Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2025-26

Class:	T.E	**Semester:**	V
Course Code:	CSC502	**Course Name:**	WC

Name of Student:	SHRUTI GAUCHANDRA
Roll No. :	18
Assignment No.:	05
Title of Assignment:	EXPRESS
Date of Submission:	30/09/25
Date of Correction:	06/10/25

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Completeness	5	5
Demonstrated Knowledge	3	3
Legibility	2	2
Total	10	10

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Completeness	5	3-4	1-2
Demonstrated Knowledge Legibility	3	2	1
Legibility	2	1	0

Checked by

Name of Faculty : Ms. KSHITIJA GHARAT

Signature : Gharat

Date : 6/10/25

CSC502.4 Construct web based Node.js applications using Express.

Q1. Create a simple Express application that integrates with React to display a list of items fetched from a server. Describe the steps involved and provide code snippets.

→ A full stack web application can be created using express.js as the backend and React as the frontend.

① Setting up the Express Backend.

First, initialize a Node.js project and install necessary packages.

```
npm init -y
npm install express cors
```

Create a file named server.js with the following code:

code:

```
const express = require('express');
const cors = require('cors');
const app = express();

app.use(cors()); // Enable CORS

// Example list of items
const items = ['Apple', 'Banana', 'Orange'];

// Define API route to fetch items
app.get('/api/items', (req, res) => {
```

```
  res.json(items);
});

//Start the server
  app.listen(5000, () => {
    console.log('server listening on port 5000');
});
```

② Building the React Frontend:

```
npx create-react-app client
cd client
```

In App.js file, write the following code:

Code:

```
import React, {useEffect, useState} from 'react';
function App {
  const [items, setItems] = useState([]);

  useEffect(() => {
    fetch('http://localhost:5000/api/items')
      .then(res => res.json())
      .then(data => setItems(data));
  }, []);

return(
  <div>
    <h1>Item List </h1>
    <ul>
      {items.map((item, idx) => (
```

```
        <li key={idx}>{item}</li>
      ))}
    </ul>
  </div>
  );
}
export default App;
```

③ Running both servers:
node server.js
npm start.

④ By these steps, a simple full-stack application is achieved, where data is provided by Express and displayed by React. The key is to ensure the frontend fetches data from correct backend API, and both servers running concurrently.

Q2. Apply Express Router to organize route handling in a Node.js application by creating a modular structure. Implement a simple example to demonstrate how Express Router improves code organization.

→ Express Router allows breaking large route files into smaller, manageable modules.

① Create the Project structure:

```
project/
  ├── app.js
  └── routes/
      └── users.js
```

② Implement Modular Route File.

Code:

```
//routes/users.js
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('User List');
});

router.get('/profile', (req, res) => {
  res.send('User Profile');
});
```

③ Import and Mount Routes in Main App.

Code:

```
// app.js
const express = require('express');
const app = express();

const usersRouter = require('./routes/users');
app.use('/users', usersRouter);

app.listen(3000, () => {
```

```
    console.log('server running on port 3000');
});
```

The modularity is that, each group of related routes is in its own file.

Q3. You are developing a login system in Express application. After a user successfully logs in, you need to store their username in a cookie that should only be accessible via HTTP and should expire after 1 minute. Write the Express route that sets this cookie and another route that reads and displays the username from the cookie.

→ Code:

```
const express = require('express');
const cookieParser =
    require('cookie-parser');
const app = express();

app.use(express.json());
app.use(cookieParser());

//Route to set the HTTP
app.post('/login',(req, res) => {
    const {username} = req.body;
    res.cookie('username', username, {
    httpOnly: true,
```

```
    maxAge : 60000
});
res.send('Login successful, cookie.set!');
});

//Route to read and display the username
 from cookie.
app.get('/profile', (req, res) => {
  const username = req.cookies.username;
  if (username) {
    res.send(`Hello, ${username}!`);
  }
  else {
    res.send('No username cookie found.');
});

app.listen(3000, () => {
  console.log('server running on port 3000');
});
```

Q4. You are building a web application where users can view a list of products. Apply your understanding of REST APIs by creating an Express route that handles an HTTP GET request at /products and returns a sample JSON array of products. Show the code snippet for the route.

→ Code:

```
const express = require('express');
const app = express();

const products = [
  {
    id: 1
    name: 'Wireless Headphones',
    category: 'Electronics',
    price: 79.99
    inStock: true
  },
  {
    id: 2
    name: 'Smart Watch',
    category: 'Wearables',
    price: 199.99
    inStock: true
  },
  {
    id: 3
    name: 'Bluetooth Speaker',
    category: 'Electronics',
    price: 49.99,
    inStock: false
  },
  {
    id: 4
    name: 'USB-C Cable',
    category: 'Accessories',
    price: 12.99,
```

```
    inStock : true
  }
];

app.get('/products',(req, res) => {
  res.status(200).json({
    success: true,
    count : products.length,
    data : products
  });
});

app.get('/products/:id',(req,res) => {
  const product =products.find(p => p.id ===
      parseInt(req.params.id));

  if(product) {
    res.status(200).json({
      success: true
      data : product
    });
  } else{
    res.status(404).json({
      success : false
      message: 'Product not found'
  });
  }
});
```

```
app.listen(3000, () => {
    console.log('server running on port 3000');

});
```