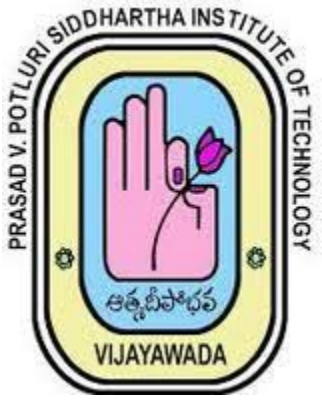
Prasad V. Potluri Siddhartha Institute of Technology

Autonomous, Kanuru,Vijayawada-07

Department of Computer Science and Engineering

MERN Stack Development

LABORATORY MANUAL



AICTE Approved, NBA and NACC A+ accredited, An ISO 9001: 2015 certified Institution, Permanent affiliation to JNTUK, Kakinada.

,

Institute Vision

To provide rich ambience for Academic and Professional Excellence, Research, Employability skills, Entrepreneurship and Social responsibility.

Institute Mission

To empower the students with Technical knowledge, Awareness of up-to-date technical trends, Inclination for research in the areas of human needs, Capacity building for Employment / Entrepreneurship, Application of technology for societal needs.

Department Vision

To be a center of excellence in academics and research in Computer Science and Engineering and take up challenges for the benefit of society.

Department Mission

Impart professional education through best curriculum in harmony with the industry needs.

Inculcate ethics, research capabilities and team work in the young minds so as to put efforts to the advancement of the nation.

Strive for student achievement and success with leadership qualities and preparing them for continuous learning in the global environment.

Program Educational Objectives

PEO-I:

The graduates of the program will excel in the concepts of basic engineering and advanced concepts of computer science engineering.

PEO-II:

The graduates of the program will be professional in computing industry or pursuing higher studies.

PEO-III:

The graduates of the program will excel in team work, ethics, communication skills and contribute to the benefit to the society.

Program Outcomes:

PO - 1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO - 2: Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO - 3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO - 4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO - 5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO - 6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO - 7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO - 8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO - 9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO - 10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO - 11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO - 12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO - I: Apply the Knowledge of Computing Skills in building the Software Systems that meet the requirements of Industry and Society.

PSO - II: Apply the Knowledge of Data Engineering and Communication Technologies for Developing Applications in the Domain of Smart and Intelligent Computing.

Syllabus

Syllabus

Course Code	20CS3653	**Year**	III	**Semester**	II
Course Category	PCC	**Branch**	CSE	**Course Type**	Practical
Credits	1.5	**L-T-P**	0-0-3	**Prerequisites**	Programming with JAVA
Continuous Internal Evaluation :	1.5	**Semester End Evaluation:**	35	**Total Marks:**	50

Course Outcomes		
Upon successful completion of the course, the student will be able to		
CO1	Apply MERN Technologies to develop Web Applications.	**L3**
CO2	Implement various applications as an individual or team member	**L3**
CO3	Develop an effective report based on various programs implemented	**L3**
CO4	Apply technical knowledge for a given problem and express with an effective oral communication	**L3**
CO5	Analyze outputs of web based applications	**L4**

Contribution of Course Outcomes towards achievement of Program Outcomes & Strength of correlations (3:Substantial, 2: Moderate, 1:Slight)

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1														3
CO2					1				2					
CO3										2				
CO4										3				
CO5			3											

Expt. No.	Contents	Mapped CO
1	Demonstration of const, let, string templates, callbacks, arrow functions, class, class-properties, methods using Java Script	CO1,CO2,CO3,CO4,CO5
2	Use of global object, Variables, Standard input and standard output in Node JS	CO1,CO2,CO3,CO4,CO5
3	to build application	CO1,CO2,CO3,CO4,CO5
4	Develop Node JS Application and Implement HTTP Services in Node JS (Request and Response)	CO1,CO2,CO3,CO4,CO5
5	Implement React Elements and Components	CO1,CO2,CO3,CO4,CO5
6	Develop Web application using React	CO1,CO2,CO3,CO4,CO5
7	Implement Read and write operations on database using MongoDb APIs	CO1,CO2,CO3,CO4,CO5
8	Developing a simple CRUD application using the MERN stack	CO1,CO2,CO3,CO4,CO5

Learning Resources

Text Books

1. Node.js, MongoDB and Angular Web Development by Brad Dayley, Brendan Dayley, Caleb Dayley, 2nd edition, Perason, ISBN-13: 978-0134655536, ISBN-10: 0134655532
2. Learning React Modern Patterns for Developing React Apps, 2nd Edition, Oreilly, ISBN: 978-1-492-D5172-5

References

1. Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node by Vasan Subramanian, Apress; 2nd ed. edition (13 May 2019), ISBN-10 : 1484243900, ISBN-13 : 978-1484243909
2. Mongodb, React, React Native Full-Stack Fundamentals and Beyond, by Eric Bush, Zaccheus Entertainment Publication, ISBN-10 : 0997196688, ISBN-13 : 978-0997196689
3. Quick Start Guide: Build web applications with MongoDB, Express.js, React, and Node by Eddy Wilson Iriarte Koroliova, Packt Publishing, ISBN-10: 1787281086, ISBN-13: 978-1787281080
4. Beginning Node.js, Express & MongoDB Development, by Greg Lim (Author), ISBN-10 : 9811480281
5. Beginning Node.js, Basarat Syed, APress, ISBN-10: 9781484201886

6. Learning React Functional Web Development with React and Redux by Alex Banks and Eve Porcello, Published by O'Reilly Media, Inc. , ISBN-10 : 9352135636, ISBN-13 : 978-9352135639
7. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL, by Azat Mardan, Published by Manning Publications, Manning Publications, ISBN-10 : 1617293342, ISBN-13 : 978-1617293344

e-Resources and other Digital Material

1. www.w3schools.com
2. https://www.javatpoint.com/mean-stack-tutorial
3. https://www.linode.com/docs/guides/mean-stack-tutorial/
4. https://blog.logrocket.com/mern-stack-tutorial/

PVP20 Regulations -

For Laboratory courses, there shall be continuous evaluation during the semester for 15 marks and semester end evaluation for 35 marks.

Distribution of Marks (CIE)

S.No.	Criterion	Marks
1	Day to Day Evaluation	5
2	Record	5
3	Internal Examination	5

Lab Rubrics -

Day to Day Evaluation (5M) + Record (5M)					
Procedure		**Execution**	**Viva-Voce**		**Record**
Apply (1M)	Self-learning (0.5M)	Individual Performance (1.5M)	Apply (1M)	Communication (1M)	Communication (5M)

Internal Examination (5M)				
Procedure		**Execution**	**Viva-Voce**	
Apply (1M)	Self-learning (0.5M)	Individual Performance (1.5M)	Apply (1M)	Communication (1M)

External Examination (35M)				
Procedure	**Execution**	**Viva**		**Output**
Apply (10M)	Individual Performance (10M)	Apply (7M)	Communication (3M)	Analysis (10M)

INDEXING

Experiment - 1

Aim: Demonstration of const, let, string templates, callbacks, arrow functions, class, class-properties, methods using Java Script.

Const: The const keyword was introduced in ES6 (2015).

1. Variables defined with const cannot be Redeclared.
2. Variables defined with const cannot be Reassigned.
3. Variables defined with const have Block Scope.
4. Constant name should be written in capital letters.

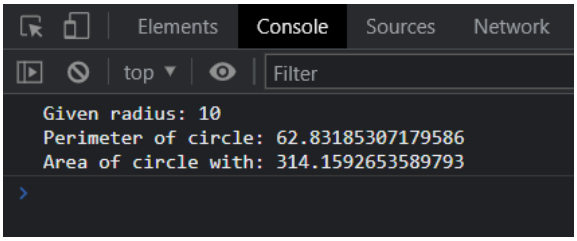
Syntax: const constant_name = value(initialization);

Example: const PI = 3.14

Program: const PI = Math.PI;const radius = 10;

console.log(`Given radius: ${radius}\nPerimeter of circle: ${2 * PI * radius}\nArea of circle with: ${PI * radius ** 2}`);

Output:



Let: The let keyword was introduced in ES6 (2015).

1. Variables defined with let cannot be Redeclared.
2. Variables defined with let must be Declared before use.
3. Variables defined with let have Block Scope.

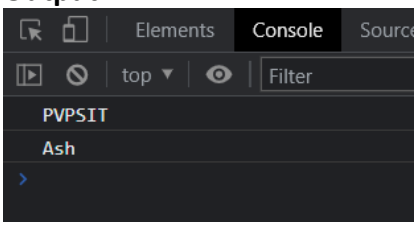
Syntax:

let variable_name = value;(value is optional)

Program:

```
let name = 'PVPSIT';
console.log(name);
name = 'Ash'
console.log(name);
```

Output:



string templates: Template Literals use back-ticks (``) rather than the quotes ("") to define a string.

String templates is also called as Template Literals (or) Template Strings (or) Back-Tics Syntax.

1. Template literals allows multiline strings

2. Template literals provide an easy way to interpolate variables and expressions into strings. The method is called **string interpolation.**
3. Template literals allow variables in strings
4. Template literals allow expressions in strings

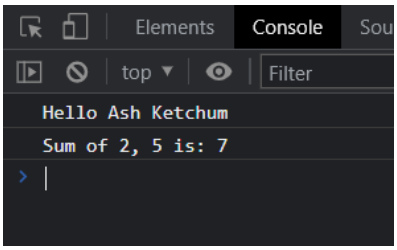
It is an ES6 feature (JavaScript 2015).

Syntax: ${variable/experssion}

Program:

```
const name = 'Ash Ketchum';
console.log(`Hello ${name}`); // variable
console.log(`Sum of 2, 5 is: ${2 + 5}`);    //expression
```

Output:



Callbacks: A callback is a function passed as an argument to another function

1. This technique allows a function to call another function
2. A callback function can run after another function has finished

These callbacks are of two types:

1. Synchronous callbacks
   Example: Just like normal function calls
2. Asynchronous callbacks
   Example: using setTimeout, setInterval functions which delay the function calls after executing.

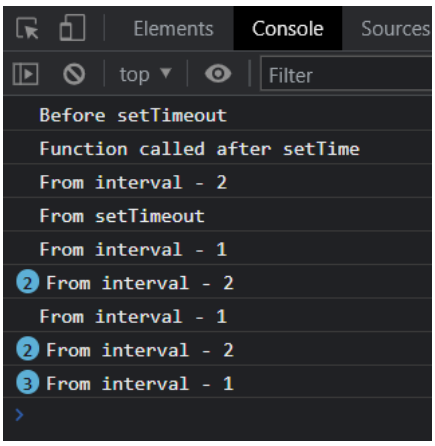
Syntax:
```
function function_name(callback) {
        // code
        callback();       // calling callback function

}
function fun() {
        // code

}
function_name(fun); // calling normal function
```

Program:

```
let c1 = 5, c2 = 5;
function fun() {
   console.log(`Function called after setTime`);
```

```
}
function myFunction() {
   c1--;
   if(c1 <= 0) clearInterval(interval1);
   console.log(`From interval - 1`);
}
console.log(`Before setTimeout`);
setTimeout(function() {
   console.log(`From setTimeout`);
}, 2000);
const interval1 = setInterval(myFunction, 3000);
const interval2 = setInterval(() => {
   c2--;
   if(c2 <= 0) clearInterval(interval2);
   console.log(`From interval - 2`);
}, 1500);
fun();
```

Output:



Arrow Function: Arrow functions were introduced in ES6.

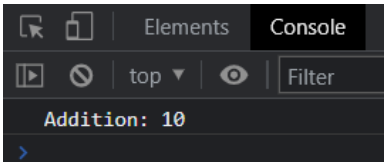
Arrow functions allow us to write shorter function syntax:

1. When there is only a single return statement in a function to execute then,

Program:

```
let add = (a, b) => a + b;
console.log(add(5, 5));
```

Output:

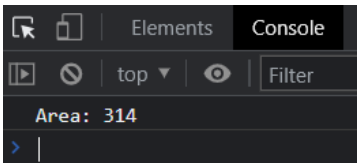


2. When there are more than one statements to execute in function then,

Program:

```
let area = (radius) => {
   const PI = 3.14;
   area = PI * radius * radius;
   return area;
}
console.log(area(10));
```

Output:



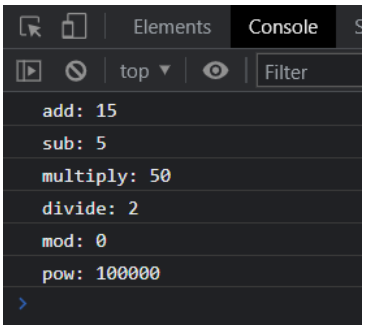
Classes: ECMAScript 2015, also known as ES6, introduced JavaScript Classes.

- JavaScript Classes are templates for JavaScript Objects.

```
class MyClass {
   constructor(a, b) {
      this.a = a;
      this.b = b;

   }
   add = () => this.a + this.b;    //type-1
   subtract = () => this.a - this.b;
   mulitply = () => {      //type-2
      return this.a * this.b;
   }
   divide() {      //type-3
      if (this.b === 0) {
         console.log(`Denominator cannot be zero`);
         return "";
      }
      return this.a / this.b;
   }

   mod = () => {
      if (this.b === 0) {
         console.log(`Denominator cannot be zero`);
         return "";
      }
```

```
        return this.a % this.b;
    }

    pow = () => {
            if (this.b === 0) {
                    console.log(`Denominator cannot be zero`);
                    return "";
            }
            return this.a ** this.b;
    }
}

const myClass = new MyClass(10, 5);  // Instance creation
console.log('add: ' + myClass.add());
console.log('sub: ' + myClass.subtract());
console.log('multiply: ' + myClass.mulitply());
console.log('divide: ' + myClass.divide());
console.log('mod: ' + myClass.mod());
console.log('pow: ' + myClass.pow());
```

Output:



Experiment - 2

Aim: Use of global object, Variables, Standard input and standard output in Node JS to build application.

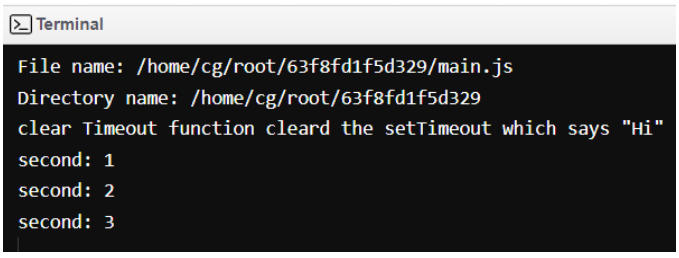
Global Objects: These objects are available in all modules. The following variables may appear to be global but are not. They exist only in the scope of modules.

1. __dirname
2. __filename
3. setTimeout
4. setInterval
5. require()

Program:

```
//1. __filename
console.log("File name: " + __filename);
//2. __dirname
console.log("Directory name: " + __dirname);
//3. setTimeout(callback(), time in ms)
const v = setTimeout(() => {
   console.log("Hi");
}, 1000);
//4. clearTimeout(timer_id)
setTimeout(() => {
   console.log(`clear Timeout function cleard the setTimeout which says "Hi"`);
   clearTimeout(v);
}, 100);
//5. setInterval
var s = 1;
const t = setInterval(() => {
   console.log("second: " + s++);
   if (s == 4) clearInterval(t);
}, 1000);
```

Output:



Standard input and standard output:

- stdin is the standard input stream and a source of input for the program.

- stdout is the standard output stream and a source of output for the program.
- stderr is the standard error stream and is used for error messages.

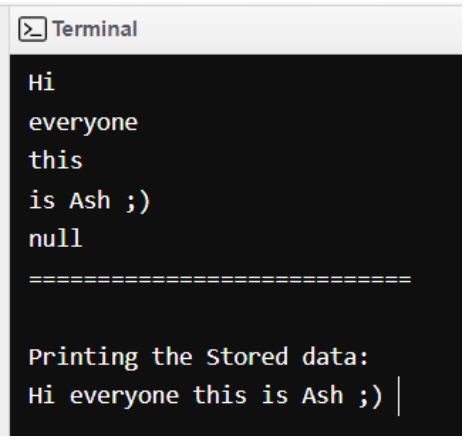
Syntax:

```
process.stdin.on(data, callbackFunction);
process.stdout.write(value);
```

Program:

```
var arr = [];
process.stdin.on("data", _ => {
    const givenData = _.toString().trim();
    if (givenData === "null") {
        data();
        process.exit();
    }
    arr.push(givenData);
})

function data() {
    console.log("\n============================\n\nPrinting the Stored data: ");
    for (let a of arr) { process.stdout.write(a + " ");
    }
};
```

Output:



Experiment - 3

Aim: Develop Node JS Application and Implement HTTP Services in Node JS (Request and Response)

Description:

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

- To include the HTTP module, use the require() method
- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client
- It create server using createServer() method
- If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type
- There will be two argument in callback function of createServer which are “req, res”
  req - is used to know the request from client.
  res - is used to send the response to the client.
- The first argument of the res.writeHead() method is the status code, the second argument is an object containing the response headers.

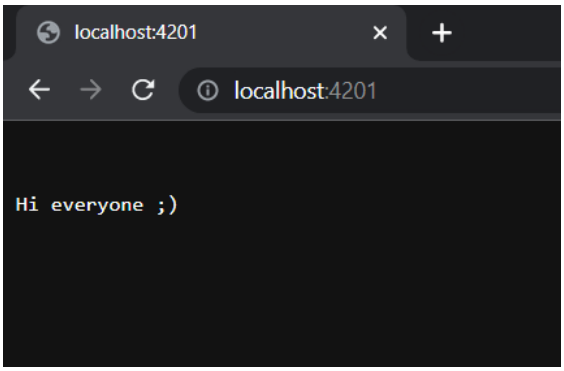
Syntax:

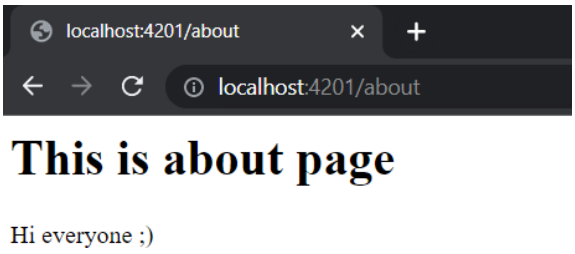
```
createServer(callbackFunction(req, res) {
        // code
}).listen(portNumber);
```

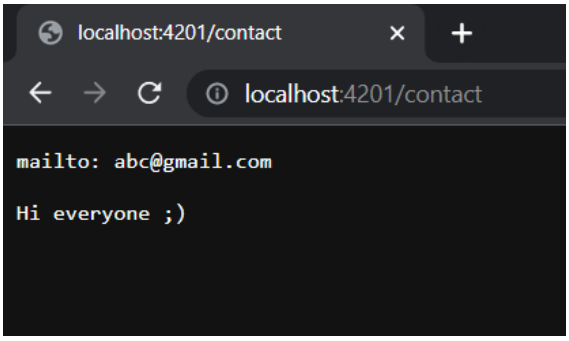
Program:

```
const { contentType } = require("express/lib/response");
const http = require(`http`);

http.createServer((req, res) => {
   res.writeHead(200, { contentType: 'text/plain, text/html' });
   if (req.url === '/about') res.write(`<h1> This is about page </h1>`);
   else if (req.url === '/contact') res.write(`mailto: abc@gmail.com`);
   res.end(`\n\nHi everyone ;)\n\n`);
}).listen(4201);
```

Output:


localhost:4201
localhost:4201
Hi everyone ;)


localhost:4201/about
localhost:4201/about
This is about page
Hi everyone ;)


localhost:4201/contact
localhost:4201/contact
mailto: abc@gmail.com
Hi everyone ;)

Experiment - 4

Aim: Implement React Elements and Components

Description:

React Components:

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components.
- When creating a React component, the component's name MUST start with an upper case letter.

Class Component

- A class component must include the **extends React.Component** statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.
- The component also requires a **render()** method, this method returns HTML.
- While implementing class component render method should must contain return statement. And it should return HTML code to render.

Component Constructor:

- If there is a **constructor()** function in your component, this function will be called whenthe component gets initiated.
- The constructor function is where you initiate the component's properties.
- In React, component properties should be kept in an object called state.

React Class Component State:

- React Class components have a built-in state object.
- Component properties should be kept in an object called state.
- In the component constructor function we have to invoke the super class constructor using **super(),** by this call we can access all the properties and functions in **React.Component** class, which provides **state** object.
- When the state object changes, the component re-renders.

Props:

- Components can be passed as props, which stands for properties.
- Props are like function arguments, and you send them into the component as attributes.
- This is another way of handling properties in an component
- Here for class component props should be pass through constructor for the super class which is **React.Component,** to initialize the props in that class.

Changing the state Object:

- To change a value in the state object, use **this.setState()** method.
- When a value in the state object changes, the component will re-render, meaning thatthe output will change according to the new value(s).
- Always use the **setState()** method to change the state object, it will ensure that the component knows it's been updated and calls the render() method (and all the other lifecycle methods).

Functional Component

- A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, and are easier to understand.
- It uses the Hooks concept to update the DOM element similar to the setState() methodin a class, but with different uses.

Program: To update the React component every second

Timer component:

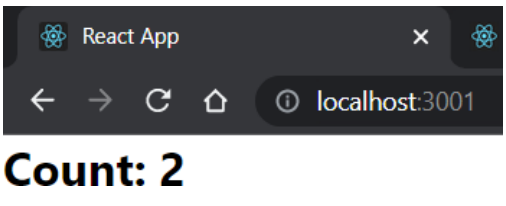
```
import React from 'react';
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
    count: 0,
    };

  }
  componentDidMount() {
    this.interval = setInterval(() => {
      this.setState({
        count: this.state.count + 1,
      });
    }, 1000);
  }

  componentWillUnmount() {
    clearInterval(this.interval);
  }
  render() {
    return (
      <div>
```

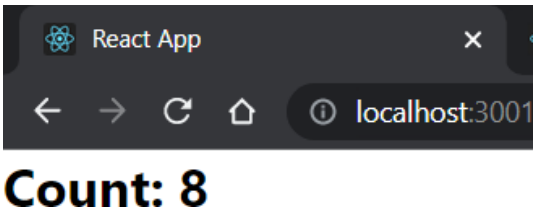
```
        <h1>Coun: {this.state.count}</h1>
      </div>
    );

  }
}

export default Timer;
```

index.js file:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Timer from './components/Timer';
const root = ReactDOM.createRoot(document.getElementById(`root`));
root.render(<Timer />);
```

Output: A count variable which increment by one for every second.





Program:

Counter component:

```
import React, { Component } from 'react';
import pikachu from './pikachu.png';

class Counter extends Component {
  constructor() {
    super();
    this.render = this.render.bind(this);
    console.log(this);
    this.html = '';

  }
  state = {
    count: 0,
    imageUrl: pikachu,
  };
```

```
styles = {
  boxSizing: 'border-box',
  fontSize: '24px',
  padding: '0.8%',
  backgroundColor: '#FF2F10',
  borderRadius: '20px',
  margin: '1%',
};

buttonStyle = {
  backgroundColor: 'yellow',
  border: 'none',
  borderRadius: '10px',
  height: '44px',
  fontSize: '24px',
  width: 'fit-content',
  cursor: 'pointer',
  margin: '1%',
};
incrementCount = () => {
  this.setState({ count: this.state.count + 1 });
};
decrementCount = () => {
  this.state.count <= 0
    ? alert(`Can't decrease than zero`)
    : this.setState({ count: this.state.count - 1 });
};
formatCount() {
const { count } = this.state;
  return count === 0 ? 'Zero' : count;
}
render() {
  return (
    <div>
      <span style={this.styles} className="badge">
        {this.formatCount()}
      </span>
      <img style={this.hidden} src={pikachu} alt="Pikachu" />
      <button onClick={this.incrementCount} style={this.buttonStyle}>
        Increment
      </button>
      <button onClick={this.decrementCount} style={this.buttonStyle}>
        Decrement
      </button>
    </div>
```

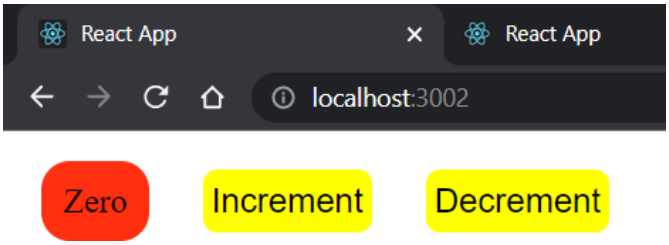
```
    );
  } }
 export default Counter;
```

index.js

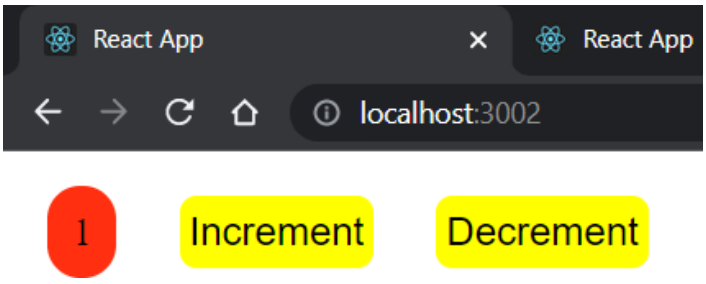
```
 import React from 'react';
 import ReactDOM from 'react-dom/client';
 import Counter from './components/counter';

 const root = ReactDOM.createRoot(document.getElementById('root'));
 root.render(
    <Counter />

 );
```

Output:

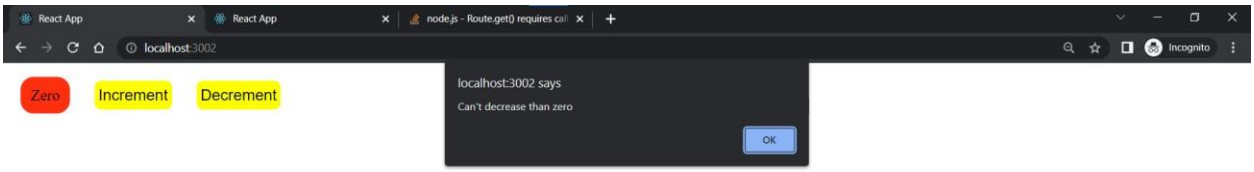
Initial:



On clicking Increment button:



On clicking Decrement button twice:



Experiment - 5

Aim: Develop Web application using React.

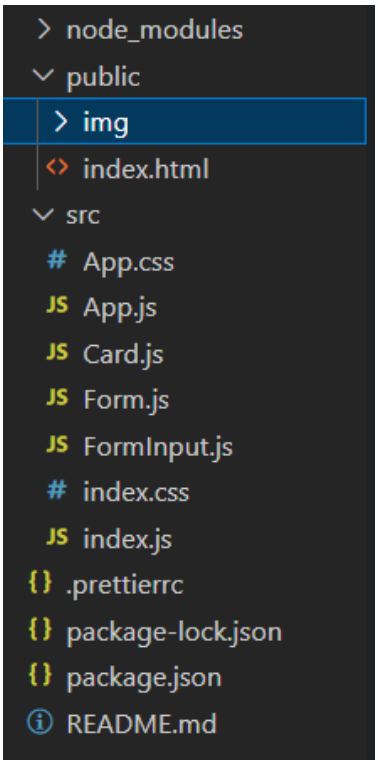
Description:

- This is a comic display web app where it will request an API to get the data it required using the **'fetch'** method

Syntax: fetch(url, {options}).then().catch();

Options: This will include the **METHOD (**GET, POST, PATCH, PUT, DELETE**)** and **headers** which will help to transfer data between API and our web app, and finally **body** where we can send any required content to process our request.

Project Structure:



Index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
```

```
import reportWebVitals from './reportWebVitals';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
reportWebVitals();
```

App.js

```
/* eslint-disable no-restricted-globals */
import { useEffect, useState } from 'react';
import './App.css';
import Form from './Form';
import Card from './Card';
function App() {
  const [data, setData] = useState([]);
  const [isCreate, setisCreate] = useState(false);
  const [isUpdate, setisUpdate] = useState(false);
  const [isDelete, setisDelete] = useState(false);
  useEffect(() => {
    // get api
    const url = `http://localhost:2401/api/v1/comics`;
    fetch(url)
      .then(res => res.json())
      .then(res => setData(res.data))
      .catch(err => console.log(err));

  });
  const create = () => {
    const bodyData = {
      name: document.getElementById('name').value,
      author: document.getElementById('author').value,
      type: document.getElementById('type').value,
      language: document.getElementById('language').value,
      country: document.getElementById('country').value,
      price: Number(document.getElementById('price').value) || 100
    };
    const url = `http://localhost:2401/api/v1/comics`;
    fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Access-Control-Allow-Origin':  '*',
```

```
      Accept: '*'
    },
    body: JSON.stringify(bodyData)
  })
    .then(res => res.json())
    .then(res => {
      setData(res.data);
    })
    .then(() => confirm('Created successfully'))
    .catch(err => console.log(err));
};
const update = () => {
  const name = document.getElementById('name').value;
  const author = document.getElementById('author').value;
  const type = document.getElementById('type').value;
  const language = document.getElementById('language').value;
  const country = document.getElementById('country').value;
  const price = document.getElementById('price').value;
  const url = `http://localhost:2401/api/v1/comics/-1/${name}`;
  const bodyData = {};
  if (name) bodyData.name = name;
  if (author) bodyData.author = author;
  if (type) bodyData.type = type;
  if (language) bodyData.language = language;
  if (country) bodyData.country = country;
  if (price) bodyData.price = price;
  console.log('name: ', name);
  fetch(url, {
    method: 'PATCH',
    headers: {
      'Content-Type': 'application/json',
      'Access-Control-Allow-Origin': '*',
      Accept: '*',
      charset: 'UTF-8'
    },
    body: JSON.stringify(bodyData)
  })
    .then(res => res.json())
    .then(res => {
      setData(res.data);
    })
    .then(() => confirm('Updated successfully'))
    .catch(err => console.log(err));

};
```

```
const deleteComic = () => {
  const name = document.getElementById('name').value;
  const url = `http://localhost:2401/api/v1/comics/-1/${name}`;
  fetch(url, {
    method: 'DELETE'
  })
    .then(res => res.json())
    .then(res => {
      setData(res.data);
    })
    .then(() => confirm('Deleted successfully'))
    .catch(err => console.log(err));

};
const createForm = () => {
  setisCreate(isCreate => !isCreate);
};
const updateForm = () => {
  setisUpdate(isUpdate => !isUpdate);
};
const deleteFrom = () => {
  setisDelete(isDelete => !isDelete);
};
return (
  <>
    <h1>Welcome</h1>
    <br />
    <button onClick={createForm}>Create</button>
    <button onClick={updateForm}>Update</button>
    <button onClick={deleteFrom}>Delete</button>
    {isCreate ? (
      <div>
        <Form caption="Create" />
        <button onClick={create}>Submit</button>
      </div>
    ) : null}
    {isUpdate ? (
      <div>
        <Form caption="Update" />
        <button onClick={update}>Submit</button>
      </div>
    ) : null}
    {isDelete ? (
      <div>
        <table>
```

```
          <caption>Delete</caption>
          <tbody>
           <tr>
            <td>
             <label htmlFor="name">Name</label>
            </td>
            <td>
             <input type="text" name="name" id="name" />
            </td>
           </tr>
          </tbody>
         </table>
         <button onClick={deleteComic}>Submit</button>
        </div>
       ) : null}
       <div className="cards">
        {data.map(item => {
         return (
          <Card
           key={item.name}
           photo={item.photo ? item.photo : '/img/default.jpeg'}
           name={item.name}
           language={item.language}
           price={item.price}
          />
         );
        })}
       </div>
     </>
   );
}
export default App;
```

Card.js (Component)

```
/* eslint-disable jsx-a11y/anchor-is-valid */
function Card(props) {
 return (
  <div className="card col">
   <img className="comic-image content" src={props.photo} alt={props.name} />
   <div className="comic-name content">Name: {props.name}</div>
   <div className="comic-language content">Language: {props.language}</div>
   <div className="comic-price content">Price: {props.price}</div>
   <a href="#" className="content">
```

```
        Details
      </a>
    </div>
  ); }
 export default Card;
```

Form.js (Component)

```
 // import FormInput from './FormInput';

 function Form(props) {
   return (
     <form>
       <table>
         <caption>{props.caption}</caption>
         <tbody>
           <tr>
             <td className="name">
               <label htmlFor="name">Name: </label>
             </td>
             <td>
               <input type="text" name="name" required id="name" />
             </td>
           </tr>
           <tr>
             <td>
               <label htmlFor="author">Author: </label>
             </td>
             <td>
               <input type="text" name="author" required id="author" />
             </td>
           </tr>
           <tr>
             <td>
               <label htmlFor="type">Type: </label>
             </td>
             <td>
               <input type="text" name="type" required id="type" />
             </td>
           </tr>
           <tr>
             <td>
               <label htmlFor="country">Country: </label>
             </td>
```

```
          <td>
            <input type="text" name="country" required id="country" />
          </td>
        </tr>
        <tr>
          <td>
            <label htmlFor="language">Language: </label>
          </td>
          <td>
            <input type="text" name="language" required id="language" />
          </td>
        </tr>
        <tr>
          <td>
            <label htmlFor="price">Price: </label>
          </td>
          <td>
            <input type="number" name="price" id="price" />
          </td>
        </tr>
      </tbody>
    </table>
  </form>
 );  }
export default Form;
```

App.css

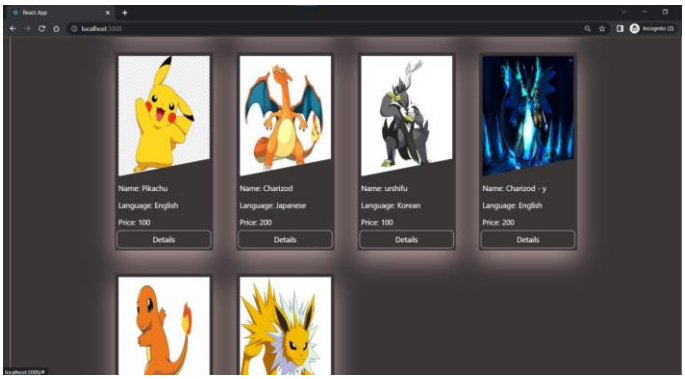
```
* {
  box-sizing: border-box;
  background-color: rgb(59, 54, 54);
  color: white;
}
.App {
  text-align: center;
  display: flex;
}
.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
```

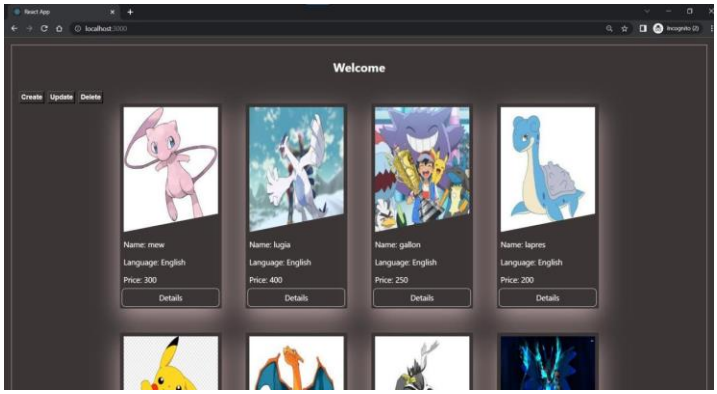
```
    animation: App-logo-spin infinite 20s linear;
  }
 }

 .App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
 }
 .App-link {
  color: #61dafb;

 }
 @keyframes App-logo-spin {
  from {
   transform: rotate(0deg);
  }
  to {
   transform: rotate(360deg);

  }
 }
 #root {
  margin: 1%;
  padding: 15px;
  border: 2px solid rgb(255, 255, 255, 0.4);
  /* background-color: #282c34; */
 }
 h1 {
  text-align: center;
  font-weight: bold;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
 }
 .cards {
  max-width: 80rem;
  margin: 0 auto;
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  grid-gap: 4rem;
 }
```

```
.card {
  border-radius: 3px;
  overflow: hidden;
  -webkit-box-shadow: 0 1.5rem 4rem rgb(0, 199, 249);
  box-shadow: 0 1.5rem 4rem rgb(240, 207, 207, 0.7);
  -webkit-transition: 0.3s all;
  transition: 0.3s all;
  -webkit-backface-visibility: hidden;
  backface-visibility: hidden;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  -webkit-box-orient: vertical;
  -webkit-box-direction: normal;
  -ms-flex-direction: column;
  flex-direction: column;
}

.content {
  font-size: 1.3rem;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
  -webkit-box-align: center;
  -ms-flex-align: center;
  align-items: center;
  padding: 10px;
  }
.comic-image {
  position: relative;
  -webkit-clip-path: polygon(0 0, 100% 0%, 100% 83%, 0% 98%);
  clip-path: polygon(0 0, 100% 0%, 100% 83%, 0% 98%);
  height: 22rem;
}

table {
  display: inline;
}

button {
  margin: 0 0.24%;
  padding: 4px;
  font-size: large;
  font-weight: bold;
  cursor: pointer;
```

```
  -webkit-user-select: none;
  user-select: none;
}
button:hover {
  font-weight: bold;
  background-color: white;
  color: black;
  box-shadow: 2px 2px
  white, -2px -2px wheat;
  border-radius: 5px;
}

label {
  -webkit-user-select: none;
  user-select: none;
}

a {
  margin: 0 2% 2%;
  text-decoration: none;
  border: 1px solid white;
  border-radius: 10px;
  justify-content: center;
}

a:hover {
  background-color: white;
  color: black;
  cursor: pointer;
  font-weight: bolder;
}
```

Web page after running the script:





Experiment - 6

Aim: Implement Read/Write operations on database using MongoDB API's

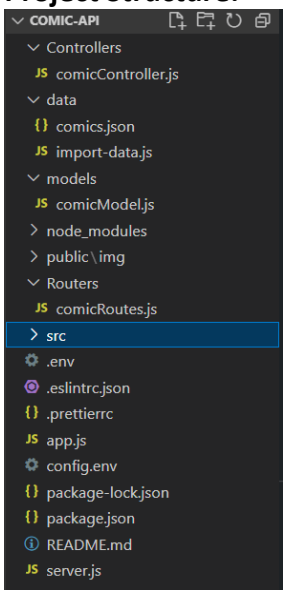
Description:

1. Install MongoDB community server and MongoDB shell into our system
2. Add mongoDB bin folder to environmental variables
3. Open CMD and type **mongosh** to enter into mongo shell

Now perform CRUD operations

1. First create database to store the collections.
2. Use the created database to access it
3. Create a collection to store the records
4. Insert the records using **insertOne()** or **insertMany()** function.

Project Structure:



ComicController.js

```
const Comic = require('../models/comicModel');

exports.getAllComics = async (req, res) => {
  const comics = await Comic.find().sort('-createdAt');
  res.status(200).json({
    // status: 'success',
    // results: comics.length,
    data: comics
  });
};
```

```
exports.getComic = async (req, res) => {
  if (!req.params.id)
    req.params.id = (await Comic.findOne({name: req.body.name}))._id;
  const comic = await Comic.find(req.params.id);
  if (!comic) {
    return res.status(404).json({
      status: 'fail',
      message: `Comic with given id: ${req.params.id} not present`
    });
  }
  res.status(200).json({
    status: 'success',
    data: comic
  });
};

exports.createComic = async (req, res) => {
  await Comic.create(req.body);
  const comics = await Comic.find().sort('-createdAt');
  // console.log(req.headers);
  res.status(201).json({
    // status: 'success',
    data: comics
  });
};

exports.updateComic = async (req, res) => {
  try {
    let id;
    console.log(req.params.id, req.params.name);
    if (req.params.id === undefined || req.params.id < 0)
      id = await Comic.findOne({
        name: req.params.name === undefined ? req.body.name : req.params.name
      }).select('_id');
    else id = req.params.id;
    console.log('Hi');
    console.log(id);
    console.log(req.body);
    await Comic.findByIdAndUpdate({_id: id._id}, req.body, {
      new: true,
      runValidators: true
    });
    const comics = await Comic.find().sort('-createdAt');
    res.status(200).json({
      data: comics
```

```
    });
  } catch (err) {
    res.status(404).json({
      status: fail,
      data: []
    });
  }
};

exports.deleteComic = async (req, res) => {
  try {
    let id;
    if (req.params.id < 0) id = await Comic.findOne({name: req.params.name});
    else id = req.params.id;
    await Comic.findByIdAndDelete({_id: id._id});
    const comics = await Comic.find().sort('-createdAt');
    res.status(200).json({
      status: 'success',
      data: comics
    });
  } catch (err) {
    res.status(404).json({
      status: fail,
      data: []
    });
  }
};
```

comics.json

```
[
  {
    "name": "Pikachu",
    "author": "Ash Ketchum",
    "type": "Electric",
    "language": "English",
    "country": "Japan",
    "price": 100,
    "photo": "/img/pikachu.jpeg"
  }
]
```

Import-data.js

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const fs = require('fs');
```

```
const Comic = require('../models/comicModel');

dotenv.config({path: '../.env'});

const allComics = JSON.parse(
  fs.readFileSync(`${__dirname}/comics.json/`, 'utf-8')
);

const DB = process.env.DATABASE.replace(
  '<PASSWORD>',
  process.env.DATABASE_PASSWORD
);

mongoose
  .connect(DB)
  .then(() => console.log('DB connection Successful'))
  .catch(err => console.log(err));

const importData = async () => {
  try {
    await Comic.create(allComics);
    console.log('Data written successfully');
  } catch (err) {
    console.log(err);
  }
  process.exit();
};

const deleteData = async () => {
  try {
    await Comic.deleteMany();
    console.log('Data deleted successfully');
  } catch (err) {
    console.log(err);
  }
  process.exit();
};

if (process.argv[2] === '--i') {
  importData();
} else if (process.argv[2] === '--d') {
  deleteData();
}
```

ComicModel.js

```
const mongoose = require('mongoose');

const comicSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Comic must contain name'],
    unique: true
  },
  author: {
    type: String,
    required: [true, 'Comic must contain author']
  },
  type: {
    type: [String],
    required: [true, 'Comic must be a certain type']
  },
  photo: String,
  language: {
    type: String,
    required: [true, 'Comic must consist a language']
  },
  country: {
    type: String,
    required: [true, 'Comic must belong to some country']
  },
  price: {
    type: Number,
    default: 100
  },
  createdAt: {
    type: Date,
    default: Date.now()
  }
});

const Comic = mongoose.model('Comic', comicSchema);

module.exports = Comic;
```

ComicRouter.js

```
const express = require('express');
const comicController = require('../Controllers/comicController');

const router = express.Router();
```

```
router
  .route('/')
  .get(comicController.getAllComics)
  .post(comicController.createComic)
  .patch(comicController.updateComic)
  .delete(comicController.deleteComic);

router
  .route('/:id/:name?')
  .get(comicController.getComic)
  .patch(comicController.updateComic)
  .delete(comicController.deleteComic);

module.exports = router;
```

app.js

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

const comicRouter = require('./Routers/comicRoutes');

const app = express();

app.use(cors());
app.use(express.json());
app.use(express.urlencoded({extended: true}));

app.use('/api/v1/comics', comicRouter);

module.exports = app;
```

server.js

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const PORT = process.env.PORT || 2401;

const app = require('./app');

dotenv.config({path: './.env'});

process.on('uncaughtException', err => {
  console.log(err.name, err.message);
  console.log('Uncaught Exception 💥 Shutting down');
```

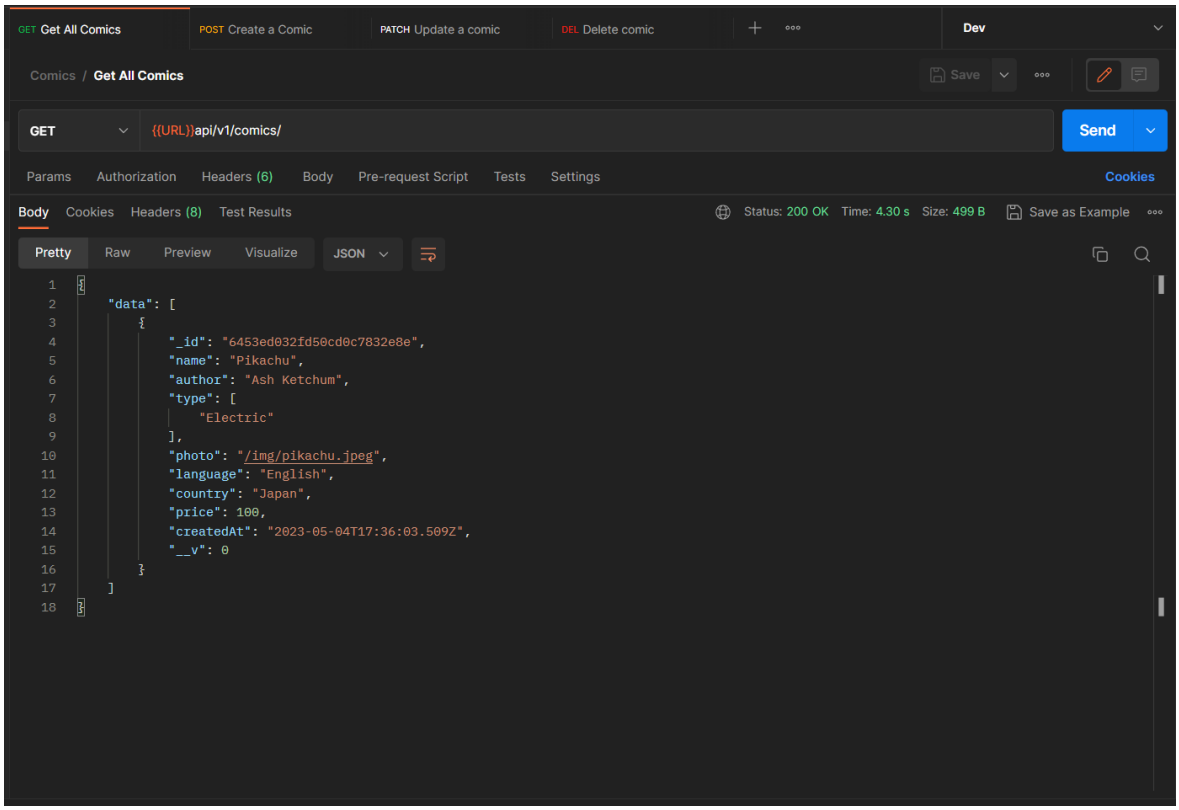
```
  process.exit(1);
});

const DB = process.env.DATABASE.replace(
  '<PASSWORD>',
  process.env.DATABASE_PASSWORD
);

mongoose
  .connect(DB)
  .then(() => console.log('Database connected successfully'))
  .catch(err => console.log(err));

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

.env:

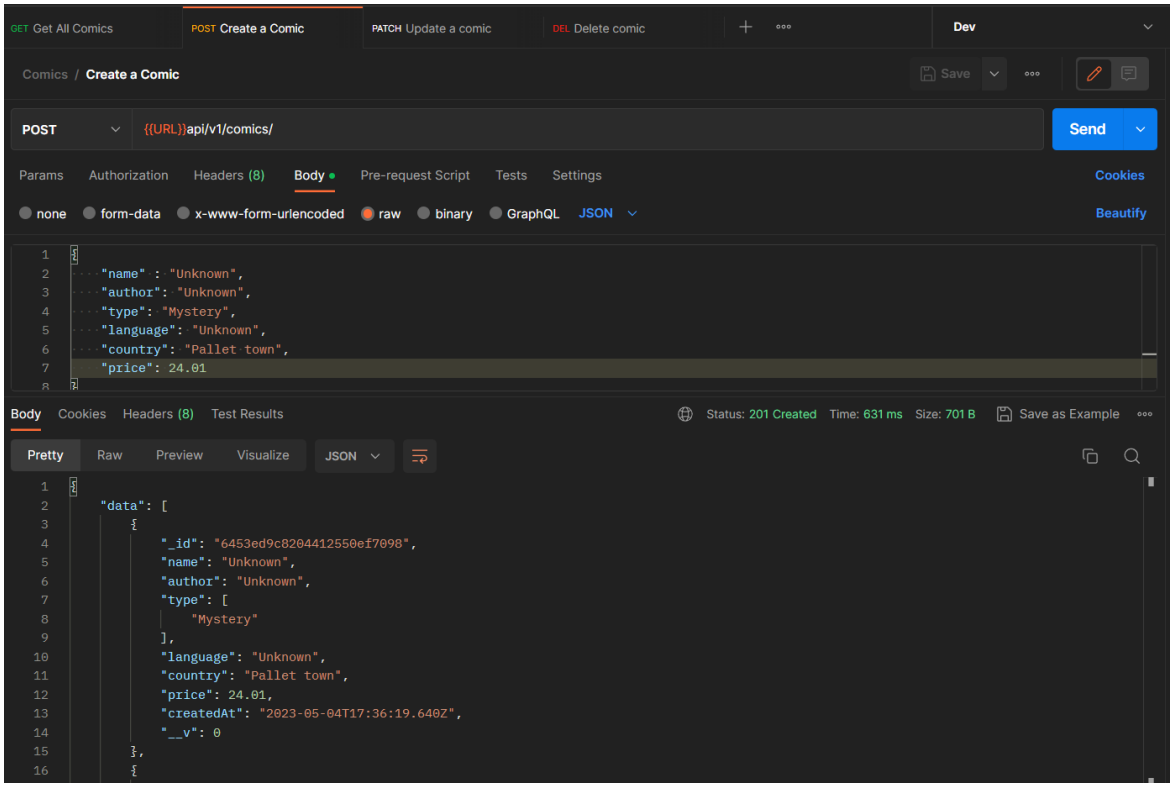
DATABASE = mongodb+srv://username:<PASSWORD>@cluster0.xqbjiob.mongodb.net/test
DATABASE_PASSWORD = fNIcGf8NocbFlCBT

- {{URL}} = http://localhost:2401

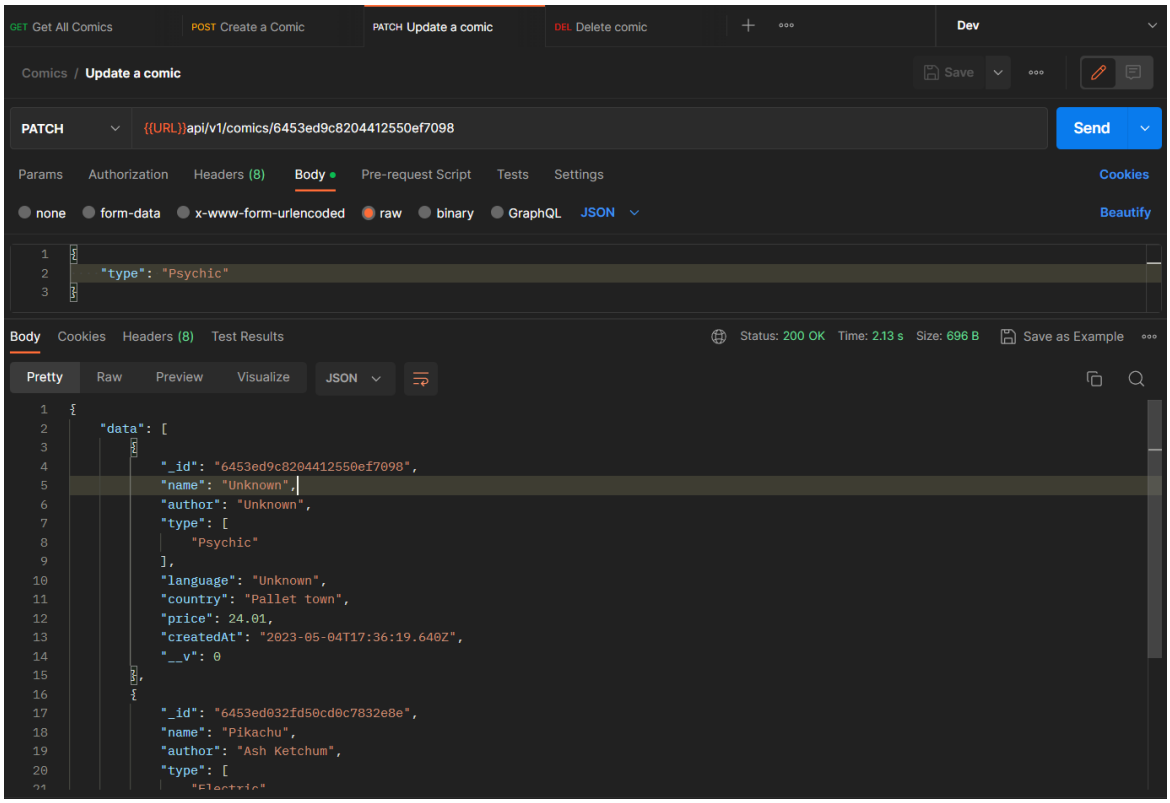
Read/Get all data:



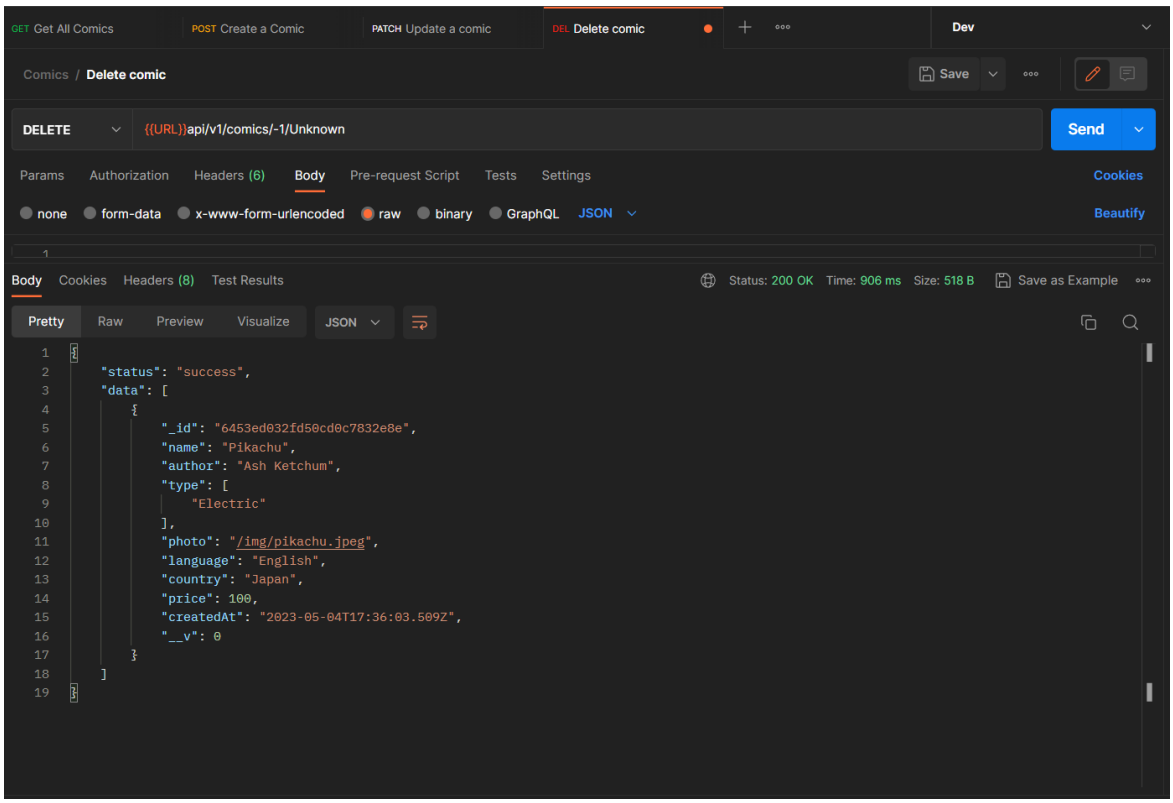
Create data:



Update data:



Delete data:



Experiment – 7

Aim: Developing a simple CRUD application using the MERN stack

Description:

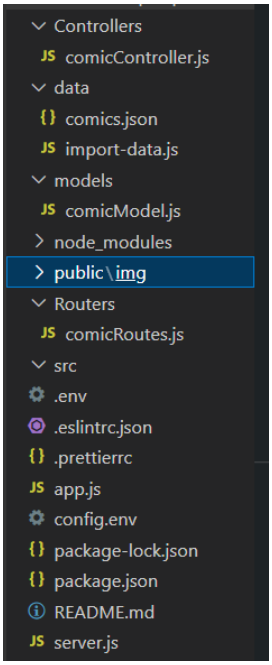
Building an API which gives a JSON data when requested by any user.

Using MongoDB to store the data into database by connecting using MongoDB mongoose driver

This is extension to the above developed front-end

Using Express JS to serve as a server when request came.

API Project Structure:



Server.js

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const PORT = process.env.PORT || 2401;
const app = require('./app');

dotenv.config({path: './.env'});
process.on('uncaughtException', err => {
  console.log(err.name, err.message);
  console.log('Uncaught Exception fu   Shutting
  down');process.exit(1);
});


const DB = process.env.DATABASE.replace(
  '<PASSWORD>',
```

```
  process.env.DATABASE_PASSWORD
 );


 mongoose
  .connect(DB)
  .then(() => console.log('Database connected successfully'))
  .catch(err => console.log(err));
 app.listen(PORT, err => {
  if (err) console.log(err);
  else console.log(`Server is running on port ${PORT}`);
 });
```

App.js

```
 const express = require('express');
 const mongoose = require('mongoose');
 const comicRouter = require('./Routers/comicRoutes');
 const cors = require('cors');
 const app = express();
 app.use(cors());
 app.use(express.json());
 app.use(express.urlencoded({extended: true}));
 app.use('/api/v1/comics', comicRouter);
 module.exports = app;
```

.env:

```
 DATABASE = mongodb+srv://mohan:<PASSWORD>@cluster0.xqbjiob.mongodb.net/test
 DATABASE_PASSWORD = temporary-pass
```

comicRouter.js

```
 const express = require('express');
 const comicController = require('../Controllers/comicController');
 const router = express.Router();

 router
  .route('/')
  .get(comicController.getAllComics)
  .post(comicController.createComic)
  .patch(comicController.updateComic)
  .delete(comicController.deleteComic);
 router
  .route('/:id/:name?')
  .get(comicController.getComic)
  .patch(comicController.updateComic)
```

```
  .delete(comicController.deleteComic);
module.exports = router;
```

comicModel.js

```
const mongoose = require('mongoose');

const comicSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Comic must contain name'],
    unique: true
  },
  author: {
    type: String,
    required: [true, 'Comic must contain author']
  },
  type: {
    type: [String],
    required: [true, 'Comic must be a certain type']
  },
  photo: String,
  language: {
    type: String,
    required: [true, 'Comic must consist a language']
  },
  country: {
    type: String,
    required: [true, 'Comic must belong to some country']
  },
  price: {
    type: Number,
    default: 100
  },
  createdAt: {
    type: Date,
    default: Date.now()

  }
});
const Comic = mongoose.model('Comic', comicSchema);
module.exports = Comic;
```

comics.json:

```
[
```

```
{
  "name": "Pikachu",
  "author": "Ash Ketchum",
  "type": "Electric",
  "language": "English",
  "country": "Japan",
  "price": 100,
  "photo": "/img/pikachu.jpeg"

},
{
  "name": "Charizod",
  "author": "Ash Ketchum",
  "type": ["Flying, Fire"],
  "language": "Japanese",
  "country": "Japan",
  "price": 200,
  "photo": "/img/Charizod.jpeg"

},
{

  "name": "mew",
  "author": "Misty",
  "type": ["Fairy", "Psychic"],
  "language": "English",
  "country": "Japan",
  "price": 300,
  "photo": "/img/mew.jpeg"
},
{

  "name": "lugia",
  "author": "Go",
  "type": ["Water", "Flying"],
  "language": "English",
  "country": "Japan",
  "price": 400,
  "photo": "/img/lugia.jpeg"
},
```

```
{
 "name": "gallon",
 "author": "Author",
 "type": "Adventures",
 "language": "English",
 "country": "Japan",
 "price": 100,
 "photo": "/img/gallon.jpeg"

},
{

 "name": "lapres",
 "author": "Brock",
 "type": ["Ice", "Water"],
 "language": "English",
 "country": "Japan",
 "price": 200,
 "photo": "/img/lapres.jpeg"
},
{

 "name": "charmilion",
 "author": "Nurse Joy",
 "type": "Fire",
 "language": "English",
 "country": "Japan",
 "price": 300,
 "photo": "/img/charmilion.jpeg"
},
{
 "name": "zordion",
 "author": "Prof. Oak",
 "type": "Electric",
 "language": "English",
 "country": "Japan",
 "price": 400,
 "photo": "/img/zordion.jpeg"

},
{

 "name": "urshifu",
 "author": "James",
 "type": "Fighting",
```

```
    "language": "Korean",
    "country": "Japan",
    "price": 100,
    "photo": "/img/urshifu.jpeg"
  },
  {

    "name": "Charizod - y",
    "author": "Leon",
    "type": "Firy",
    "language": "English",
    "country": "Japan",
    "price": 200,
    "photo": "/img/charizod-y.jpeg"
  }
]
```

Import-data.js:

```
const mongoose = require('mongoose');
const dotenv = require('dotenv');
dotenv.config({path: '../.env'});
const fs = require('fs');
const Comic = require('../models/comicModel');


const allComics = JSON.parse(
  fs.readFileSync(`${__dirname}/comics.json/`, 'utf-8')
);


const DB = process.env.DATABASE.replace(
  '<PASSWORD>',
  process.env.DATABASE_PASSWORD
);


mongoose
  .connect(DB)
  .then(() => console.log('DB connection Successful'))
  .catch(err => console.log(err));


const importData = async () => {
  try {
    await Comic.create(allComics);
```

```
    console.log('Data written successfully');
  } catch (err) {
    console.log(err);
  }

  process.exit();
};


const deleteData = async () => {
  try {
    await Comic.deleteMany();
    console.log('Data deleted successfully');
  } catch (err) {
    console.log(err);
  }
  process.exit();
};


if (process.argv[2] === '--i') {
  importData();
} else if (process.argv[2] === '--d') {
  deleteData();
}
```

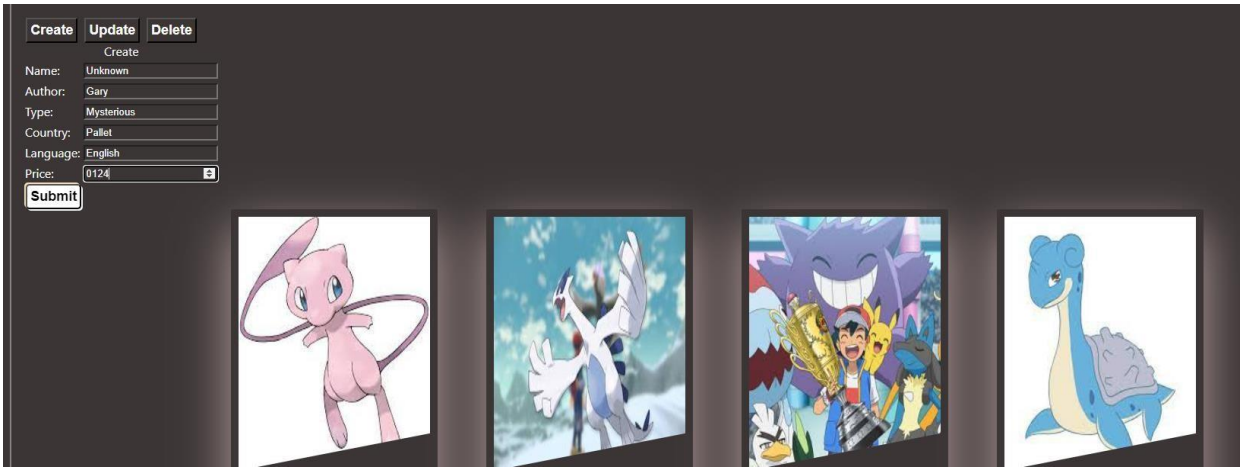
comicController.js:

```
const Comic = require('../models/comicModel');


exports.getAllComics = async (req, res) => {
  const comics = await Comic.find().sort('-createdAt');
  res.status(200).json({
    // status: 'success',
    // results: comics.length,
    data: comics
  });
};

exports.getComic = async (req, res) => {
  if (!req.params.id)
    req.params.id = (await Comic.findOne({name: req.body.name}))._id;
  const comic = await Comic.find(req.params.id);
  if (!comic) {
    return res.status(404).json({
```

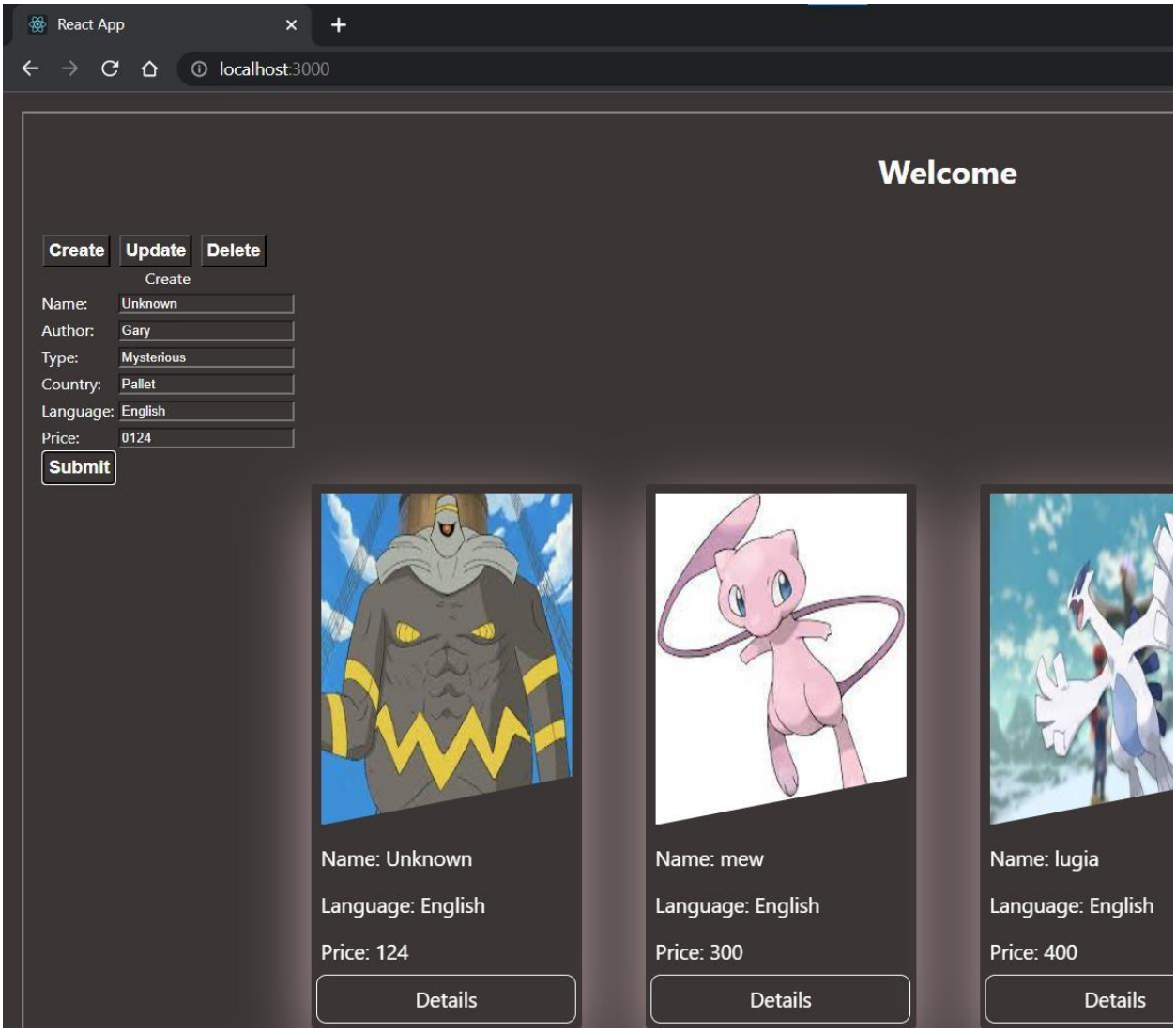
```
      status: 'fail',
      message: `Comic with given id: ${req.params.id} not present`
    });

  }
  res.status(200).json({
    status: 'success',
    data: comic
  });
};


exports.createComic = async (req, res) => {
  await Comic.create(req.body);
  const comics = await Comic.find().sort('-createdAt');
  // console.log(req.headers);
  res.status(201).json({
    // status: 'success',
    data: comics
  });
};

exports.updateComic = async (req, res) => {
  try {
    let id;
    if (req.params.id < 0) id = await Comic.findOne({name: req.params.name});
    else id = req.params.id;
    await Comic.findByIdAndUpdate({_id: id._id}, req.body, {
      new: true,
      runValidators: true
    });
    const comics = await Comic.find().sort('-createdAt');
    res.status(200).json({
      data: comics
    });
  } catch (err) {
    res.status(404).json({
      status: fail,
      data: []
    });

  }
};

exports.deleteComic = async (req, res) => {
```

```
  try {
    let id;
    if (req.params.id < 0) id = await Comic.findOne({name: req.params.name});
    else id = req.params.id;
    await Comic.findByIdAndDelete({_id: id._id});
    const comics = await Comic.find().sort('-createdAt');
    res.status(200).json({
      status: 'success',
      data: comics
    });
  } catch (err) {
    res.status(404).json({
      status: fail,
      data: []
    });

  }
};
```

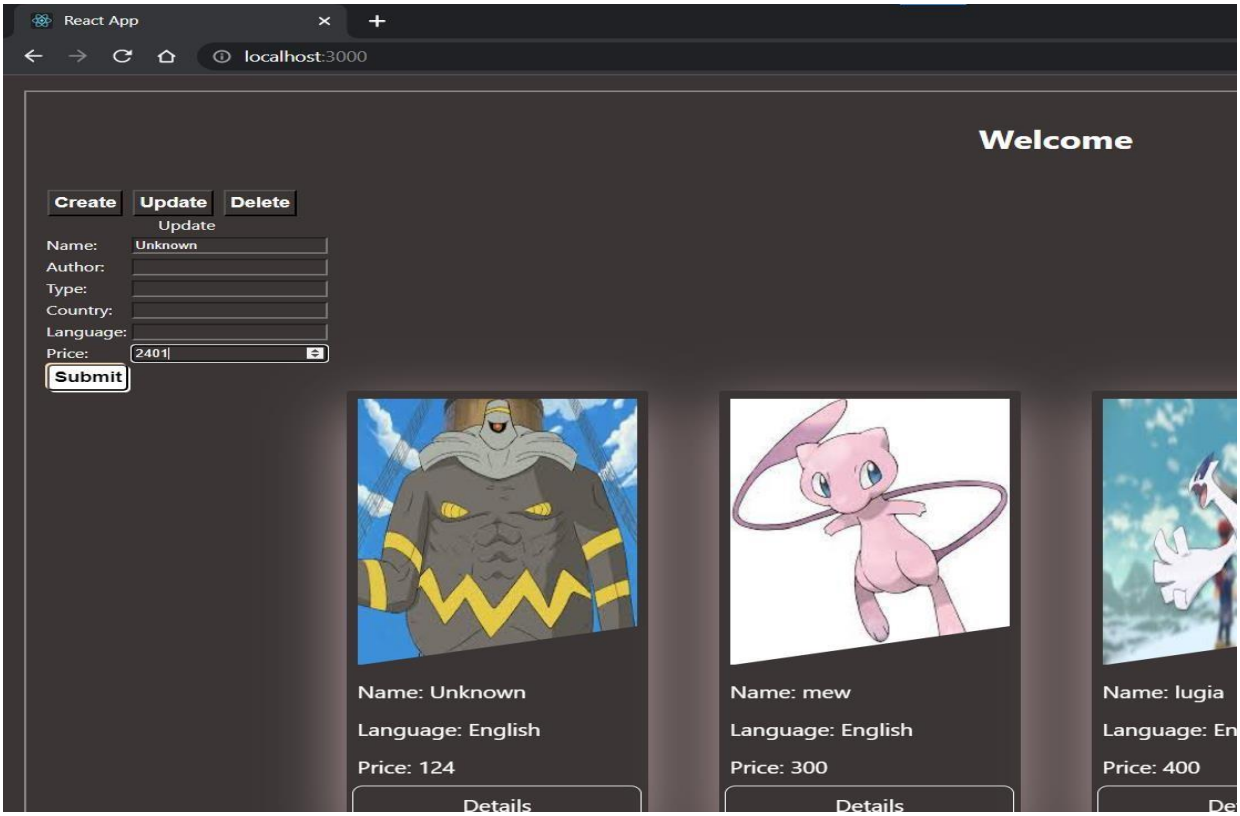
Creating a new comic:

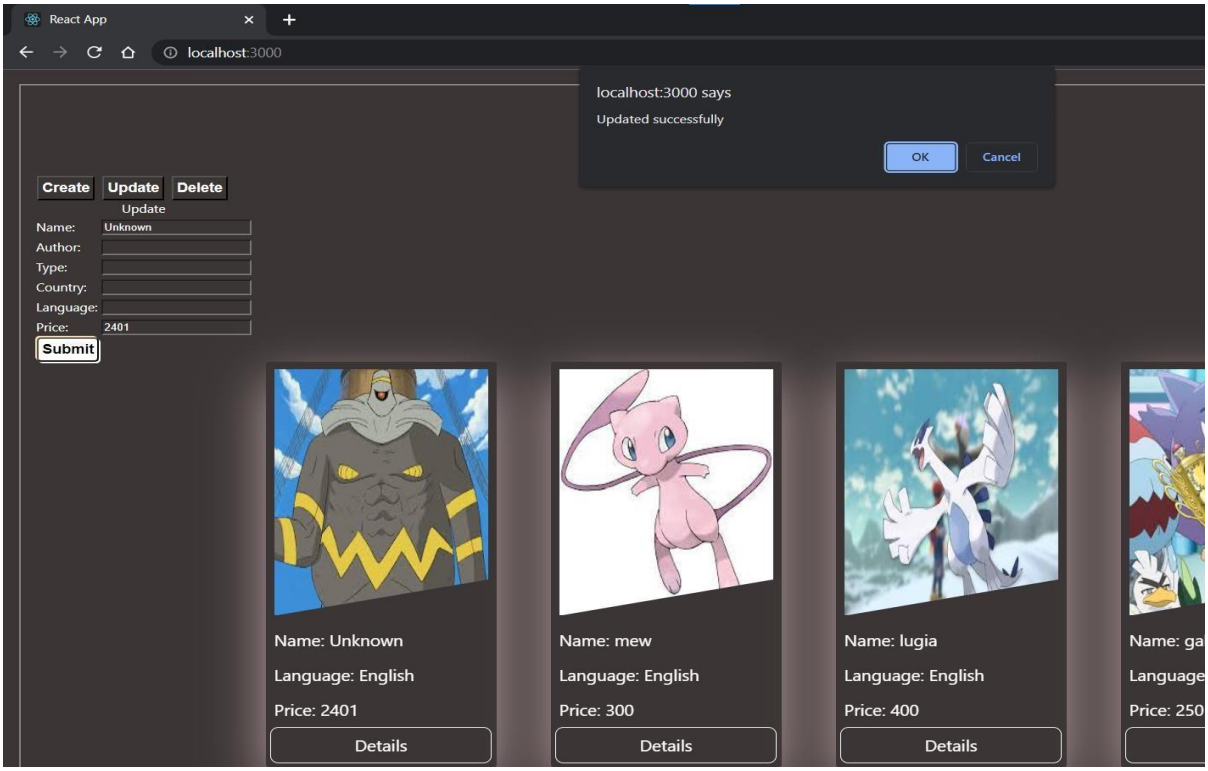


New Comic with given name is created and shown in web page


React App
localhost:3000
Welcome
Create
Update
Delete
Create
Name: Unknown
Author: Gary
Type: Mysterious
Country: Pallet
Language: English
Price: 0124
Submit
Name: Unknown
Language: English
Price: 124
Details
Name: mew
Language: English
Price: 300
Details
Name: lugia
Language: English
Price: 400
Details

Updating the newly created comic price from 0124 to 2401





Deleting the newly created comic with name Unknown