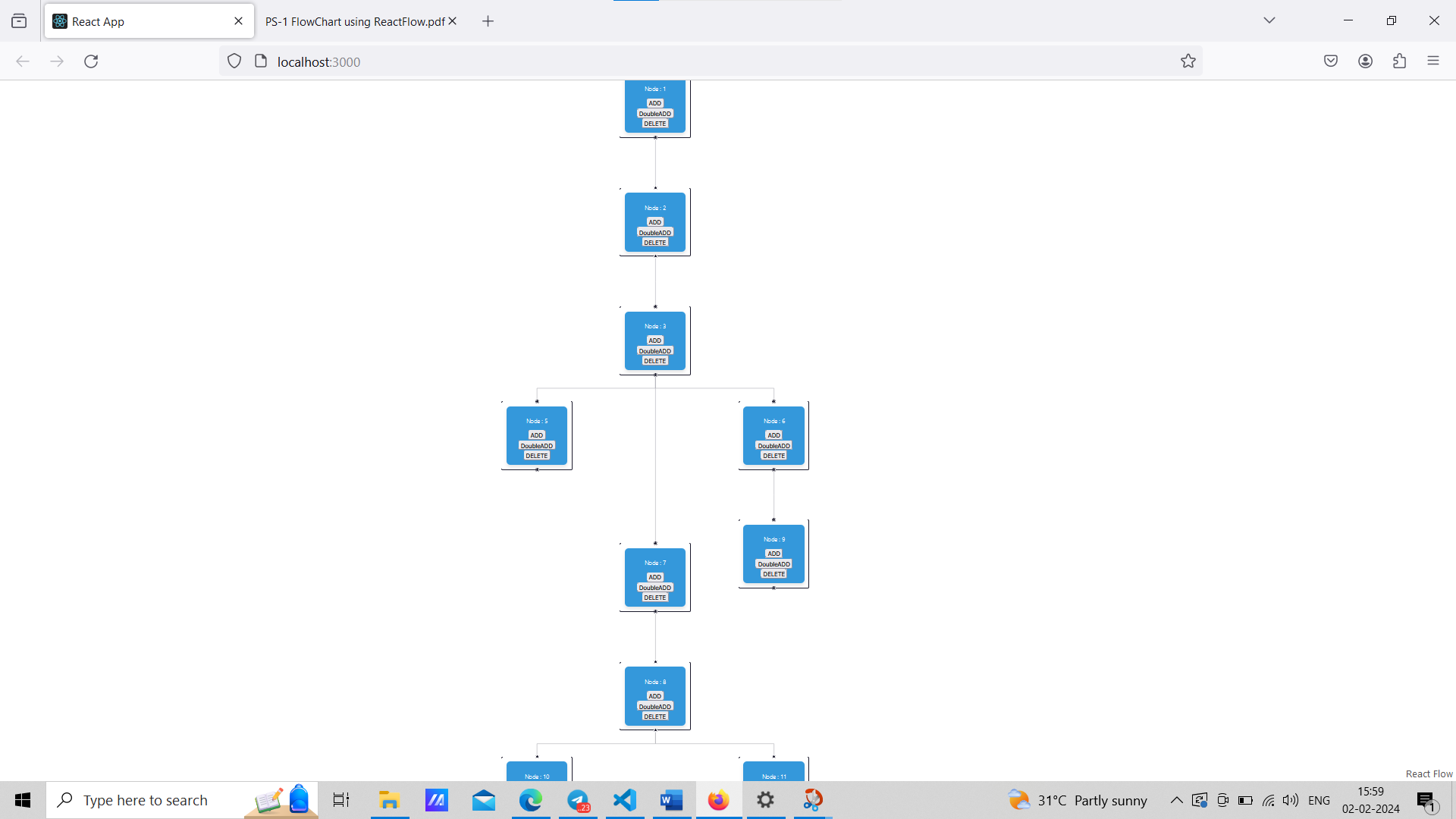
****

**OverAll Code with CSS :**

|  |
| --- |
| **CSS:** |
| /\* App.css \*/  .box {    background-color: #3498db; /\* Set your desired background color \*/    color: #ffffff; /\* Set your desired text color \*/    padding: 10px;    border-radius: 8px;    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);    transition: background-color 0.3s ease-in-out;  }  .box:hover {    background-color: #2980b9; /\* Set your desired hover background color \*/  } |

|  |
| --- |
| **Code:** |
| // App.js  import React, { useState } from 'react';  import ReactFlow from 'react-flow-renderer';  import './App.css';  const App = () => {    const [nodes, setNodes] = useState([      {        id: '1',        data: {          label: (            <div className='box'>              <p>Node : 1</p>              <button onClick={() => handleAddNode('1')}>ADD</button>              <br />              <button onClick={() => handleDoubleAddNode('1')}>DoubleADD</button>              <br />              <button onClick={() => handleDeleteNode('1')}>DELETE</button>            </div>          ),        },        type: 'default',        position: { x: 650, y: 50 },      },    ]);    const [edges, setEdges] = useState([]);    const handleDoubleAddNode = (clickedNodeId) => {      setNodes((prevNodes) => {        const clickedNode = prevNodes.find((node) => node.id === clickedNodeId);        const newNodeId = `${prevNodes.length + 1}`;        const newNodeId2 = `${prevNodes.length + 2}`;        const createNode = (id, positionOffsetX) => ({          id,          data: {            label: (              <div className='box'>                <p>Node : {id}</p>                <button onClick={() => handleAddNode(id)}>ADD</button>                <br />                <button onClick={() => handleDoubleAddNode(id)}>DoubleADD</button>                <br />                <button onClick={() => handleDeleteNode(id)}>DELETE</button>              </div>            ),          },          type: 'default',          position: { x: clickedNode.position.x + positionOffsetX, y: clickedNode.position.y + 200 },        });        const newNode1 = createNode(newNodeId, -250);        const newNode2 = createNode(newNodeId2, 250);        const edge1 = createEdge(clickedNodeId, newNodeId);        const edge2 = createEdge(clickedNodeId, newNodeId2);        setEdges((prevEdges) => [...prevEdges, edge1, edge2]);        return [...prevNodes, newNode1, newNode2];      });    };    const handleAddNode = (clickedNodeId) => {      setNodes((prevNodes) => {        const clickedNode = prevNodes.find((node) => node.id === clickedNodeId);        const newNodeId = `${prevNodes.length + 1}`;        const newNode = {          id: newNodeId,          data: {            label: (              <div className='box'>                <p>Node : {newNodeId}</p>                <button onClick={() => handleAddNode(newNodeId)}>ADD</button>                <br />                <button onClick={() => handleDoubleAddNode(newNodeId)}>DoubleADD</button>                <br />                <button onClick={() => handleDeleteNode(newNodeId)}>DELETE</button>              </div>            ),          },          type: 'default',          position: { x: clickedNode.position.x, y: clickedNode.position.y + 250 },        };        const edge = createEdge(clickedNodeId, newNodeId);        setEdges((prevEdges) => [...prevEdges, edge]);        return [...prevNodes, newNode];      });    };    const createEdge = (source, target) => ({      id: `edge-${source}-${target}`,      source,      target,      type: 'step',    });      const handleDeleteNode = (deletedNodeId) => {      setNodes((prevNodes) => {        const deletedNode = prevNodes.find((node) => node.id === deletedNodeId);          if (!deletedNode) {          return prevNodes;        }          const updatedNodes = prevNodes.filter((node) => node.id !== deletedNodeId);          setEdges((prevEdges) => {          const incomingEdges = prevEdges.filter((edge) => edge.target === deletedNodeId);          const outgoingEdges = prevEdges.filter((edge) => edge.source === deletedNodeId);            // Filter out edges that are not connected to the deleted node          const otherEdges = prevEdges.filter(            (edge) => edge.source !== deletedNodeId && edge.target !== deletedNodeId          );            const updatedEdges = incomingEdges.concat(            outgoingEdges.map((edge) => ({              ...edge,              source: incomingEdges.length === 1 ? incomingEdges[0].source : edge.source,              target: outgoingEdges.length === 1 ? outgoingEdges[0].target : edge.target,            }))          );            // Combine updated edges with other edges          const finalEdges = updatedEdges.concat(otherEdges);            return finalEdges;        });          return updatedNodes;      });    };        return (      <div style={{ height: '100vh', width: '100%' }}>        <ReactFlow  nodes={nodes} edges={edges} elements={nodes.concat(edges)} />      </div>    );  };  export default App; |

**Output:**

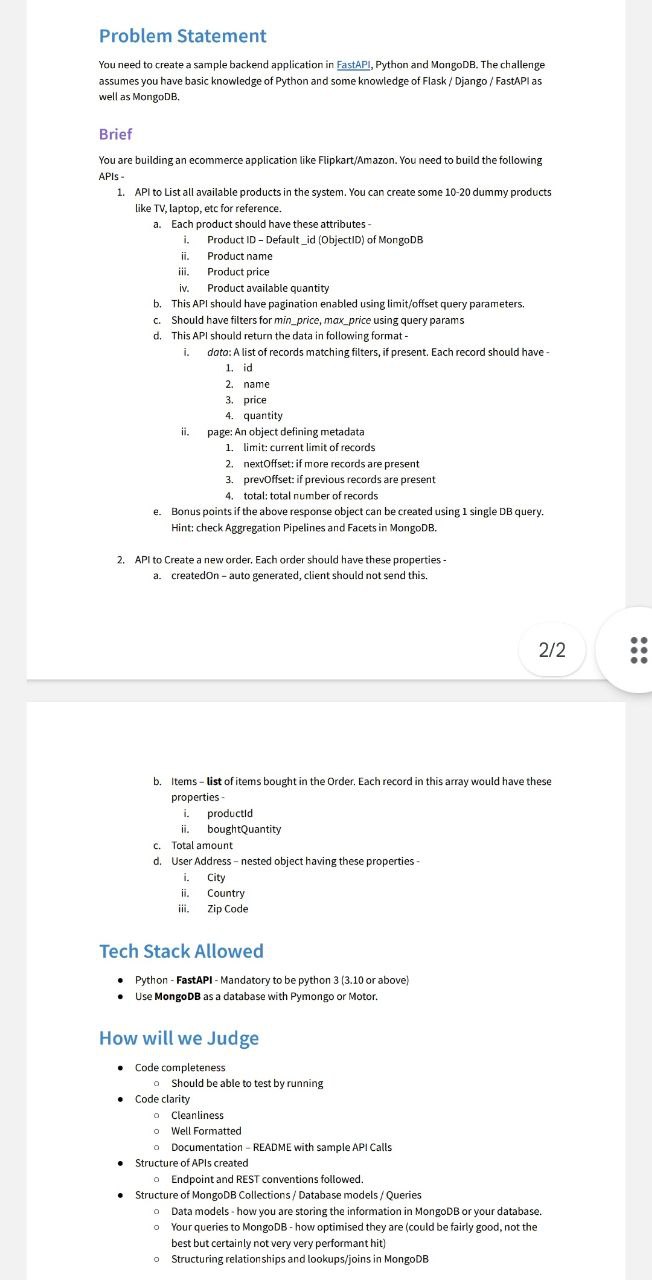
****

**Conclusion:**

**In summary, the ReactJS application leveraging React Flow provides a user-friendly flowchart building experience. Users can effortlessly add single or branched nodes, delete nodes, and maintain clear parent-child connections. The application showcases a visually appealing design achieved through CSS styling.**

**BACKEND :**

**PROBLEM STATEMENT -2 USING PYTHON AND MONGODB**

****

**SETUP:**

**1)Visual Studio Code**

**2)FLASK**

**3)MONGODB**

**4)MonogoDBCompass**

**COMMANDS:**

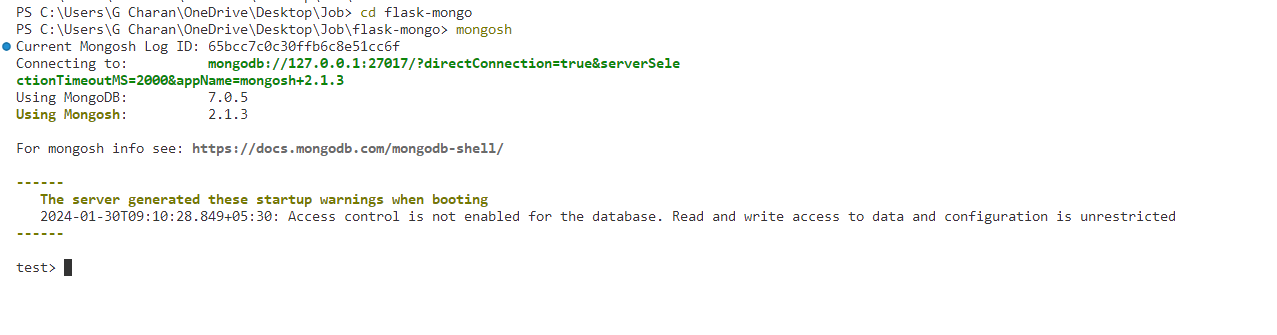
**1)To install env into your folder : pip install pipenv**

**2)To install fask : pipevn install flask**

**3)To install PyMongo : pip install pymongo**

**Process:**

**1)First Connecting to the mong(database) we use command mongosh in the terminal**

****

**2)Add 10-20 no.of products to the database with each product should have these attributes :**

**i.Product ID-Default\_id (ObjectID) of MongoDB**

**ii. Product name**

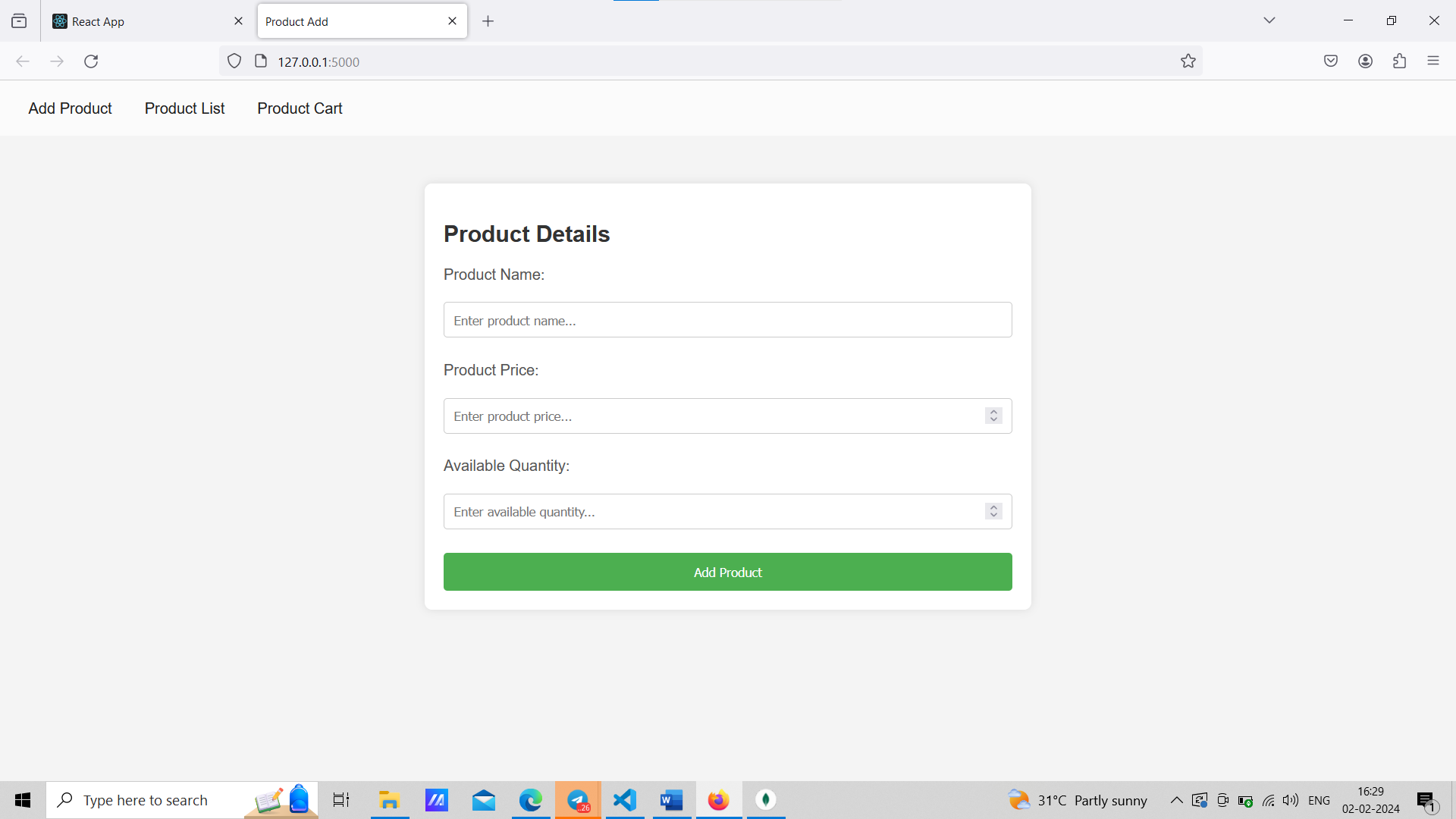
**iii.Product price**

**iv. Product available quantity**

**Product.HTML Code:**

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">        <link rel="stylesheet" href="{{url\_for('static', filename='prostyles.css')}}"/>      <title>Product Add</title>  </head>  <body>      {% include 'navbar.html' %}      <div class="product">          <h2>Product Details</h2>          <form method="POST" action="{{ url\_for('product') }}">              <label for="product-name">Product Name:</label>              <input type="text" id="product-name" name="product-name" placeholder="Enter product name..." required>              <label for="product-price">Product Price:</label>              <input type="number" id="product-price" name="product-price" placeholder="Enter product price..." required>              <label for="product-quantity">Available Quantity:</label>              <input type="number" id="product-quantity" name="product-quantity" placeholder="Enter available quantity..." required>              <button type="submit">Add Product</button>          </form>      </div>  </body>  </html> |

**Output: To add products to the Data base with 4 attributes**

****

**Flask Code :**

|  |
| --- |
| #Just to add product details  @app.route("/", methods=['GET','POST'])  def product():      if request.method=='POST':          product\_name = request.form['product-name']          product\_price = request.form['product-price']          product\_quantity = request.form['product-quantity']          todos.insert\_one({              'product\_name': product\_name,              'product\_price': product\_price,              'product\_quantity': product\_quantity          })          return redirect(url\_for('product'))      all\_todos=todos.find()      return render\_template('product.html',todos=all\_todos) |

**3)Pagination for the List of Products :**

|  |
| --- |
| **App.py:** |
| #To view the Data using pagination  @app.route("/List", methods=['GET'])  def product\_list():      try:          # Set default values for limit and offset          limit = int(request.args.get("limit", 10))          offset = int(request.args.get("offset", 0))            # Fetch products with pagination          products = list(todos.find().skip(offset).limit(limit))          return render\_template('plist.html', products=products,offset=offset)      except Exception as e:          return render\_template('error.html', error=str(e)) |
| **Plist.HTML:** |
| <!DOCTYPE html>  <html lang="en">  <head>      <meta charset="UTF-8">      <meta name="viewport" content="width=device-width, initial-scale=1.0">      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">      <link rel="stylesheet" href="{{ url\_for('static', filename='plist.css') }}"/>      <title>Product List</title>  </head>  <body>      {% include 'navbar.html' %}      <div class="container">          <h1>Product List</h1>          <!-- Price Filters -->          <div class="price-filters">           <form method="POST" action="{{ url\_for('product\_filter') }}">              <label for="min-price">Min Price:</label>              <input type="number" id="min-price" name="min\_price" placeholder="Enter Min Price">              <label for="max-price">Max Price:</label>              <input type="number" id="max-price" name="max\_price" placeholder="Enter Max Price">              <button type="submit">Apply Filter</button>          </form>          </div>          <br>          <!-- Product Table -->          <table>              <thead>                  <tr>                      <th>Product ID</th>                      <th>Product Name</th>                      <th>Price</th>                      <th>Available Quantity</th>                      <th>Action</th>                  </tr>              </thead>              <tbody>                  {% for product in products %}                      <tr class="product-row">                          <td>{{product['\_id']}} </td>                          <td>{{ product['product\_name'] }}</td>                          <td>${{ product['product\_price'] }}</td>                          <td>{{ product['product\_quantity'] }}</td>                          <td>                              <form action="{{ url\_for('addtocart', product\_id=product['\_id']) }}" method="POST">                                  <label for="quantity">Quantity:</label>                                  <input type="number" id="quantity" name="quantity" placeholder="Enter quantity" min="1" required>                                  <button type="submit" onclick="return confirm('Are you sure wwant to Add this iteam to cart')">Add to Cart</button>                              </form>                          </td>                      </tr>                    {% endfor %}              </tbody>          </table>          <!-- Pagination -->          <div class="pagination">              {% if offset > 0 %}                  <a href="{{ url\_for('product\_list', limit=10, offset=offset-10) }}" class="page-link">&laquo; Previous</a>              {% endif %}                <a href="{{ url\_for('product\_list', limit=10, offset=offset+10) }}" class="page-link">Next &raquo;</a>          </div>      </div>  </body>  </html> |
| **Output:** |
|  |
|  |

**4)Filter using MIN\_PRICE AND MAX\_PRICE using query params:**

|  |
| --- |
| **App.py:**  **Sample API :** [**http://127.0.0.1:5000/List?limit=10&offset=10**](http://127.0.0.1:5000/List?limit=10&offset=10) |
| #filter by minprice and max price  @app.route("/filter", methods=['POST'])  def product\_filter():      min\_price=request.form['min\_price']      max\_price=request.form['max\_price']      offset =0      filter={"product\_price": {"$gte": float(min\_price),"$lte":float(max\_price)}}      products=todos.find(filter)      return render\_template('plist.html',products=products,offset=offset) |
| **Output:** |
|  |
|  |

**5)Create a Order API to Create a new order. Each order should have these properties-**

**a. createdOn-auto generated, client should not send this.**

**b. Items list of items bought in the Order. Each record in this array would have these properties-**

**i.productid.**

**ii.brought-quantity.**

**c. Total amount**

**d. User Address-nested object having these properties**

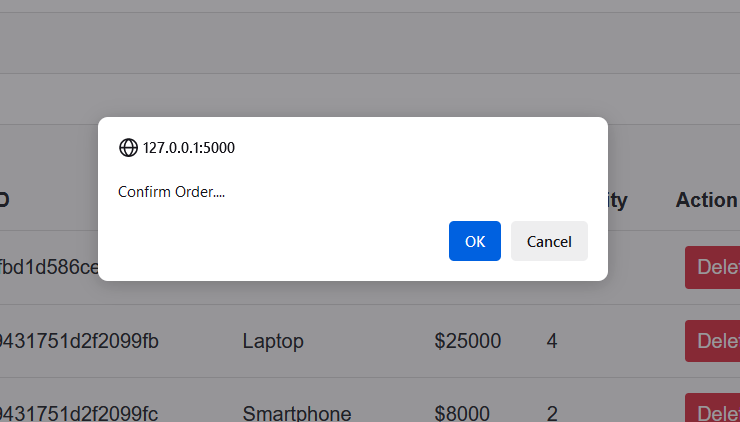
**I.City**

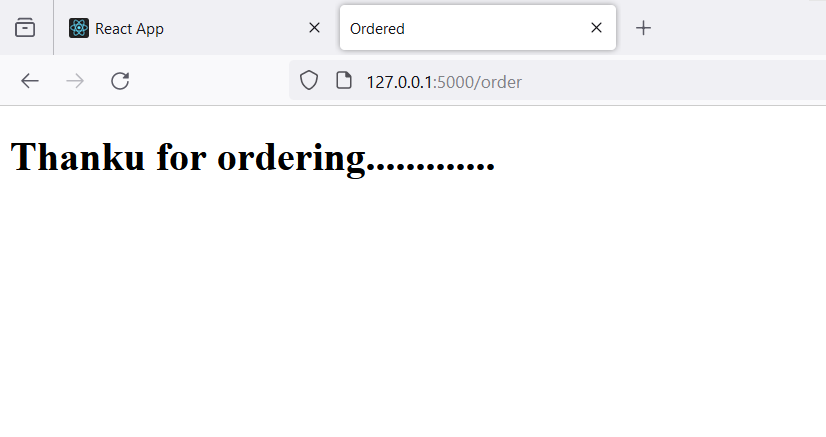
**ii.Country**

**iii. Zip Code**

|  |
| --- |
| **App.py:** |
| #To send product with no of quantities to database  @app.post("/<product\_id>/addtocart/")  def addtocart(product\_id):      try:          # Try to fetch product details using ObjectId          product\_details = todos.find\_one({"\_id": ObjectId(product\_id)})          bq=request.form['quantity']          # If product\_details is not found using ObjectId, try using string representation          if not product\_details:              product\_details = todos.find\_one({"\_id": product\_id})            if product\_details:              # Add product details to the order collection              cart.insert\_one({                  '\_id':product\_id,                  'product\_name': product\_details['product\_name'],                  'product\_price': product\_details['product\_price'],                  'brought\_quantity':bq              })              return redirect(url\_for('product\_list'))          else:              return f"Product with ID {product\_id} not found"      except Exception as e:          return str(e)  #To get the total amount  @app.route("/cart", methods=['GET'])  def product\_cart():      try:          products = list(cart.find())          total\_price = sum(float(item['product\_price']) \* int(item['brought\_quantity']) for item in cart.find())          return render\_template('productcart.html', products=products,tp=total\_price)      except Exception as e:          return render\_template('error.html', error=str(e)) |
| **Output: Give no of Quantities required and click on Add to cart.** |
|  |
|  |
| **Here, We can See the no of Products we added to card with total amount.** |
|  |

**6)Click on Place Order then It will collect user Address and all products carts and total amount into the database.**

****

****

**DataBase:**

**It will store with properties ID,Created On , Adress(city,country,zip\_code),Products(no of products in the cart),Total Price.**

****

**APP.py: Full Code:**

from flask import Flask,render\_template,url\_for,request,redirect

from pymongo import MongoClient

from bson import ObjectId

from datetime import datetime

app = Flask(\_\_name\_\_)

client = MongoClient('localhost',27017)

#Just to add product details

@app.route("/", methods=['GET','POST'])

def product():

    if request.method=='POST':

        product\_name = request.form['product-name']

        product\_price = request.form['product-price']

        product\_quantity = request.form['product-quantity']

        todos.insert\_one({

            'product\_name': product\_name,

            'product\_price': product\_price,

            'product\_quantity': product\_quantity

        })

        return redirect(url\_for('product'))

    all\_todos=todos.find()

    return render\_template('product.html',todos=all\_todos)

#To view the Data using pagination

@app.route("/List", methods=['GET'])

def product\_list():

    try:

        # Set default values for limit and offset

        limit = int(request.args.get("limit", 10))

        offset = int(request.args.get("offset", 0))

        # Fetch products with pagination

        products = list(todos.find().skip(offset).limit(limit))

        return render\_template('plist.html', products=products,offset=offset)

    except Exception as e:

        return render\_template('error.html', error=str(e))

#filter by minprice and max price

@app.route("/filter", methods=['POST'])

def product\_filter():

    min\_price=request.form['min\_price']

    max\_price=request.form['max\_price']

    offset =0

    filter={"product\_price": {"$gte": float(min\_price),"$lte":float(max\_price)}}

    products=todos.find(filter)

    return render\_template('plist.html',products=products,offset=offset)

#To send product with no of quantities to database

@app.post("/<product\_id>/addtocart/")

def addtocart(product\_id):

    try:

        # Try to fetch product details using ObjectId

        product\_details = todos.find\_one({"\_id": ObjectId(product\_id)})

        bq=request.form['quantity']

        # If product\_details is not found using ObjectId, try using string representation

        if not product\_details:

            product\_details = todos.find\_one({"\_id": product\_id})

        if product\_details:

            # Add product details to the order collection

            cart.insert\_one({

                '\_id':product\_id,

                'product\_name': product\_details['product\_name'],

                'product\_price': product\_details['product\_price'],

                'brought\_quantity':bq

            })

            return redirect(url\_for('product\_list'))

        else:

            return f"Product with ID {product\_id} not found"

    except Exception as e:

        return str(e)

@app.route("/cart", methods=['GET'])

def product\_cart():

    try:

        products = list(cart.find())

        total\_price = sum(float(item['product\_price']) \* int(item['brought\_quantity']) for item in cart.find())

        return render\_template('productcart.html', products=products,tp=total\_price)

    except Exception as e:

        return render\_template('error.html', error=str(e))

@app.route("/order", methods=['POST'])

def order():

    try:

        # Get address details from the form

        city = request.form['city']

        country = request.form['country']

        zip\_code = request.form['zip\_code']

        # Get product details from the form

        products = []

        total\_price = 0

        for product in cart.find():

            product\_id = str(product['\_id'])

            brought\_quantity = int(product['brought\_quantity'])

            if brought\_quantity > 0:

                # Calculate the total price for each product

                product\_price = float(product['product\_price'])

                total\_price += product\_price \* brought\_quantity

                # Add product details to the order collection

                products.append({

                    'product\_id': product\_id,

                    'product\_name': product['product\_name'],

                    'product\_price': product\_price,

                    'brought\_quantity': brought\_quantity

                })

        # Add order details to the order collection

        order\_details = {

            'createdOn': datetime.now(),

            'address': {

                'city': city,

                'country': country,

                'zip\_code': zip\_code

            },

            'products': products,

            'total\_price': total\_price

        }

        orders.insert\_one(order\_details)

        # Clear the cart

        cart.delete\_many({})

        return render\_template('ordered.html', order\_details=order\_details)

    except Exception as e:

        return render\_template('error.html', error=str(e))

@app.route('/<product\_id>/delete')

def delete(product\_id):

    pro = cart.find\_one({'\_id':product\_id})

    cart.delete\_one(pro)

    return redirect('/cart')

@app.route("/orderlist", methods=['GET'])

def orderlist():

    try:

        # Fetch products with pagination

        products = orders.find()

        return render\_template('porders.html', order=products)

    except Exception as e:

        return render\_template('error.html', error=str(e))

db = client.product\_database

cart=db.cart

todos = db.todos

orders=db.orders

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

**Conclusion :**

**In conclusion, I have developed an E-Commerce application using Python, Flask, and MongoDB. The application allows users to add products to the database and view a paginated list of products. Additionally, it features filters for maximum and minimum prices. Users can add products to their carts, specifying the desired quantity. The cart section displays the number of products added, along with address fields and the total amount for the selected products. Upon clicking "Place Order," the order is stored in the database with attributes such as ID, creation date, address, list of products, and total amount.**