NGI
NAGARJUNA
GROUP OF INSTITUTIONS

NAGARJUNA
COLLEGE OF ENGINEERING & TECHNOLOGY

NAGARJUNA COLLEGE OF ENGINERING AND TECHNOLOGY

(An Autonomous Institute under VTU)

Mudugurki, Venkatgirikote, Devanahalli, Karnataka, Bengaluru-562164.

Department of Information Science and Engineering

Laboratory Manual

VI Semester – 2023

ADVANCED WEB APPLICATION PROGRAMMING LAB

(20ISI63)

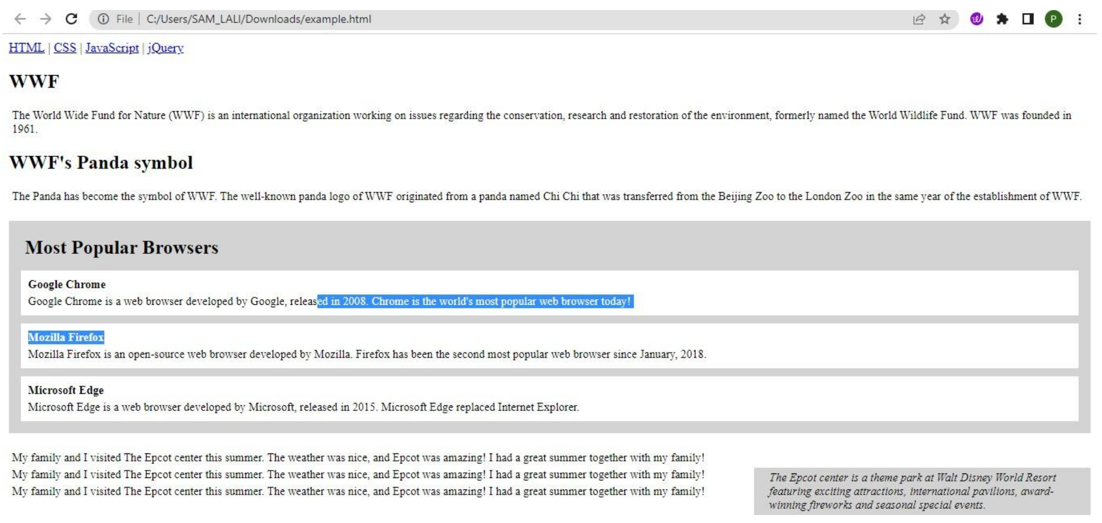
1. Explain the role of the following semantic elements of HTML5 with syntax and script segments:
i <nav> ii <section> iii <aside>

```
<!DOCTYPE html>
<html>
<style>
.all-browsers {
  margin: 0;
  padding: 5px;
  background-color: lightgray;
}

.all-browsers > h1, .browser {
  margin: 10px;
  padding: 5px;
}

.browser {
  background: white;
}

.browser > h2, p {
  margin: 4px;
  font-size: 90%;
}
aside  {
  width: 30%;
  padding-left: 15px;
  margin-left: 15px;
  float: right;
  font-style: italic;
  background-color: lightgray;
}
```

```
</style>

<body>


<nav>

  <a href="/html/">HTML</a> |

  <a href="/css/">CSS</a> |

  <a href="/js/">JavaScript</a> |

  <a href="/jquery/">jQuery</a>

</nav>

<section>

  <h1>WWF</h1>

  <p>The World Wide Fund for Nature (WWF) is an international organization working on issues
regarding the conservation, research and restoration of the environment, formerly named the World
Wildlife Fund. WWF was founded in 1961.</p>

</section>

<section>

  <h1>WWF's Panda symbol</h1>

  <p>The Panda has become the symbol of WWF. The well-known panda logo of WWF originated
from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the
same year of the establishment of WWF.</p>

</section>


<br>

<article class="all-browsers">

  <h1>Most Popular Browsers</h1>

  <article class="browser">

    <h2>Google Chrome</h2>

    <p>Google Chrome is a web browser developed by Google, released in 2008. Chrome is the
world's most popular web browser today!</p>

  </article>

  <article class="browser">

    <h2>Mozilla Firefox</h2>

    <p>Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the
second most popular web browser since January, 2018.</p>

  </article>

  <article class="browser">
```

```
    <h2>Microsoft Edge</h2>

    <p>Microsoft Edge is a web browser developed by Microsoft, released in 2015. Microsoft Edge replaced Internet Explorer.</p>

  </article>

</article>

<br>

<p>My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!</p>

<aside>

<p>The Epcot center is a theme park at Walt Disney World Resort featuring exciting attractions, international pavilions, award-winning fireworks and seasonal special events.</p>

</aside>


<p>My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!</p>


</body>

</html>
```

OUTPUT:



C:/Users/SAM_LALI/Downloads/example.html

HTML | CSS | JavaScript | jQuery

WWF

The World Wide Fund for Nature (WWF) is an international organization working on issues regarding the conservation, research and restoration of the environment, formerly named the World Wildlife Fund. WWF was founded in 1961.

WWF's Panda symbol

The Panda has become the symbol of WWF. The well-known panda logo of WWF originated from a panda named Chi Chi that was transferred from the Beijing Zoo to the London Zoo in the same year of the establishment of WWF.

Most Popular Browsers

Google Chrome
Google Chrome is a web browser developed by Google, released in 2008. Chrome is the world's most popular web browser today!

Mozilla Firefox
Mozilla Firefox is an open-source web browser developed by Mozilla. Firefox has been the second most popular web browser since January, 2018.

Microsoft Edge
Microsoft Edge is a web browser developed by Microsoft, released in 2015. Microsoft Edge replaced Internet Explorer.

My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!
My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!
My family and I visited The Epcot center this summer. The weather was nice, and Epcot was amazing! I had a great summer together with my family!

The Epcot center is a theme park at Walt Disney World Resort featuring exciting attractions, international pavilions, award-winning fireworks and seasonal special events.

2. Build a web server using HTTP Module in Node JS and perform file system modules like
i. Read files
ii. Create files
iii. Update files
iv. Delete files
v. Rename files

demofile1.html

```
<html>
<body>
<h1>My Header</h1>
<p>My paragraph.</p>
</body>
</html>
```

Demo.js

```
var http = require('http');
var fs = require('fs');

fs.appendFile('mynewfile1.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});

fs.open('mynewfile2.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});

fs.appendFile('mynewfile1.txt', ' This is my text.', function (err) {
  if (err) throw err;
  console.log('Updated!');
});

fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {
  if (err) throw err;
  console.log('Replaced!');
});

fs.unlink('mynewfile2.txt', function (err) {
  if (err) throw err;
  console.log('File deleted!');
});

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});

http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(data);
    return res.end();
  });
}).listen(8080);
```

OUTPUT:

output of each operation:

1. `fs.appendFile('mynewfile1.txt', 'Hello content!', ...)`: This appends the text 'Hello content!' to the file 'mynewfile1.txt'. Output: Saved!
2. `fs.open('mynewfile2.txt', 'w', ...)`: This opens the file 'mynewfile2.txt' in write mode ('w'). Output: Saved!
3. `fs.writeFile('mynewfile3.txt', 'Hello content!', ...)`: This writes the text 'Hello content!' to the file 'mynewfile3.txt'. Output: Saved!
4. `fs.appendFile('mynewfile1.txt', ' This is my text.', ...)`: This appends the text ' This is my text.' to the existing content in 'mynewfile1.txt'. Output: Updated!
5. `fs.writeFile('mynewfile3.txt', 'This is my text', ...)`: This replaces the existing content in 'mynewfile3.txt' with 'This is my text'. Output: Replaced!
6. `fs.unlink('mynewfile2.txt', ...)`: This deletes the file 'mynewfile2.txt'. Output: File deleted!
7. `fs.rename('mynewfile1.txt', 'myrenamedfile.txt', ...)`: This renames 'mynewfile1.txt' to 'myrenamedfile.txt'. Output: File Renamed!
8. `http.createServer(...).listen(8080);`: This creates an HTTP server that listens on port 8080 and serves the content of the file 'demofile1.html' when a request is made to the server.

3. Perform CRUD Operation in MongoDB with connection to NodeJS.

To perform CRUD (Create, Read, Update, Delete) operations in MongoDB with a connection to Node.js, you can use the MongoDB Node.js driver.

Install the MongoDB Node.js driver by running the following command in your project directory:

```
npm install mongodb
```

Require the MongoDB driver in your Node.js script:

javascript

```
const MongoClient = require('mongodb').MongoClient;
```

Set up the MongoDB connection URL and specify the database name:

```
const url = 'mongodb://localhost:27017'; // Update with your MongoDB connection URL
const dbName = 'your-database-name'; // Update with your database name
```

Connect to the MongoDB server:

```
MongoClient.connect(url, (err, client) => {
if (err) {
    console.error('Error connecting to MongoDB:', err);
    return;
  }
  console.log('Connected to MongoDB successfully');
  const db = client.db(dbName);
  // Perform CRUD operations here
  client.close();
});
```

Perform CRUD operations within the callback function:

Create operation:

```
const collection = db.collection('your-collection-name');
const document = { name: 'John Doe', age: 30 };
collection.insertOne(document, (err, result) => {
  if (err) {
    console.error('Error creating document:', err);
    return;
  }
  console.log('Document created successfully:', result.insertedId);
});
```

Read operation:

```
const collection = db.collection('your-collection-name');
collection.find({}).toArray((err, documents) => {
  if (err) {
    console.error('Error reading documents:', err);
    return;
  }
  console.log('Documents:', documents);
});
```

Update operation:

```
const collection = db.collection('your-collection-name');
const filter = { name: 'John Doe' };
const update = { $set: { age: 35 } };
collection.updateOne(filter, update, (err, result) => {
if (err) {
    console.error('Error updating document:', err);
    return;
  }

  console.log('Document updated successfully');
});
```

Delete operation:

```
const collection = db.collection('your-collection-name');
const filter = { name: 'John Doe' };
collection.deleteOne(filter, (err, result) => {
  if (err) {
    console.error('Error deleting document:', err);
    return;
  }
  console.log('Document deleted successfully');
});
```

OUTPUT:

1. Connecting to MongoDB:
- If the connection to MongoDB is successful, it will log: "Connected to MongoDB successfully."
- If there is an error connecting to MongoDB, it will log the error message: "Error connecting to MongoDB: [Error Message]".

2. CRUD Operations:
- Create Operation:
  - If the document is successfully created, it will log the inserted document's ObjectId: "Document created successfully: [ObjectId]".
  - If there is an error creating the document, it will log the error message: "Error creating document: [Error Message]".
- Read Operation:
  - If documents are successfully fetched, it will log an array of documents: "Documents: [Array of Documents]".
  - If there is an error reading documents, it will log the error message: "Error reading documents: [Error Message]".
- Update Operation:
  - If the document is successfully updated, it will log: "Document updated successfully".
  - If there is an error updating the document, it will log the error message: "Error updating document: [Error Message]".
- Delete Operation:
  - If the document is successfully deleted, it will log: "Document deleted successfully".
  - If there is an error deleting the document, it will log the error message: "Error deleting document: [Error Message]".

4. Write a Program to handle async wait in Javascript.

A program that demonstrates the usage of async/await in JavaScript to handle asynchronous operations:

```
// Function that returns a promise
function getData() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = 'Async data';
      resolve(data);
    }, 2000);
  });
}
// Async function using the await keyword
async function processData() {
  try {
    console.log('Fetching data...');
    const result = await getData();
    console.log('Data:', result);
    console.log('Data processed successfully!');
  } catch (error) {
    console.error('Error:', error);
  }
}
// Call the async function
processData();
```

OUTPUT:

1. Function `getData()`:

- The `getData()` function returns a Promise that resolves after a delay of 2000 milliseconds (2 seconds) with the data 'Async data'.

2. Async Function `processData()`:

- The `processData()` function is declared as an `async` function, which means it can use the `await` keyword to wait for a Promise to resolve before continuing further.

- Inside `processData()`, the `await` keyword is used before the `getData()` function call. This means the function will wait for the Promise returned by `getData()` to resolve before moving to the next line of code.
- While waiting for the Promise to resolve, the code logs 'Fetching data...' to the console.
- When the Promise is resolved, the result is stored in the `result` variable, and the code logs 'Data: Async data' to the console.
- Finally, the code logs 'Data processed successfully!' to the console.
- If any error occurs during the Promise execution, the `catch` block will catch the error and log 'Error: [Error Message]' to the console.

3. Calling the `processData()` function:

- The `processData()` function is called, initiating the asynchronous operation.

```
Fetching data...
Data: Async data
Data processed successfully!
```

5. Design a page by creating Class and Functional based Components in React JS.

Step 1: Create a new React app

```
npx create-react-app todo-list-app
cd todo-list-app
```

Step 2: Install additional dependencies (optional)

```
npm install bootstrap
```

Class-based Component - TodoListClass.js

```
import React, { Component } from 'react';

class TodoListClass extends Component {
  constructor(props) {
    super(props);
    this.state = {
    todos: [],
      newTodo: '',
    };
  }

  handleInputChange = (event) => {
    this.setState({ newTodo: event.target.value });
  };

  handleAddTodo = () => {
    if (this.state.newTodo.trim() !== '') {
      this.setState((prevState) => ({
        todos: [...prevState.todos, this.state.newTodo],
        newTodo: '',
      }));
    }
  };

  handleDeleteTodo = (index) => {
    this.setState((prevState) => ({
      todos: prevState.todos.filter((_, i) => i !== index),
    }));
  };

  render() {
    const { todos, newTodo } = this.state;

    return (
      <div>
        <h1>Todo List (Class-based)</h1>
        <div className="input-group mb-3">
          <input
            type="text"
            className="form-control"
            placeholder="Add new todo"
            value={newTodo}
            onChange={this.handleInputChange}
          />
          <div className="input-group-append">
            <button className="btn btn-primary" onClick={this.handleAddTodo}>
              Add
            </button>
          </div>
        </div>
        <ul className="list-group">
          {todos.map((todo, index) => (
            <li className="list-group-item" key={index}>
              {todo}
              <button
                className="btn btn-sm btn-danger float-right"
                onClick={() => this.handleDeleteTodo(index)}
              >
                Delete
              </button>
            </li>
          ))}
        </ul>
      </div>
    );
  }
}

export default TodoListClass;
```

Functional Component - TodoListFunction.js

```
import React, { useState } from 'react';

const TodoListFunction = () => {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState('');

  const handleInputChange = (event) => {
    setNewTodo(event.target.value);
  };

  const handleAddTodo = () => {
    if (newTodo.trim() !== '') {
      setTodos([...todos, newTodo]);
      setNewTodo('');
    }
  };

  const handleDeleteTodo = (index) => {
    setTodos(todos.filter((_, i) => i !== index));
  };
  return (
    <div>
      <h1>Todo List (Functional)</h1>
      <div className="input-group mb-3">
        <input
          type="text"
          className="form-control"
          placeholder="Add new todo"
          value={newTodo}
          onChange={handleInputChange}
        />
        <div className="input-group-append">
          <button className="btn btn-primary" onClick={handleAddTodo}>
            Add
          </button>
        </div>
      </div>
      <ul className="list-group">
        {todos.map((todo, index) => (
          <li className="list-group-item" key={index}>
            {todo}
            <button
              className="btn btn-sm btn-danger float-right"
              onClick={() => handleDeleteTodo(index)}
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoListFunction;
```

App.js

```
import React from 'react';
import './App.css';
import TodoListClass from './components/TodoListClass';
import TodoListFunction from './components/TodoListFunction';

function App() {
  return (
    <div className="container">
      <div className="row">
        <div className="col">
          <TodoListClass />
        </div>
        <div className="col">
          <TodoListFunction />
        </div>
      </div>
    </div>
  );
}

export default App;
```

OUTPUT:

output of this code, follow these steps:

1. Create a new React project or use an existing one.
2. Create two files, `TodoListClass.js` and `TodoListFunction.js`, and copy the respective component codes into each file.
3. Create an `App.js` file and copy the provided `App` component code into it.
4. Replace the contents of the `App.css` file with your preferred styling, or you can leave it empty for simplicity.
5. Import and use the components in the `App.js` file as shown in the provided code.
6. Ensure all the dependencies are installed by running `npm install` in your project directory.
7. Start the development server by running `npm start`.

Open the web browser and navigate to `http://localhost:3000`.

Display of two Todo lists side by side: one rendered by the TodoListClass component (labeled as "Todo List (Class-based)") and the other by the TodoListFunction component (labeled as "Todo List (Functional)").
The output should resemble two Todo lists on the web page, and you can interact with them by adding and deleting Todo items. The class-based and functional components have the same functionality, but they are implemented differently using different React patterns.

6. Create a basic app with Spring Boot and React to handle RESTful APIs for performing CRUD operations

Step 1: Set up the Spring Boot Backend

1. Create a new Spring Boot project using Spring Initializr (https://start.spring.io/) with the following dependencies:

Web
JPA
H2 Database (or any other database of your choice)

2. Build and run the Spring Boot project.

Step 2: Set up the React Frontend

1. Create a new folder for your React app and navigate to it in the terminal.
2. Initialize a new React app using Create React App:

```
npx create-react-app frontend
cd frontend
```

3. Install axios, a library for making HTTP requests, to communicate with the backend:

```
npm install axios
```

4. Replace the content of **src/App.js** with the following code:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function App() {
  const [todos, setTodos] = useState([]);
  const [newTodo, setNewTodo] = useState('');

  useEffect(() => {
    fetchTodos();
  }, []);

  const fetchTodos = async () => {
    try {
      const response = await axios.get('/api/todos');
      setTodos(response.data);
    } catch (error) {
      console.error('Error fetching todos:', error);
    }
  };

  const handleInputChange = (event) => {
    setNewTodo(event.target.value);
  };

  const handleAddTodo = async () => {
    if (newTodo.trim() !== '') {
      try {
        await axios.post('/api/todos', { text: newTodo });
        fetchTodos();
        setNewTodo('');
      } catch (error) {
        console.error('Error adding todo:', error);
      }
    }
  };

  const handleDeleteTodo = async (id) => {
    try {
      await axios.delete(`/api/todos/${id}`);
      fetchTodos();
    } catch (error) {
      console.error('Error deleting todo:', error);
    }
  };

  return (
    <div className="container">
      <h1>Todo List</h1>
      <div className="input-group mb-3">
        <input
          type="text"
          className="form-control"
          placeholder="Add new todo"
          value={newTodo}
          onChange={handleInputChange}
        />
        <div className="input-group-append">
          <button className="btn btn-primary" onClick={handleAddTodo}>
            Add
          </button>
        </div>
      </div>
      <ul className="list-group">
        {todos.map((todo) => (
          <li className="list-group-item" key={todo.id}>
            {todo.text}
            <button
              className="btn btn-sm btn-danger float-right"
              onClick={() => handleDeleteTodo(todo.id)}
            >
              Delete
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Step 3: Implement the Spring Boot Backend

1. Create a new package com.example.todo under the src/main/java directory.
2. Create a new entity class Todo.java:

```
package com.example.todo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Todo {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String text;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

Create a new repository interface **TodoRepository.java**:

```
package com.example.todo;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface TodoRepository extends JpaRepository<Todo, Long> {
}
```

Create a new controller class **TodoController.java**:

```
      package com.example.todo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/api/todos")
public class TodoController {
    private final TodoRepository todoRepository;

    @Autowired
    public TodoController(TodoRepository todoRepository) {
        this.todoRepository = todoRepository;
    }

    @GetMapping
    public List<Todo> getAllTodos() {
        return todoRepository.findAll();
    }

    @PostMapping
    public Todo createTodo(@RequestBody Todo todo) {
        return todoRepository.save(todo);
    }

    @DeleteMapping("/{id}")
    public void deleteTodo(@PathVariable Long id) {
        todoRepository.deleteById(id);
    }
}
```

OUTPUT:

Run the Application

1. Start the Spring Boot backend by running the main class (e.g., **TodoApplication.java**).
2. In a separate terminal, navigate to the **frontend** directory and start the React frontend:

```
npm start
```

Spring Boot backend and React frontend should be running, run the app at **http://localhost:3000**.

CO-PO & PSO Mapping:

POS / COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	1	2		2								3		1
CO2	1	1	3		3		1						1	3	
CO3	3	2	2		3								2	2	
CO4	2	1	3	1	3			2	2	1	2		2	2	2
CO5	2	3	3	2	3	1	1	2	1	2	2	2	2	2	2
	2.2	**1.4**	**2.6**	**1.5**	**2.8**	**1**	**1**	**2**	**1.5**	**1.5**	**2**	**2**	**2**	**1.8**	**1.7**