**P.V.P Siddhartha Institute of Technology**

**Department of Computer Science and Engineering**

**Subject Code: 20CS3603, Subject Name: MERN Stack Development:PVP20**

**Descriptive Examination -2**

1 A) Demonstrate how to create an endpoints or APIs using express for the following requirements

1. For reading student list use the endpoint <http://localhost:4000/studentList>
2. For reading student based on the id <http://localhost:4000/student/id>
3. For creating a new student <http://localhost:4000/createStudent>
4. For deleting an existing student based on id <http://localhost:4000/id>
5. Assume a local variable student contains all the students’ records. CO2, L3, 5M

From the given data we can create an Express.js server with endpoints (APIs) for handling student data as per your requirements. We'll assume the student records are stored in a local variable (i.e., in-memory array) and implement CRUD operations using different HTTP methods.

Install Express (if not already):

npm init -y

npm install express

File: server.js

const express = require('express');

const app = express();

const PORT = 4000;

// Middleware to parse JSON data

app.use(express.json());

// Sample in-memory student data

let students = [

{ id: 1, name: 'Alice', age: 20 },

{ id: 2, name: 'Bob', age: 21 },

{ id: 3, name: 'Charlie', age: 22 }

];

// 1️⃣ GET all students - http://localhost:4000/studentList

app.get('/studentList', (req, res) => {

res.status(200).json(students);

});

// 2️⃣ GET a student by ID - http://localhost:4000/student/:id

app.get('/student/:id', (req, res) => {

const id = parseInt(req.params.id);

const student = students.find(stu => stu.id === id);

if (student) {

res.status(200).json(student);

} else {

res.status(404).json({ message: "Student not found" });

}

});

// 3️⃣ POST - Create new student - http://localhost:4000/createStudent

app.post('/createStudent', (req, res) => {

const newStudent = req.body;

if (!newStudent.id || !newStudent.name || !newStudent.age) {

return res.status(400).json({ message: "Missing student data" });

}

const exists = students.some(stu => stu.id === newStudent.id);

if (exists) {

return res.status(409).json({ message: "Student with this ID already exists" });

}

students.push(newStudent);

res.status(201).json({ message: "Student created", student: newStudent });

});

// 4️⃣ DELETE - Delete student by ID - http://localhost:4000/:id

app.delete('/:id', (req, res) => {

const id = parseInt(req.params.id);

const index = students.findIndex(stu => stu.id === id);

if (index !== -1) {

const removed = students.splice(index, 1);

res.status(200).json({ message: "Student deleted", student: removed[0] });

} else {

res.status(404).json({ message: "Student not found" });

}

});

// Start the server

app.listen(PORT, () => {

console.log(`Server is running at http://localhost:${PORT}`);

});

Using **Postman**, **cURL**, or a browser (for GET requests):

| **Operation** | **Method** | **Endpoint** | **Payload (for POST)** |
| --- | --- | --- | --- |
| Get all students | GET | http://localhost:4000/studentList | — |
| Get student by ID | GET | http://localhost:4000/student/2 | — |
| Create new student | POST | http://localhost:4000/createStudent | { "id": 4, "name": "David", "age": 19 } |
| Delete student by ID | DELETE | http://localhost:4000/2 | — |

2 A) What is React Component? What are the different ways of creating a component in react explain with a suitable example CO3, L2, 2M

A **React component** is a **reusable building block** in a React application that defines how a part of the user interface (UI) should appear and behave. Components let you split the UI into **independent, self-contained pieces**, and each component manages its own state and rendering logic. 0.5M

React components can be:

* Reused across different parts of the app
* Nested inside other components
* Stateful or stateless

**Types of Components in React 1.5M**

There are **two main ways** to create a component in React:

**Function Component (Modern Approach)**

Function components are **JavaScript functions** that return JSX. They are simpler and commonly used with React Hooks (useState, useEffect, etc.).

import React from 'react';

function Welcome(props) {

return <h2>Hello, {props.name}!</h2>;

} export default Welcome;

Usage:

<Welcome name="Alice" />

**Class Component (Traditional/Legacy)**

Class components are created using ES6 classes and can maintain their own **state** and use **lifecycle methods** like componentDidMount().

Example:

import React, { Component } from 'react';

class Welcome extends Component {

render() {

return <h2>Hello, {this.props.name}!</h2>;

}

}export default Welcome;

Usage:

<Welcome name="Bob" />

| **Feature** | **Functional Component** | **Class Component** |
| --- | --- | --- |
| **Syntax** | **Function** | **Class** |
| **State Management** | **Hooks (useState)** | **this.state** |
| **Lifecycle Methods** | **Hooks (useEffect)** | **Yes (componentDidMount, etc.)** |
| **Code Simplicity** | **Simpler** | **More complex** |
| **Performance** | **Slightly better** | **Slightly heavier** |

B) An user needs to display the content in the browser as follows

**Employee Details**

|  |  |  |  |
| --- | --- | --- | --- |
| empid | Name | Department | Salary |
| 101 | Alex | Accounts | 25000.50 |
| 102 | William | Marketing | 21500.25 |
| 103 | Jhon | Production | 22500.25 |
| 104 | Mike | Accounts | 28500.50 |
| 105 | Watson | Production | 16850.25 |

App.js

export default App(){

return (

// write the JSX code to display the above UI

);

}

Assume index.js

const root =ReactDOM.createRoot(document.getElementById('root'));

root.render(

  <React.StrictMode>

    <App />

  </React.StrictMode>

); CO3, L3, 3M

Implementing the given output in **React JSX** using a clean and semantic HTML table inside the App.js file

import React from 'react';

export default function App() {

return (

<div style={{ padding: '20px', fontFamily: 'Arial' }}>

<h2>Employee Details</h2>

<table border="1" cellPadding="10" cellSpacing="0">

<thead>

<tr>

<th>empid</th>

<th>Name</th>

<th>Department</th>

<th>Salary</th>

</tr>

</thead>

<tbody>

<tr>

<td>101</td>

<td>Alex</td>

<td>Accounts</td>

<td>25000.50</td>

</tr>

<tr>

<td>102</td>

<td>William</td>

<td>Marketing</td>

<td>21500.25</td>

</tr>

<tr>

<td>103</td>

<td>Jhon</td>

<td>Production</td>

<td>22500.25</td>

</tr>

<tr>

<td>104</td>

<td>Mike</td>

<td>Accounts</td>

<td>28500.50</td>

</tr>

<tr>

<td>105</td>

<td>Watson</td>

<td>Production</td>

<td>16850.25</td>

</tr>

</tbody>

</table>

</div>

);

}

3 . Demonstrate with a suitable example how CRUD operations are performed from server and choose appropriate http method and database(MongoDB) method for the following operations.

1. Create a new book
2. Read a book based on bookId
3. Read all books
4. Update the book price

Delete the book based on bookId CO4, L3 5M

**Initialize Project and Install Dependencies:**

npm init -y

npm install express mongoose body-parser

**book.model.js – Define Mongoose Schema**

const mongoose = require('mongoose');

const bookSchema = new mongoose.Schema({

title: String,

author: String,

price: Number,

publishedYear: Number

});

module.exports = mongoose.model('Book', bookSchema);

**server.js – Express Server with CRUD APIs**

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const Book = require('./book.model');

const app = express();

app.use(bodyParser.json());

// Connect to MongoDB

mongoose.connect('mongodb://localhost:27017/libraryDB', {

useNewUrlParser: true,

useUnifiedTopology: true

}).then(() => console.log('MongoDB Connected'))

.catch(err => console.log(err));

// 1️⃣ Create a new book (HTTP POST)

app.post('/books', async (req, res) => {

try {

const newBook = new Book(req.body);

await newBook.save();

res.status(201).json({ message: 'Book created', book: newBook });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// 2️⃣ Read a book based on bookId (HTTP GET)

app.get('/books/:id', async (req, res) => {

try {

const book = await Book.findById(req.params.id);

if (book)

res.status(200).json(book);

else

res.status(404).json({ message: 'Book not found' });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// 3️⃣ Read all books (HTTP GET)

app.get('/books', async (req, res) => {

try {

const books = await Book.find();

res.status(200).json(books);

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// 4️⃣ Update book price (HTTP PUT or PATCH)

app.put('/books/:id/price', async (req, res) => {

try {

const updatedBook = await Book.findByIdAndUpdate(

req.params.id,

{ price: req.body.price },

{ new: true }

);

if (updatedBook)

res.status(200).json({ message: 'Price updated', book: updatedBook });

else

res.status(404).json({ message: 'Book not found' });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// 5️⃣ Delete book by bookId (HTTP DELETE)

app.delete('/books/:id', async (req, res) => {

try {

const deletedBook = await Book.findByIdAndDelete(req.params.id);

if (deletedBook)

res.status(200).json({ message: 'Book deleted', book: deletedBook });

else

res.status(404).json({ message: 'Book not found' });

} catch (err) {

res.status(500).json({ error: err.message });

}

});

// Start server

app.listen(4000, () => {

console.log('Server running on http://localhost:4000');});