Let's address each question from the exam paper with detailed answers and code examples using React.js.

### Q1. Fetch data from an API, display `totalResults`, and display all articles

1. \*\*Fetch data in your browser console window:\*\*

```javascript

fetch('https://newsapi.org/v2/everything?q=apple&from=2023-06-06&to=2023-06-06&sortBy=popularity&apiKey=YOUR\_API\_KEY')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Error fetching data:', error));

```

2. \*\*Display value of `totalResults` in a web application:\*\*

```jsx

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

const NewsApp = () => {

const [totalResults, setTotalResults] = useState(0);

useEffect(() => {

fetch('https://newsapi.org/v2/everything?q=apple&from=2023-06-06&to=2023-06-06&sortBy=popularity&apiKey=YOUR\_API\_KEY')

.then(response => response.json())

.then(data => setTotalResults(data.totalResults))

.catch(error => console.error('Error fetching data:', error));

}, []);

return (

<div>

<h1>Total Results: {totalResults}</h1>

</div>

);

};

export default NewsApp;

```

3. \*\*Display all articles in a web application:\*\*

```jsx

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

const NewsApp = () => {

const [totalResults, setTotalResults] = useState(0);

const [articles, setArticles] = useState([]);

useEffect(() => {

fetch('https://newsapi.org/v2/everything?q=apple&from=2023-06-06&to=2023-06-06&sortBy=popularity&apiKey=YOUR\_API\_KEY')

.then(response => response.json())

.then(data => {

setTotalResults(data.totalResults);

setArticles(data.articles);

})

.catch(error => console.error('Error fetching data:', error));

}, []);

return (

<div>

<h1>Total Results: {totalResults}</h1>

<div>

{articles.map((article, index) => (

<div key={index}>

<h2>{article.title}</h2>

<p>{article.description}</p>

<img src={article.urlToImage} alt={article.title} style={{ width: '100px', height: '100px' }} />

</div>

))}

</div>

</div>

);

};

export default NewsApp;

```

### Q2. Single Page Web Application (SPA)

\*\*What is meant by single page web-application?\*\*

- An SPA is a web application that loads a single HTML page and dynamically updates the page as the user interacts with the app.

\*\*How is it constructed?\*\*

- Constructed using client-side JavaScript frameworks like React, Angular, or Vue.js.

- Utilizes routing libraries to manage navigation within the application without reloading the entire page.

\*\*Importance:\*\*

- Provides a seamless user experience with fast and dynamic content updates.

- Reduces server load and enhances performance by fetching only the necessary data.

\*\*Example using React:\*\*

```jsx

import React from 'react';

import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';

const Home = () => <div>Home Page</div>;

const About = () => <div>About Page</div>;

const App = () => {

return (

<Router>

<nav>

<Link to="/">Home</Link>

<Link to="/about">About</Link>

</nav>

<Switch>

<Route exact path="/" component={Home} />

<Route path="/about" component={About} />

</Switch>

</Router>

);

};

export default App;

```

### Q3. CRUD Operations in a Web Application

\*\*What are CRUD Operations?\*\*

- Create: Add new data.

- Read: Retrieve data.

- Update: Modify existing data.

- Delete: Remove data.

\*\*Importance:\*\*

- Fundamental for managing data in web applications.

- Enables full control over data lifecycle.

\*\*Example:\*\*

Backend (Node.js + Express.js + MongoDB):

```javascript

// models/Student.js

const mongoose = require('mongoose');

const StudentSchema = new mongoose.Schema({

name: String,

age: Number,

grade: String,

});

module.exports = mongoose.model('Student', StudentSchema);

// routes/students.js

const express = require('express');

const router = express.Router();

const Student = require('../models/Student');

// Create

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

const newStudent = new Student(req.body);

try {

const student = await newStudent.save();

res.status(201).json(student);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Read

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

try {

const students = await Student.find();

res.json(students);

} catch (err) {

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

}

});

// Update

router.put('/:id', async (req, res) => {

try {

const student = await Student.findByIdAndUpdate(req.params.id, req.body, { new: true });

res.json(student);

} catch (err) {

res.status(400).json({ message: err.message });

}

});

// Delete

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

try {

await Student.findByIdAndDelete(req.params.id);

res.json({ message: 'Student deleted' });

} catch (err) {

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

}

});

module.exports = router;

// server.js

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

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

require('dotenv').config();

const app = express();

app.use(cors());

app.use(bodyParser.json());

mongoose.connect(process.env.MONGO\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true,

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

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

const studentsRouter = require('./routes/students');

app.use('/api/students', studentsRouter);

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

```

Frontend (React.js):

```jsx

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

import axios from 'axios';

const App = () => {

const [students, setStudents] = useState([]);

const [name, setName] = useState('');

const [age, setAge] = useState('');

const [grade, setGrade] = useState('');

const [editing, setEditing] = useState(false);

const [currentStudent, setCurrentStudent] = useState({});

useEffect(() => {

fetchStudents();

}, []);

const fetchStudents = async () => {

const response = await axios.get('/api/students');

setStudents(response.data);

};

const handleSubmit = async (e) => {

e.preventDefault();

const studentData = { name, age, grade };

if (editing) {

await axios.put(`/api/students/${currentStudent.\_id}`, studentData);

setEditing(false);

} else {

await axios.post('/api/students', studentData);

}

setName('');

setAge('');

setGrade('');

fetchStudents();

};

const handleEdit = (student) => {

setEditing(true);

setCurrentStudent(student);

setName(student.name);

setAge(student.age);

setGrade(student.grade);

};

const handleDelete = async (id) => {

await axios.delete(`/api/students/${id}`);

fetchStudents();

};

return (

<div>

<h1>Student Management</h1>

<form onSubmit={handleSubmit}>

<input

type="text"

value={name}

onChange={(e) => setName(e.target.value)}

placeholder="Student Name"

required

/>

<input

type="number"

value={age}

onChange={(e) => setAge(e.target.value)}

placeholder="Age"

required

/>

<input

type="text"

value={grade}

onChange={(e) => setGrade(e.target.value)}

placeholder="Grade"

required

/>

<button type="submit">{editing ? 'Update' : 'Add'}</button>

</form>

<ul>

{students.map((student) => (

<li key={student.\_id}>

{student.name} ({student.age}) - {student.grade}

<button onClick={() => handleEdit(student)}>Edit</button>

<button onClick={() => handleDelete(student.\_id)}>Delete</button>

</li>

))}

</ul>

</div>

);

};

export default App;

```

### Q4. Routing in Web Application

\*\*What is routing in a web application?\*\*

- Routing refers to the mechanism of navigating between different views or pages in a

web application.

\*\*Importance:\*\*

- Allows users to navigate through the application.

- Enhances user experience by providing a seamless transition between different parts of the application.

\*\*Example using React Router:\*\*

```jsx

import React from 'react';

import { BrowserRouter as Router, Route, Switch, Link } from 'react-router-dom';

const Home = () => <div>Home Page</div>;

const About = () => <div>About Page</div>;

const Contact = () => <div>Contact Page</div>;

const App = () => {

return (

<Router>

<nav>

<Link to="/">Home</Link>

<Link to="/about">About</Link>

<Link to="/contact">Contact</Link>

</nav>

<Switch>

<Route exact path="/" component={Home} />

<Route path="/about" component={About} />

<Route path="/contact" component={Contact} />

</Switch>

</Router>

);

};

export default App;

```

### Q5. HTML and CSS (or Tailwind CSS) Cloning

\*\*Create a clone of the page using HTML and CSS or Tailwind CSS:\*\*

Sure, let's create a complete solution for question five using Tailwind CSS to clone the provided page.

Here is the complete HTML code using Tailwind CSS:

```html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Page Clone</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-100 p-6">

<div class="container mx-auto bg-white p-6 shadow-lg rounded">

<h1 class="text-3xl font-bold mb-4">Our Services</h1>

<div class="flex flex-col md:flex-row">

<div class="md:w-1/2 mb-4 md:mb-0">

<p class="text-lg">Any Text</p>

<ul class="list-disc list-inside mt-4">

<li class="mb-2">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</li>

<li class="mb-2">Phasellus imperdiet, nulla et dictum interdum, nisi lorem egestas odio.</li>

<li class="mb-2">Vitae scelerisque enim ligula venenatis dolor. Maecenas nisl est.</li>

<li class="mb-2">Euismod in, dolor. Integer scelerisque euismod magna.</li>

</ul>

</div>

<div class="md:w-1/2">

<img src="https://via.placeholder.com/300" alt="Image" class="w-full h-auto rounded shadow-md">

</div>

</div>

</div>

</body>

</html>

```

### Explanation

1. \*\*HTML Structure\*\*:

- \*\*Head Section\*\*: Includes the necessary meta tags and links to Tailwind CSS CDN.

- \*\*Body Section\*\*: The main content is wrapped in a `div` with `container mx-auto bg-white p-6 shadow-lg rounded` classes for styling.

2. \*\*Header\*\*:

- The header `<h1>` contains the title "Our Services" with appropriate Tailwind classes for styling.

3. \*\*Content Area\*\*:

- The content is divided into two main parts using `div` elements:

- \*\*Text Section\*\*: Uses `md:w-1/2 mb-4 md:mb-0` for responsive width and margin adjustments.

- \*\*Image Section\*\*: Uses `md:w-1/2` for responsive width and contains an image with Tailwind classes for styling.

4. \*\*Flexbox Layout\*\*:

- Uses `flex flex-col md:flex-row` to arrange the sections side by side on medium and larger screens, but stack them on smaller screens.

5. \*\*Additional Styling\*\*:

- The image is given additional classes like `rounded` and `shadow-md` for visual enhancements.

### Notes

- Replace `"https://via.placeholder.com/300"` with the actual image URL.

- You can adjust the content inside the text section and the image as needed to match the design requirements.

This solution leverages Tailwind CSS to achieve a clean and responsive design. Let me know if you need any more adjustments or explanations!