Web Development-II

ITE-465P

University School of Information and Communication Technology
Guru Gobind Singh Indraprastha University
New Delhi-110078

Submitted by:
MD HUSAIN
B.Tech (CSE-7th Semester)
Enrollment No:06416403222

Submitted to:
Miss Pooja Tyagi

Index

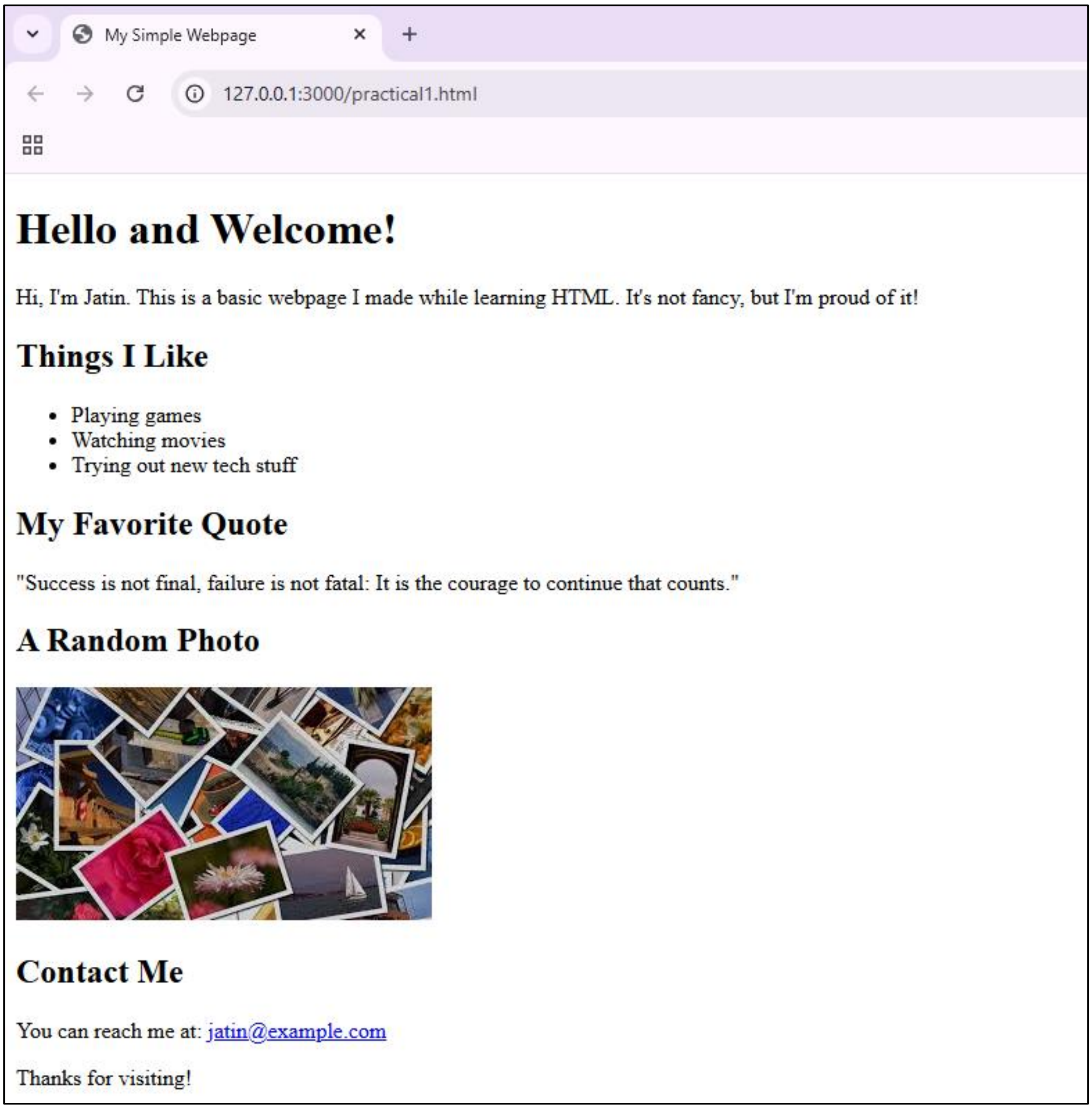
Practical-1

AIM: Using various HTML tags for creating different web pages.

Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>My Simple Webpage</title>
</head>
<body>

  <h1>Hello and Welcome!</h1>

  <p>Hi, I'm Jatin. This is a basic webpage I made while learning HTML. It's not fancy, but I'm proud of it!</p>

  <h2>Things I Like</h2>
  <ul>
    <li>Playing games</li>
    <li>Watching movies</li>
    <li>Trying out new tech stuff</li>
  </ul>

  <h2>My Favorite Quote</h2>
  <p>"Success is not final, failure is not fatal: It is the courage to continue that counts."</p>

  <h2>A Random Photo</h2>
  <img src="https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcQkdPFA6r_IbzQJcyXrKT5TSritv0S_iWwFmw&s" alt="A random image">

  <h2>Contact Me</h2>
  <p>You can reach me at: <a href="mailto:jatin@example.com">jatin@example.com</a></p>

  <p>Thanks for visiting!</p>

</body>
</html>
```

Output:



Hello and Welcome!

Hi, I'm Jatin. This is a basic webpage I made while learning HTML. It's not fancy, but I'm proud of it!

Things I Like

- Playing games
- Watching movies
- Trying out new tech stuff

My Favorite Quote

"Success is not final, failure is not fatal: It is the courage to continue that counts."

A Random Photo

Contact Me

You can reach me at: jatin@example.com

Thanks for visiting!

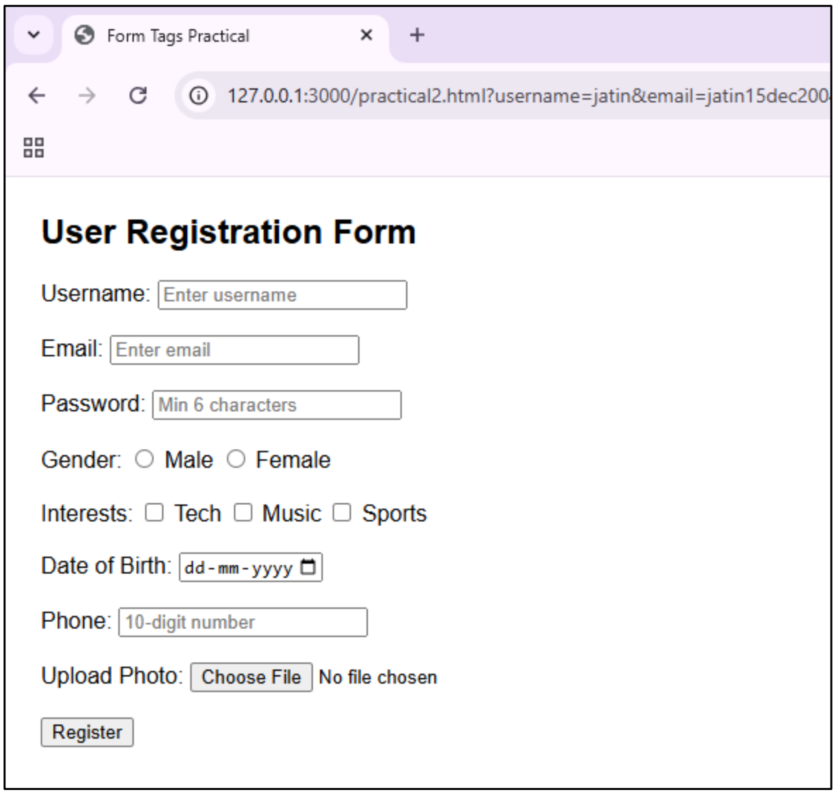
Practical-2

AIM: Using various Form tags for interactivity, authentication, date validation etc.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Form Tags Practical</title>
</head>
<body style="font-family: Arial; margin: 25px;">
  <h2>User Registration Form</h2>
  <form>
    <!-- Authentication -->
    <label>Username:</label>
    <input type="text" name="username" required placeholder="Enter username"><br><br>

    <label>Email:</label>
    <input type="email" name="email" required placeholder="Enter email"><br><br>

    <label>Password:</label>
    <input type="password" name="password" required minlength="6" placeholder="Min 6
characters"><br><br>

    <!-- Interactivity -->
    <label>Gender:</label>
    <input type="radio" name="gender" value="male"> Male
    <input type="radio" name="gender" value="female"> Female<br><br>

    <label>Interests:</label>
    <input type="checkbox" name="tech"> Tech
    <input type="checkbox" name="music"> Music
    <input type="checkbox" name="sports"> Sports<br><br>

    <!-- Date validation -->
    <label>Date of Birth:</label>
    <input type="date" name="dob" required min="1990-01-01" max="2025-12-31"><br><br>

    <!-- Other input types -->
    <label>Phone:</label>
    <input type="tel" name="phone" pattern="[0-9]{10}" placeholder="10-digit number" required><br><br>

    <label>Upload Photo:</label>
    <input type="file" name="photo"><br><br>

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

Output:


Form Tags Practical

127.0.0.1:3000/practical2.html?username=jatin&email=jatin15dec200

User Registration Form

Username: Enter username

Email: Enter email

Password: Min 6 characters

Gender: ○ Male ○ Female

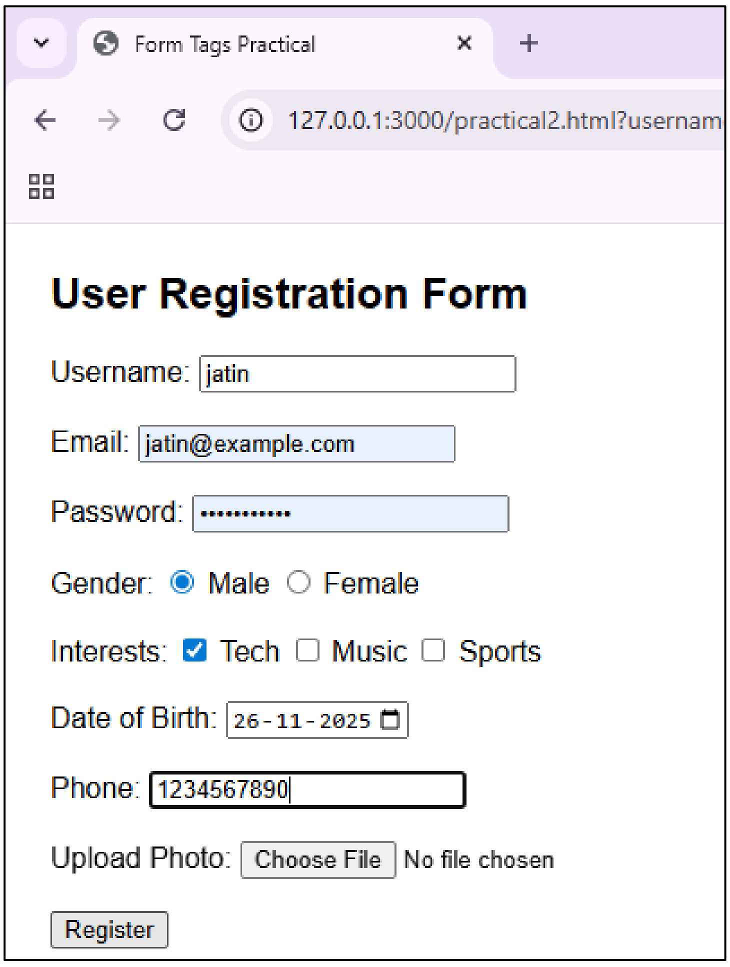
Interests: ☐ Tech ☐ Music ☐ Sports

Date of Birth: dd-mm-yyyy

Phone: 10-digit number

Upload Photo: Choose File No file chosen

Register


Form Tags Practical

127.0.0.1:3000/practical2.html?usernam

User Registration Form

Username: jatin

Email: jatin@example.com

Password: •••••••••••

Gender: [✓ Male] ○ Female

Interests: [✓ Tech] ☐ Music ☐ Sports

Date of Birth: 26-11-2025

Phone: 1234567890

Upload Photo: Choose File No file chosen

Register

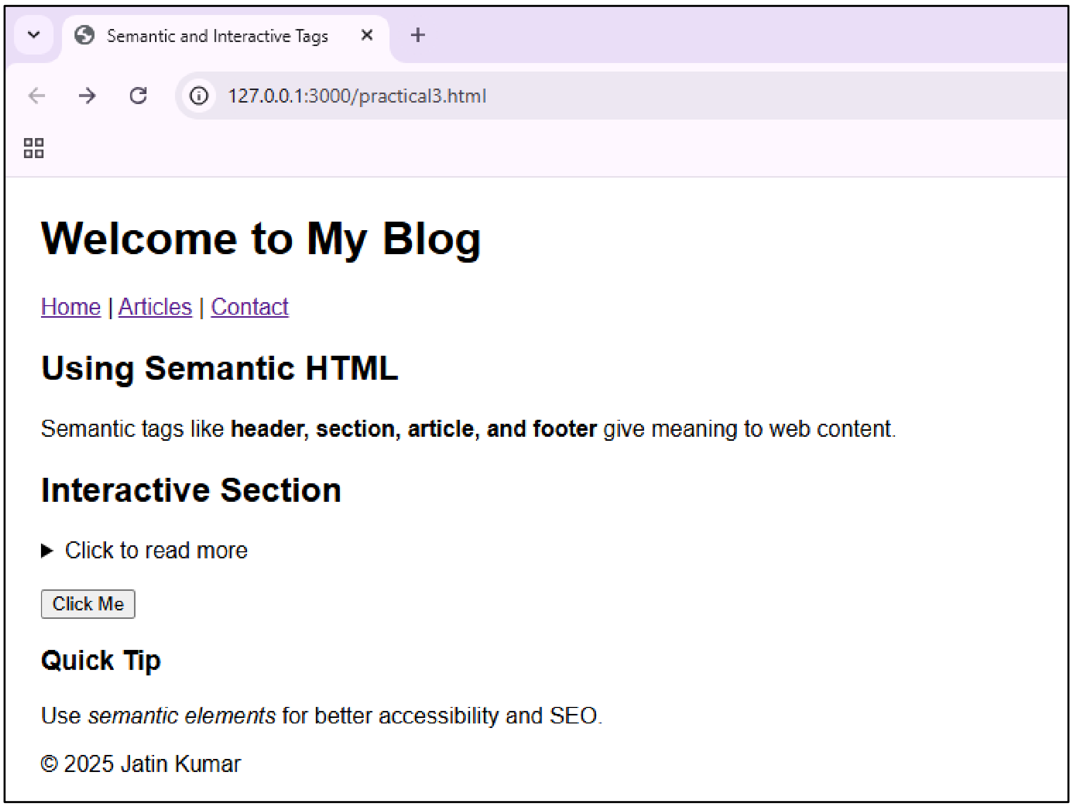
Practical-3

AIM: Using Semantic HTML tags /tags associated with interactivity.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Semantic and Interactive Tags</title>
</head>
<body style="font-family: Arial; margin: 25px;">

  <header>
    <h1>Welcome to My Blog</h1>
    <nav>
      <a href="#">Home</a> |
      <a href="#">Articles</a> |
      <a href="#">Contact</a>
    </nav>
  </header>

  <section>
    <article>
      <h2>Using Semantic HTML</h2>
      <p>Semantic tags like <strong>header, section, article, and footer</strong> give meaning to web
content.</p>
    </article>

    <article>
      <h2>Interactive Section</h2>
      <details>
        <summary>Click to read more</summary>
        <p>This is hidden text revealed when the user clicks on the summary tag — an example of
interactivity.</p>
      </details>
      <br>
      <button onclick="alert('Button clicked!')">Click Me</button>
    </article>
  </section>

  <aside>
    <h3>Quick Tip</h3>
    <p>Use <em>semantic elements</em> for better accessibility and SEO.</p>
  </aside>

  <footer>
    <p>&copy; 2025 Jatin Kumar</p>
  </footer>
</body>
</html>
```

Output:



Semantic and Interactive Tags

127.0.0.1:3000/practical3.html

Welcome to My Blog

Home | Articles | Contact

Using Semantic HTML

Semantic tags like **header, section, article, and footer** give meaning to web content.

Interactive Section

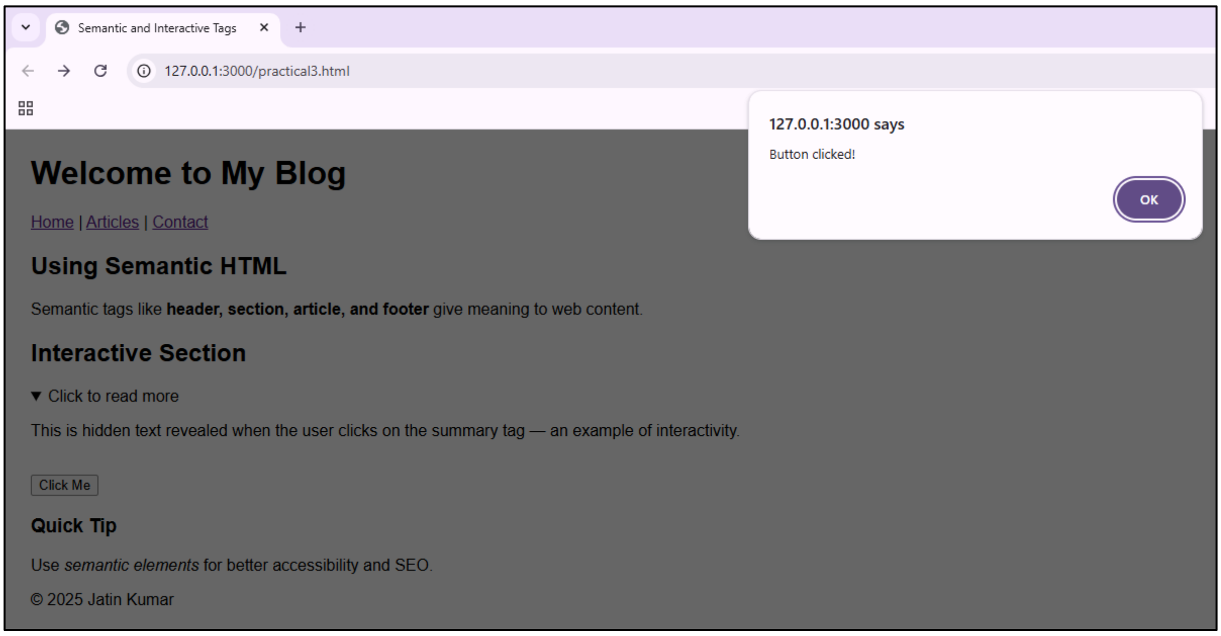
▶ Click to read more

Click Me

Quick Tip

Use *semantic elements* for better accessibility and SEO.

© 2025 Jatin Kumar



Semantic and Interactive Tags

127.0.0.1:3000/practical3.html

127.0.0.1:3000 says

Button clicked!

OK

Welcome to My Blog

Home | Articles | Contact

Using Semantic HTML

Semantic tags like **header, section, article, and footer** give meaning to web content.

Interactive Section

▼ Click to read more

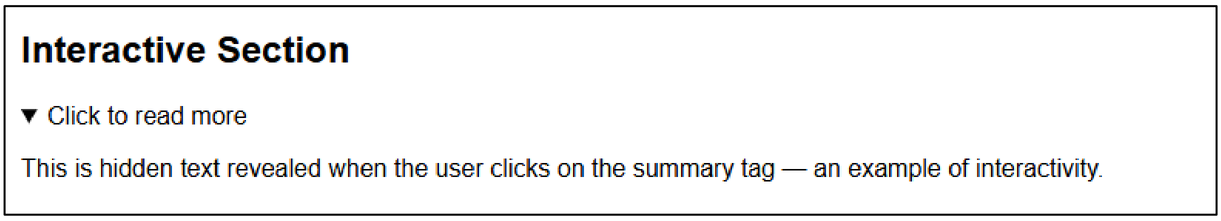
This is hidden text revealed when the user clicks on the summary tag — an example of interactivity.

Click Me

Quick Tip

Use *semantic elements* for better accessibility and SEO.

© 2025 Jatin Kumar



Interactive Section

▼ Click to read more

This is hidden text revealed when the user clicks on the summary tag — an example of interactivity.

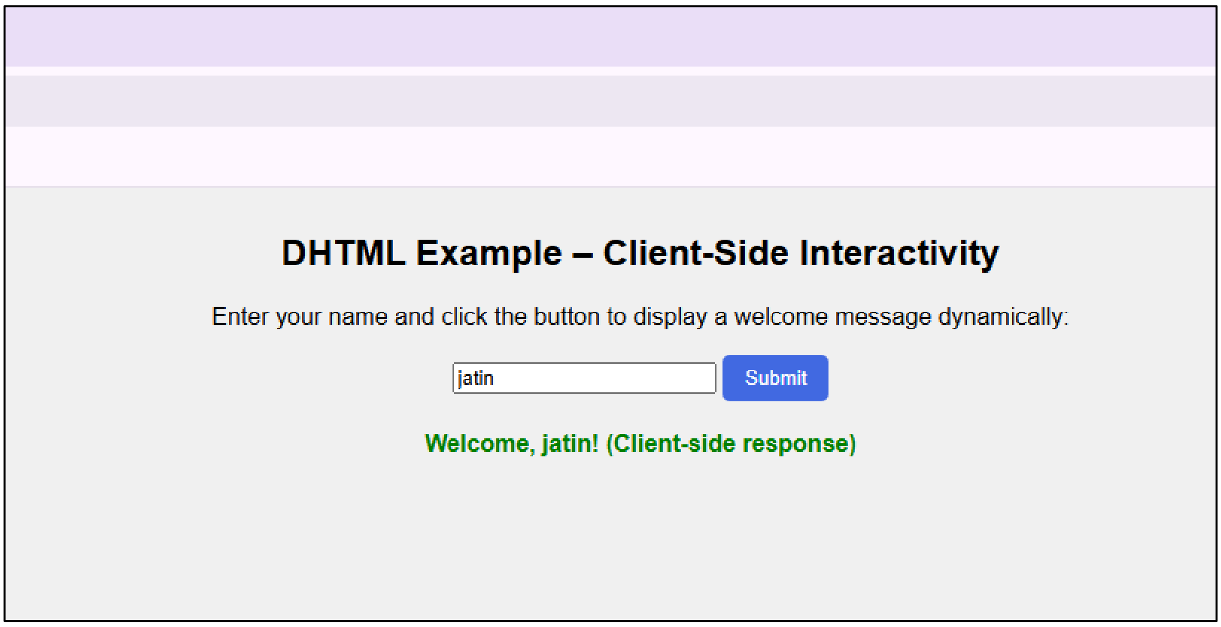
Practical-4

AIM: Using DHTML tags in concern to client server application.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DHTML Example - Client Side Interaction</title>
  <style>
    body {
      font-family: Arial;
      text-align: center;
      margin: 30px;
      background-color: #f0f0f0;
    }
    #msg {
      margin-top: 20px;
      color: darkblue;
      font-weight: bold;
    }
    button {
      background-color: royalblue;
      color: white;
      border: none;
      padding: 8px 15px;
      border-radius: 5px;
      cursor: pointer;
    }
  </style>
</head>
<body>

  <h2>DHTML Example – Client-Side Interactivity</h2>

  <p>Enter your name and click the button to display a welcome message dynamically:</p>

  <input type="text" id="username" placeholder="Enter name">
  <button onclick="showMessage()">Submit</button>

  <p id="msg"></p>

  <script>
    function showMessage() {
      let name = document.getElementById("username").value;
      if (name.trim() === "") {
        document.getElementById("msg").innerHTML = "Please enter your name!";
      } else {
        document.getElementById("msg").innerHTML = "Welcome, " + name + "! (Client-side response)";
        document.getElementById("msg").style.color = "green";
      }
    }
  </script>

</body>
</html>
```

Output:

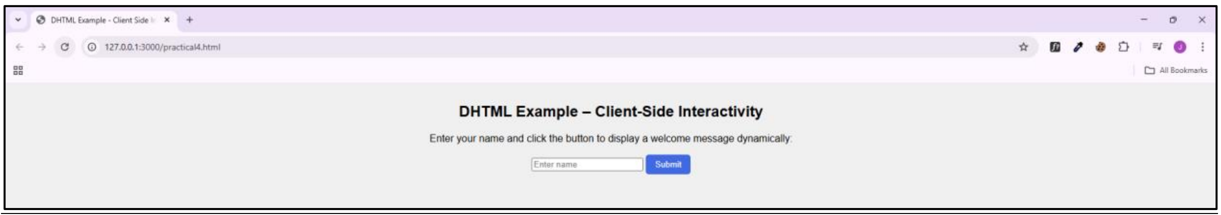


DHTML Example – Client-Side Interactivity

Enter your name and click the button to display a welcome message dynamically:

jatin Submit

Welcome, jatin! (Client-side response)



DHTML Example - Client Side I

127.0.0.1:3000/practical4.html

All Bookmarks

DHTML Example – Client-Side Interactivity

Enter your name and click the button to display a welcome message dynamically:

Enter name Submit

Practical-5

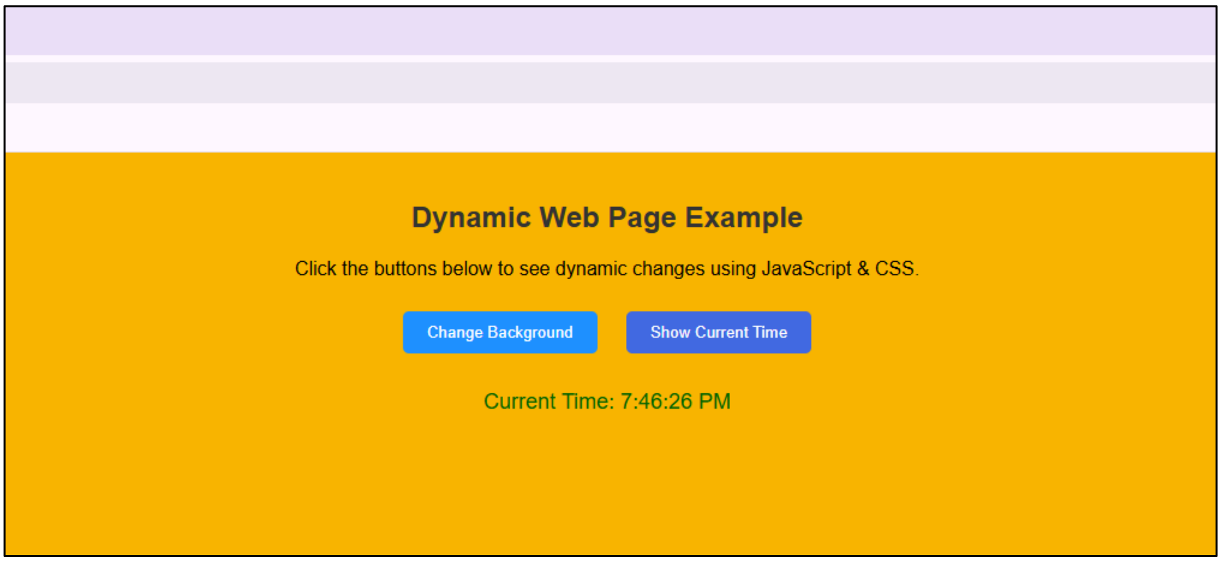
AIM: Using JavaScript/CSS for dynamic web pages.

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dynamic Web Page using JavaScript and CSS</title>
  <style>
    body {
      font-family: Arial;
      text-align: center;
      margin: 40px;
      background-color: #f5f5f5;
      transition: background-color 0.5s ease;
    }

    h2 {
      color: #333;
    }

    button {
      background-color: royalblue;
      color: white;
      border: none;
      padding: 10px 20px;
      border-radius: 5px;
      cursor: pointer;
      margin: 10px;
    }

    button:hover {
      background-color: dodgerblue;
    }

    #output {
      font-size: 18px;
      color: darkgreen;
      margin-top: 20px;
    }
  </style>
</head>
<body>

  <h2>Dynamic Web Page Example</h2>
  <p>Click the buttons below to see dynamic changes using JavaScript & CSS.</p>

  <button onclick="changeColor()">Change Background</button>
```

```
 <button onclick="showTime()">Show Current Time</button>

 <p id="output"></p>

 <script>
  function changeColor() {
   // Random color generator
   const colors = ["#f8b400", "#ff6b6b", "#4ecdc4", "#6a4c93", "#1dd1a1"];
   document.body.style.backgroundColor = colors[Math.floor(Math.random() * colors.length)];
  }

  function showTime() {
   const now = new Date();
   document.getElementById("output").innerHTML =
    "Current Time: " + now.toLocaleTimeString();
  }
 </script>

</body>
</html>
```

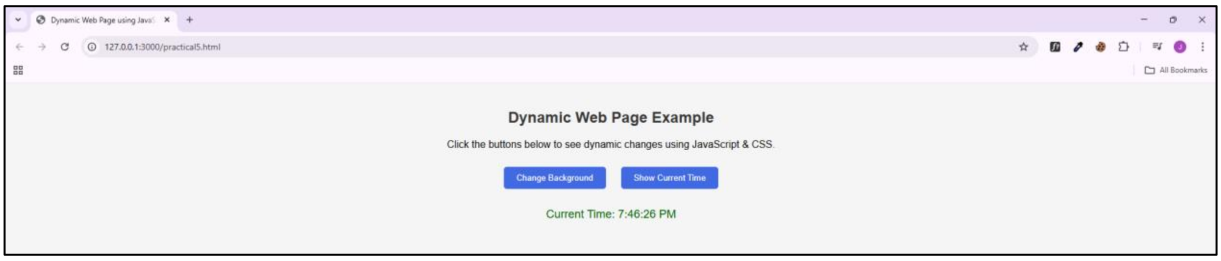
Output:



Dynamic Web Page Example

Click the buttons below to see dynamic changes using JavaScript & CSS.

Current Time: 7:46:26 PM



Dynamic Web Page Example

Click the buttons below to see dynamic changes using JavaScript & CSS.

Current Time: 7:46:26 PM

Practical-6

AIM: Utilize HTML/CSS and JavaScript frameworks (ReactJS, NextJS) to construct dynamic user interfaces.

Code:

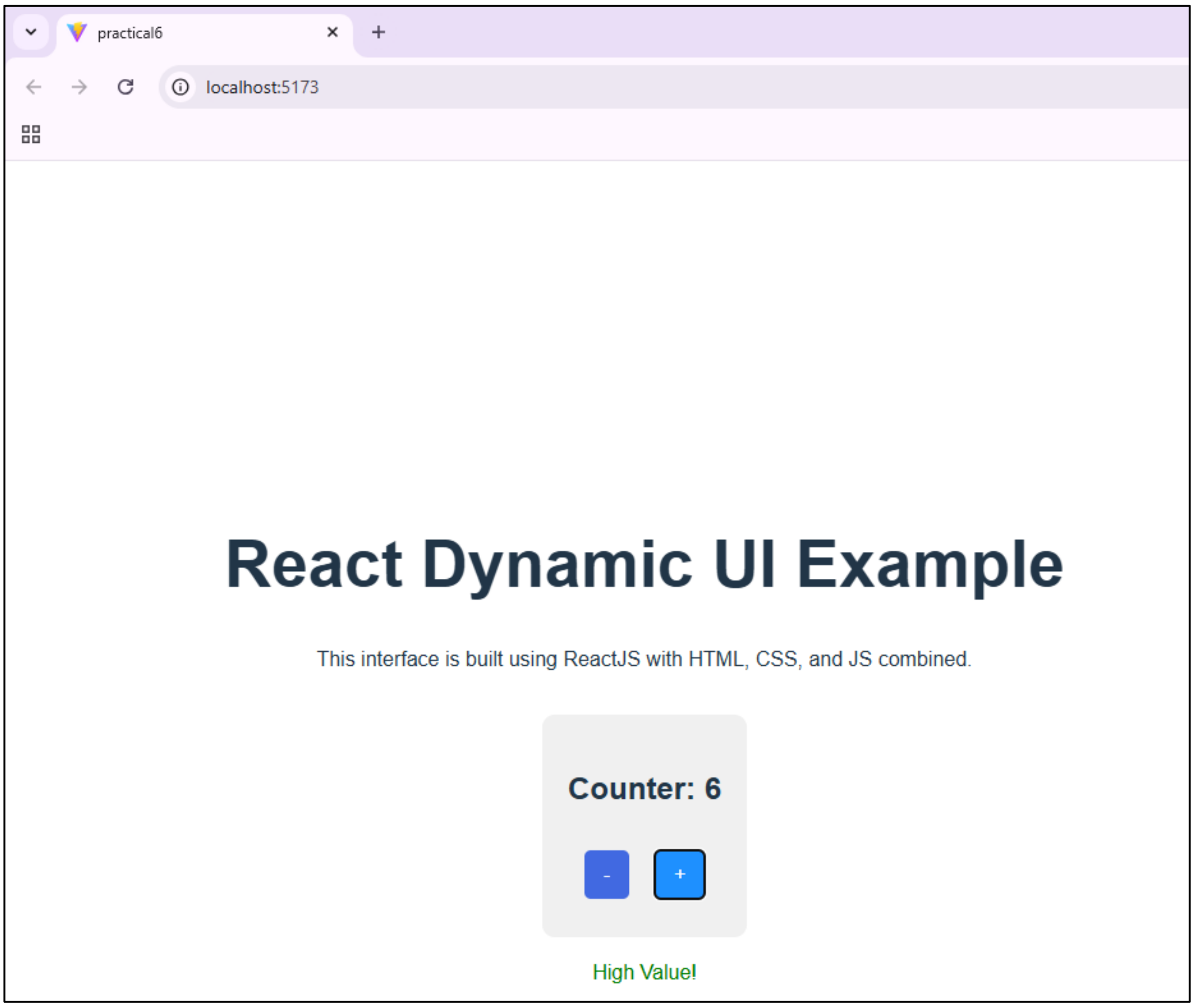
(App.jsx)

```
import React, { useState } from "react";
import "./App.css";

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

  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count > 0 ? count - 1 : 0);

  return (
    <div className="app">
      <h1>React Dynamic UI Example</h1>
      <p>This interface is built using ReactJS with HTML, CSS, and JS combined.</p>

      <div className="counter-box">
        <h2>Counter: {count}</h2>
        <button onClick={decrement}>-</button>
        <button onClick={increment}>+</button>
      </div>

      <p style={{ color: count > 5 ? "green" : "red" }}>
        {count > 5 ? "High Value!" : "Keep Clicking..."}
      </p>
    </div>
  );
}

export default App;
```

(App.css)

```
.app {
  width: 1000px;
  text-align: center;
  font-family: Arial;
  margin-top: 40px;
}

.counter-box {
  background-color: #f0f0f0;
  display: inline-block;
  padding: 20px;
  border-radius: 10px;
  margin-top: 15px;
}

button {
  background-color: royalblue;
  color: white;
  border: none;
  margin: 10px;
  padding: 10px 15px;
  border-radius: 5px;
  cursor: pointer;
}

button:hover {
  background-color: dodgerblue;
}
```

Output:



React Dynamic UI Example

This interface is built using ReactJS with HTML, CSS, and JS combined.

Counter: 6

High Value!

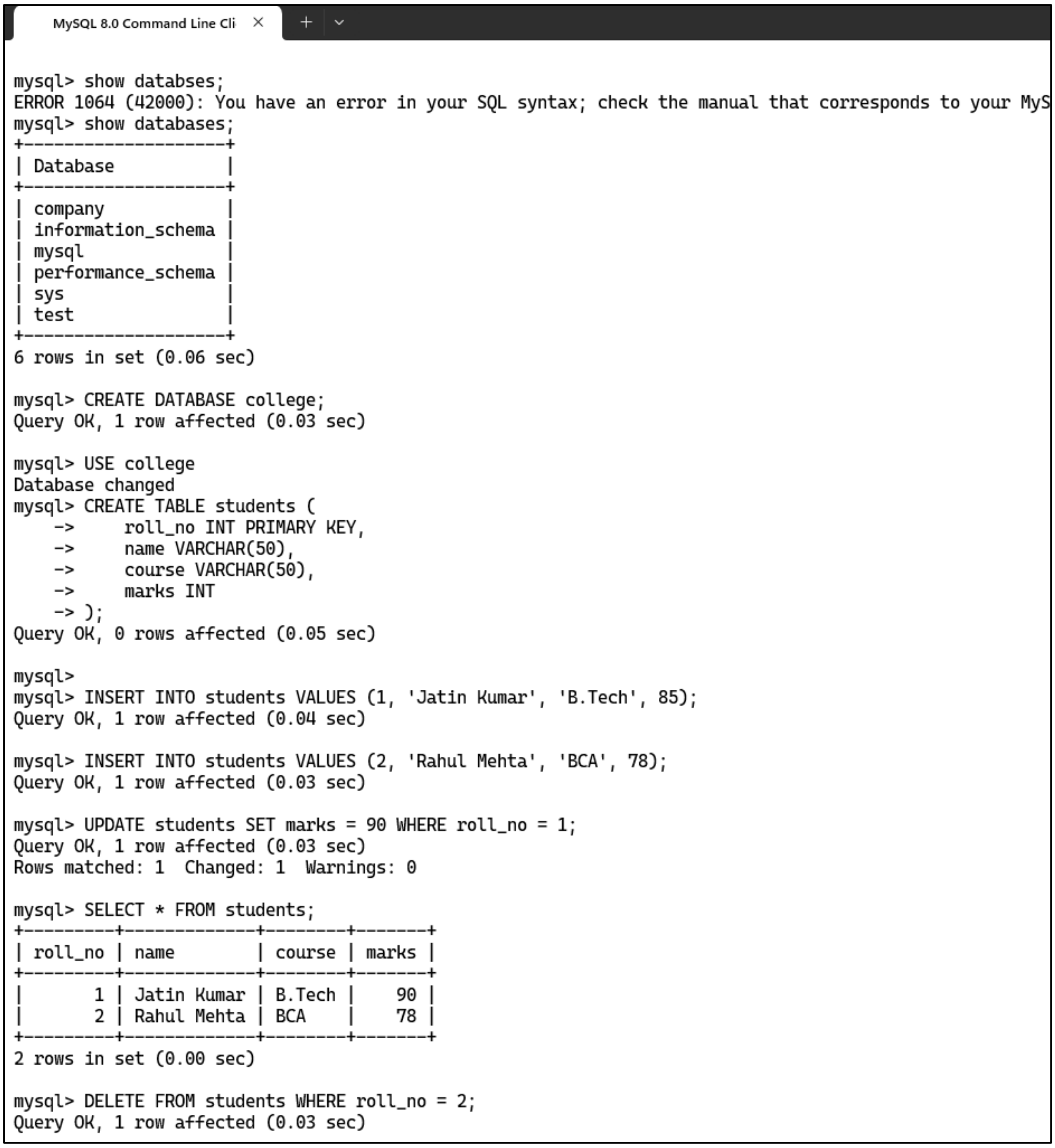
Practical-7

AIM: Create various databases using SQL/MongoDB/or other to show interactivity.

Code:

```
-- Create Database
CREATE DATABASE college;

-- Use the Database
USE college;

-- Create Table
CREATE TABLE students (
   roll_no INT PRIMARY KEY,
   name VARCHAR(50),
   course VARCHAR(50),
   marks INT
);

-- Insert Records
INSERT INTO students VALUES (1, 'Jatin Kumar', 'B.Tech', 85);
INSERT INTO students VALUES (2, 'Rahul Mehta', 'BCA', 78);

-- Update a Record
UPDATE students SET marks = 90 WHERE roll_no = 1;

-- Retrieve Records (interactivity - viewing data)
SELECT * FROM students;

-- Delete a Record
DELETE FROM students WHERE roll_no = 2;
```

Output:



```
mysql> show databses;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MyS
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| company            |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
6 rows in set (0.06 sec)

mysql> CREATE DATABASE college;
Query OK, 1 row affected (0.03 sec)

mysql> USE college
Database changed
mysql> CREATE TABLE students (
    ->     roll_no INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     course VARCHAR(50),
    ->     marks INT
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> INSERT INTO students VALUES (1, 'Jatin Kumar', 'B.Tech', 85);
Query OK, 1 row affected (0.04 sec)

mysql> INSERT INTO students VALUES (2, 'Rahul Mehta', 'BCA', 78);
Query OK, 1 row affected (0.03 sec)

mysql> UPDATE students SET marks = 90 WHERE roll_no = 1;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM students;
+---------+-------------+--------+-------+
| roll_no | name        | course | marks |
+---------+-------------+--------+-------+
|       1 | Jatin Kumar | B.Tech |    90 |
|       2 | Rahul Mehta | BCA    |    78 |
+---------+-------------+--------+-------+
2 rows in set (0.00 sec)

mysql> DELETE FROM students WHERE roll_no = 2;
Query OK, 1 row affected (0.03 sec)
```

Practical-8

AIM: Perform CRUD operations using React JS as frontend technology and Node JS as backend technology.

Code:

Backend(Node JS)

(Server.js)

```
// server/server.js
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import dotenv from "dotenv";
dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

// --- DB ---
await mongoose.connect(process.env.MONGO_URI || "mongodb://127.0.0.1:27017/crud_demo");

// --- Model ---
const taskSchema = new mongoose.Schema({
  title: { type: String, required: true },
  done: { type: Boolean, default: false },
}, { timestamps: true });

const Task = mongoose.model("Task", taskSchema);

// --- Routes (CRUD) ---
// C = Create
app.post("/api/tasks", async (req, res) => {
  try {
    const task = await Task.create({ title: req.body.title, done: !!req.body.done });
    res.status(201).json(task);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

// R = Read (all)
app.get("/api/tasks", async (_req, res) => {
  const tasks = await Task.find().sort({ createdAt: -1 });
  res.json(tasks);
});

// U = Update
app.put("/api/tasks/:id", async (req, res) => {
  try {
    const task = await Task.findByIdAndUpdate(
      req.params.id,
      { title: req.body.title, done: req.body.done },
      { new: true }
    );
    res.json(task);
  } catch (e) { res.status(400).json({ error: e.message }); }
});

// D = Delete
app.delete("/api/tasks/:id", async (req, res) => {
  try {
    await Task.findByIdAndDelete(req.params.id);
    res.json({ ok: true });
  } catch (e) { res.status(400).json({ error: e.message }); }
});

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`API running on http://localhost:${PORT}`));
```

(.env)

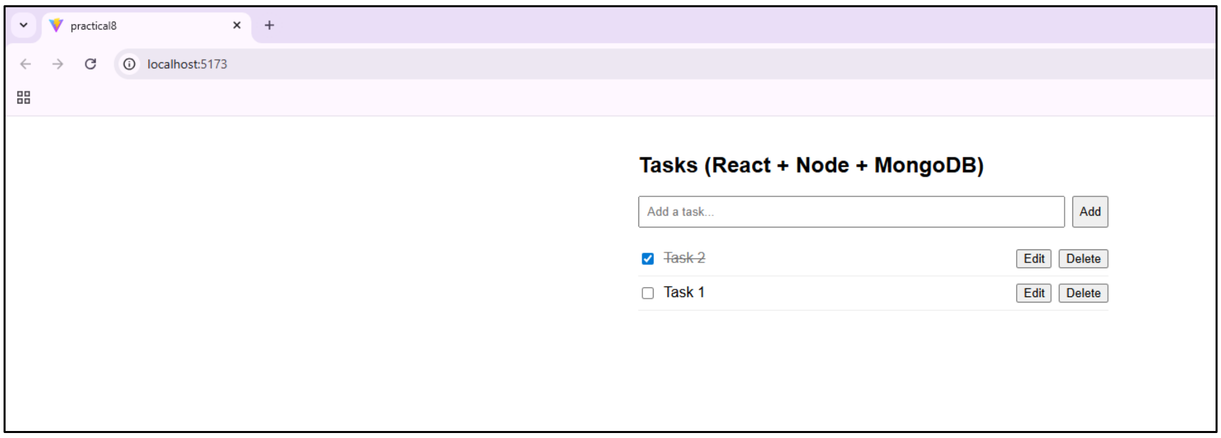
```
MONGO_URI=mongodb://localhost:27017/
```

Frontend (React JS)

(src/App.jsx)

```
import { useEffect, useState } from "react";

const API = "http://localhost:5000/api/tasks";

export default function App() {
  const [tasks, setTasks] = useState([]);
  const [title, setTitle] = useState("");
  const [editing, setEditing] = useState(null);
  const [editTitle, setEditTitle] = useState("");

  async function load() {
    const r = await fetch(API);
    setTasks(await r.json());
  }
  useEffect(() => { load(); }, []);

  async function addTask(e) {
    e.preventDefault();
    if (!title.trim()) return;
    await fetch(API, {
      method: "POST", headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ title })
    });
    setTitle(""); load();
  }

  async function toggleDone(id, done) {
    const t = tasks.find(x => x._id === id);
    await fetch(`${API}/${id}`, {
      method: "PUT", headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ title: t.title, done })
    });
    load();
  }

  async function startEdit(t) {
    setEditing(t._id); setEditTitle(t.title);
  }

  async function saveEdit(id) {
    await fetch(`${API}/${id}`, {
      method: "PUT", headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ title: editTitle, done: tasks.find(x=>x._id===id).done })
    });
    setEditing(null); setEditTitle(""); load();
  }

  async function remove(id) {
    await fetch(`${API}/${id}`, { method: "DELETE" }); load();
  }

  return (
    <div style={{ fontFamily: "Arial", maxWidth: 520, margin: "40px auto" }}>
      <h2>Tasks (React + Node + MongoDB)</h2>

      <form onSubmit={addTask} style={{ display: "flex", gap: 8 }}>
        <input
          placeholder="Add a task..."
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          style={{ flex: 1, padding: 8 }}
        />
        <button type="submit">Add</button>
      </form>

      <ul style={{ listStyle: "none", padding: 0, marginTop: 16 }}>
        {tasks.map(t => (
          <li key={t._id} style={{
            display: "flex", alignItems: "center", gap: 8, padding: "8px 0",
            borderBottom: "1px solid #eee"
          }}>
            <input
              type="checkbox"
              checked={!!t.done}
              onChange={(e) => toggleDone(t._id, e.target.checked)}
              title="Toggle done"
            />
            {editing === t._id ? (
              <>
                <input
                  value={editTitle}
                  onChange={(e) => setEditTitle(e.target.value)}
                  style={{ flex: 1, padding: 6 }}
                />
                <button onClick={() => saveEdit(t._id)}>Save</button>
                <button onClick={() => setEditing(null)}>Cancel</button>
              </>
            ) : (
              <>
                <span style={{
                  flex: 1,
                  textDecoration: t.done ? "line-through" : "none",
                  color: t.done ? "#777" : "#000"
                }}>
                  {t.title}
                </span>
                <button onClick={() => startEdit(t)}>Edit</button>
                <button onClick={() => remove(t._id)}>Delete</button>
              </>
            )}
          </li>
        ))}
      </ul>
    </div>
  );
 }
```

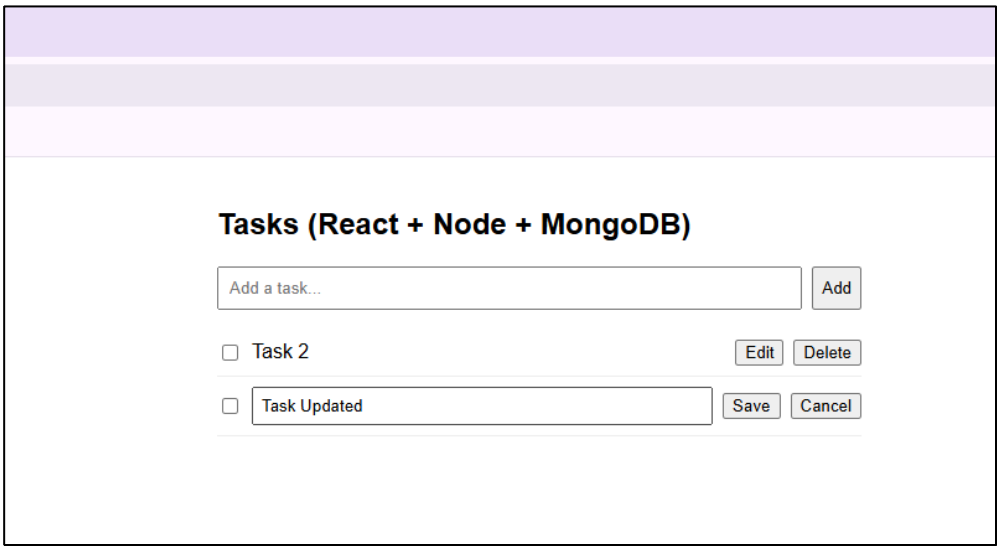
<u>Output:</u>



Tasks (React + Node + MongoDB)

[✓] ~~Task 2~~ Edit Delete

[ ] Task 1 Edit Delete



Tasks (React + Node + MongoDB)

[ ] Task 2 Edit Delete

[ ] Task Updated Save Cancel

Practical-9

AIM: Develop robust back-end systems using Node.js.

Code:

(server.js)

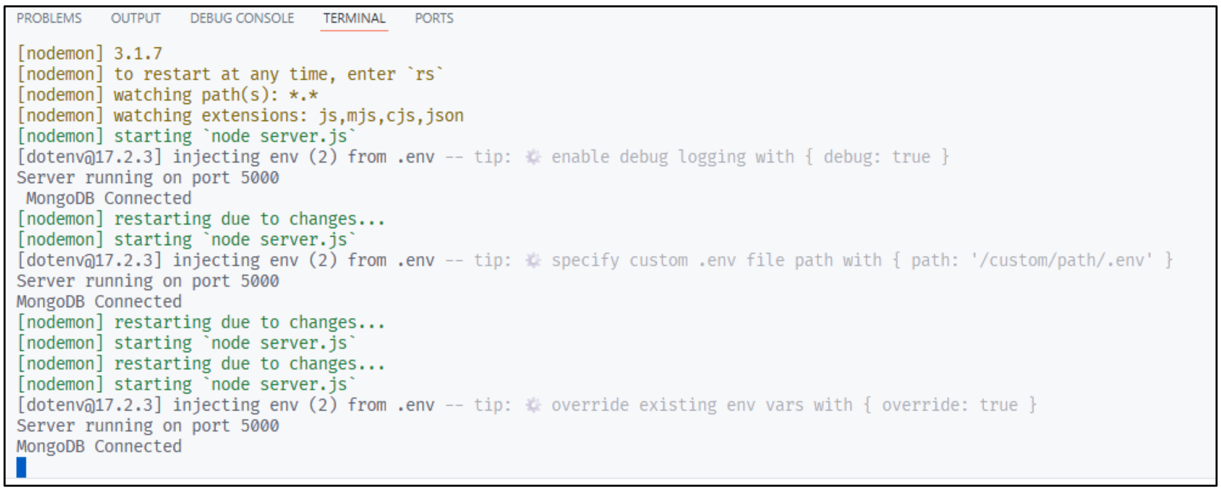
```
// server.js
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import dotenv from "dotenv";
dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

// --- Database Connection ---
const MONGO_URI = process.env.MONGO_URI || "mongodb://127.0.0.1:27017/backend_demo";
mongoose.connect(MONGO_URI)
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error("DB Error:", err));

// --- Schema & Model ---
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: { type: Number, min: 1 },
}, { timestamps: true });

const User = mongoose.model("User", userSchema);

// --- Routes (CRUD) ---

// CREATE
app.post("/api/users", async (req, res) => {
  try {
    const { name, email, age } = req.body;
    if (!name || !email) return res.status(400).json({ message: "Name and email required" });
    const user = await User.create({ name, email, age });
    res.status(201).json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
});

// READ
app.get("/api/users", async (req, res) => {
  const users = await User.find().sort({ createdAt: -1 });
  res.json(users);
});
```

```
// UPDATE
app.put("/api/users/:id", async (req, res) => {
  try {
    const user = await User.findByIdAndUpdate(req.params.id, req.body, { new: true });
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: "Update failed" });
  }
});

// DELETE
app.delete("/api/users/:id", async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.json({ message: "User deleted successfully" });
  } catch (error) {
    res.status(500).json({ message: "Delete failed" });
  }
});

// --- Global Error Handler ---
app.use((err, _req, res, _next) => {
  console.error("Server Error:", err);
  res.status(500).json({ message: "Internal Server Error" });
});

// --- Start Server ---
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

(.env)

```
MONGO_URI=mongodb://127.0.0.1:27017/backend_demo
PORT=5000
```

<u>Output:</u>



```
[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (2) from .env -- tip: ⚙ enable debug logging with { debug: true }
Server running on port 5000
 MongoDB Connected
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (2) from .env -- tip: ⚙ specify custom .env file path with { path: '/custom/path/.env' }
Server running on port 5000
MongoDB Connected
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (2) from .env -- tip: ⚙ override existing env vars with { override: true }
Server running on port 5000
MongoDB Connected
```

Practical-10

AIM: Showing database interactivity using PHP/Python/or any current technology used in web industry.

Code:

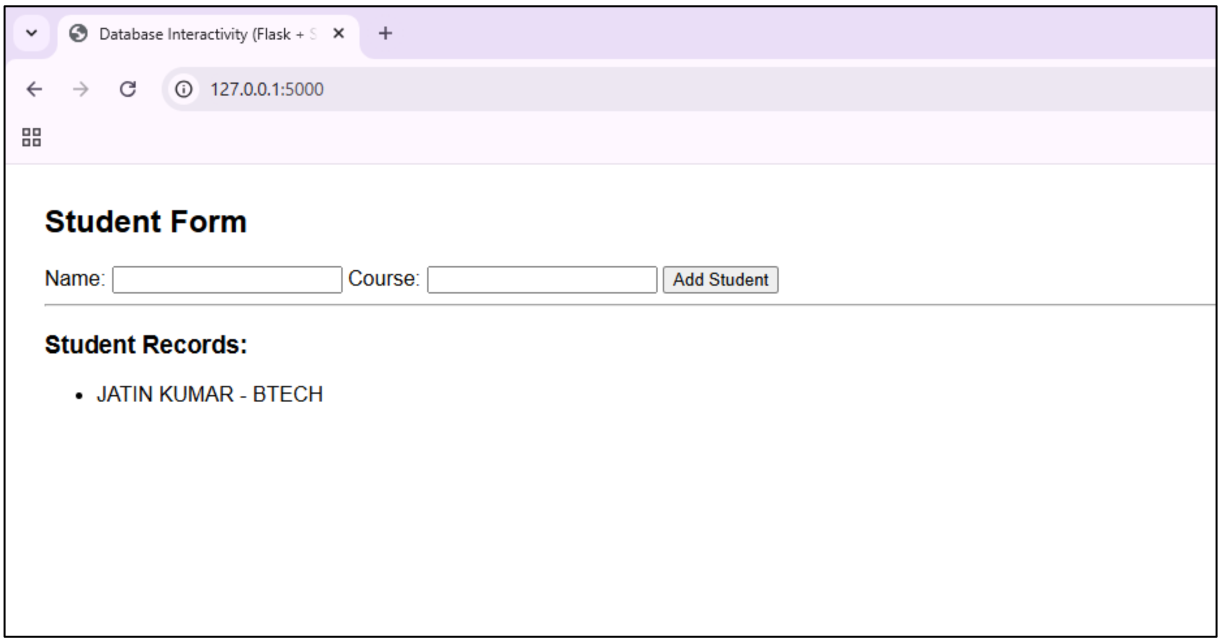
Using Python , Flask , SQLite

(app.py)

```
from flask import Flask, request, render_template_string
import sqlite3

app = Flask(__name__)

HTML = """
<!DOCTYPE html>
<html>
<head><title>Database Interactivity (Flask + SQLite)</title></head>
<body style="font-family:Arial; margin:30px;">
  <h2>Student Form</h2>
  <form method="post">
    Name: <input name="name" required>
    Course: <input name="course" required>
    <input type="submit" value="Add Student">
  </form>
  <hr>
  <h3>Student Records:</h3>
  <ul>
    {% for s in students %}
      <li>{{s[1]}} - {{s[2]}}</li>
    {% endfor %}
  </ul>
</body></html>
"""

def init_db():
    conn = sqlite3.connect('college.db')
    conn.execute("CREATE TABLE IF NOT EXISTS students (id INTEGER PRIMARY KEY, name TEXT, course TEXT)")
    conn.close()

@app.route("/", methods=["GET", "POST"])
def index():
    conn = sqlite3.connect('college.db')
    if request.method == "POST":
        name = request.form["name"]
        course = request.form["course"]
        conn.execute("INSERT INTO students (name, course) VALUES (?, ?)", (name, course))
        conn.commit()
    cur = conn.execute("SELECT * FROM students")
    students = cur.fetchall()
    conn.close()
    return render_template_string(HTML, students=students)
```

```
if __name__ == "__main__":
    init_db()
    app.run(debug=True)
```

Output:



Student Form

Name: Course: Add Student

Student Records:

- JATIN KUMAR - BTECH

Practical-11

AIM: Any current web industry relevant example of database usage and interactivity using any suitable backend technology.

Code:

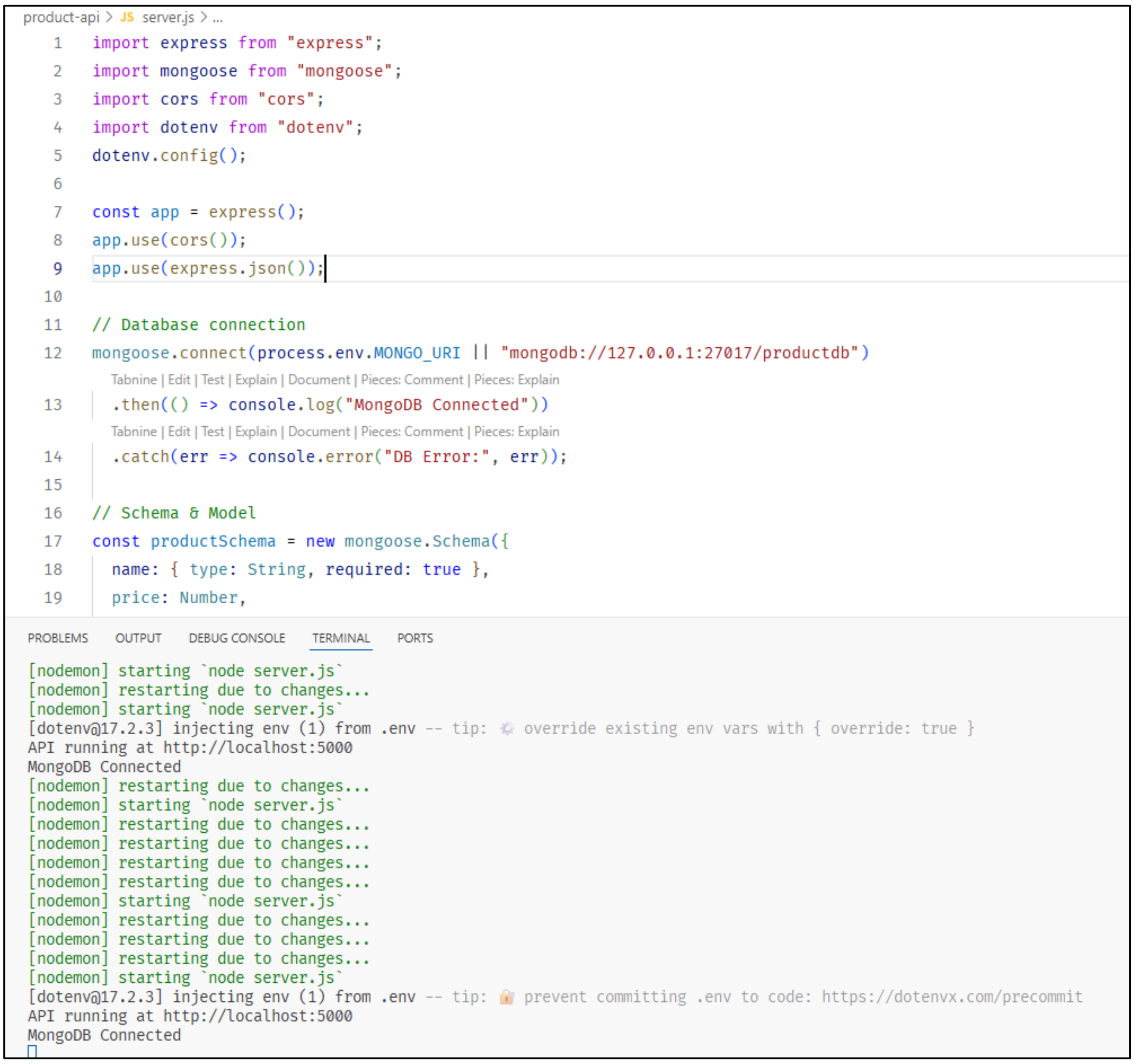
(server.js)

```
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import dotenv from "dotenv";
dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

// Database connection
mongoose.connect(process.env.MONGO_URI || "mongodb://127.0.0.1:27017/productdb")
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error("DB Error:", err));

// Schema & Model
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: Number,
  category: String,
  inStock: { type: Boolean, default: true }
}, { timestamps: true });

const Product = mongoose.model("Product", productSchema);

// Routes
// CREATE
app.post("/api/products", async (req, res) => {
  try {
    const product = await Product.create(req.body);
    res.status(201).json(product);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
});

// READ
app.get("/api/products", async (_req, res) => {
  const products = await Product.find().sort({ createdAt: -1 });
  res.json(products);
});

// UPDATE
app.put("/api/products/:id", async (req, res) => {
  const product = await Product.findByIdAndUpdate(req.params.id, req.body, { new: true });
  res.json(product);
});

// DELETE
app.delete("/api/products/:id", async (req, res) => {
  await Product.findByIdAndDelete(req.params.id);
  res.json({ message: "Product deleted" });
});

// Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`API running at http://localhost:${PORT}`));
```

(.env)

```
MONGO_URI=mongodb://localhost:27017/
```

Output:


product-api > JS server.js > ...
Tabnine | Edit | Test | Explain | Document | Pieces: Comment | Pieces: Explain
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
import express from "express";
import mongoose from "mongoose";
import cors from "cors";
import dotenv from "dotenv";
dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

// Database connection
mongoose.connect(process.env.MONGO_URI || "mongodb://127.0.0.1:27017/productdb")
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.error("DB Error:", err));

// Schema & Model
const productSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: Number,
```

```
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (1) from .env -- tip: ⚙️ override existing env vars with { override: true }
API running at http://localhost:5000
MongoDB Connected
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] restarting due to changes...
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (1) from .env -- tip: 🔐 prevent committing .env to code: https://dotenvx.com/precommit
API running at http://localhost:5000
MongoDB Connected
```