EGOTECHWORLD.COM

Beginner-friendly tutorials for real developers

React Pizza Manager

Build a Full CRUD App with Image Upload

{ PIZZA }

What you will build

A pizza menu app that adds, edits, deletes, and lists pizzas with image upload — using only React and useState.

BEGINNER LEVEL • CRUD • IMAGE UPLOAD • HOOKS

Step-by-step React tutorial — no prior framework knowledge needed

Welcome • Before You Begin

What this tutorial covers and what you need

REACT

What you will build

In this tutorial you will build **Pizza Manager** — a small but real React app that lets you add new pizzas, edit them, delete them, and upload an image for each pizza. By the end, you will understand the four building blocks every React developer uses every day: **components, state, events, and lists**.

Who this is for

This guide is written for absolute beginners. If you know basic HTML, CSS, and a tiny bit of JavaScript (variables, functions, arrays), you are ready. We will explain every line. No previous React experience is needed.

What you need installed

1	**Node.js 18 or newer**	nodejs.org — gives you npm
2	**A code editor**	VS Code is recommended
3	**A modern browser**	Chrome, Firefox, or Edge

How this tutorial is structured

Each part introduces one concept, shows the code, and explains what every line does. You can follow along by typing the code yourself — that is how you actually learn React.

PRO TIP

Do not copy and paste. Type the code yourself. Your fingers learn the patterns and the syntax becomes natural after just a few hours of practice.

Part 1 • React in 2 Minutes

REACT

The minimum you must understand before coding

What is a React component?

A **component** is just a JavaScript function that returns HTML-like code (called JSX). When you call the function, React shows the HTML on the screen. That is the whole idea.

The simplest component possible

JSX

```jsx
import React from "react";

function Greeting() {
  return <h1>Hello Pizza World</h1>;
}

export default Greeting;
```

Line-by-line explanation

Line 1 Imports React. Required so JSX (the HTML-like code) can be understood.

Line 3 Defines a function called Greeting. Component names must start with a capital letter.

Line 4 Returns JSX. Looks like HTML but it is actually JavaScript.

Line 7 Exports the component so other files can use it.

REMEMBER

Every React app is a tree of components. App contains PizzaForm, PizzaList, and PizzaCard. Each component is one function in one file.

What is state?

State is data that can change. When state changes, React automatically updates the screen. We use **useState** — a built-in React hook — to add state to a component.

```jsx
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>
        Add one
      </button>
    </div>
  );
}
```

useState(0) creates a state variable starting at 0. It returns two things: **count** (the current value) and **setCount** (a function to change it). When you click the button, setCount runs and React redraws the screen with the new value.

Part 2 • Project Setup

REACT

Create the React project and run it

1 Create the project with Vite

We will use **Vite** — the fastest way to start a React project. Open your terminal (in VS Code: View > Terminal) and run these commands one by one:

BASH

```
npm create vite@latest pizza-manager -- --template react
cd pizza-manager
npm install
npm run dev
```

After the last command, open **http://localhost:5173** in your browser. You should see the default Vite + React welcome page. If you do — congratulations, your environment works.

2 Understand the folder structure

File / Folder	What it does
`src/main.jsx`	Entry point — starts your app
`src/App.jsx`	Main component — we will edit this most
`src/index.css`	Global styles for your app
`public/`	Static files like logos and images
`package.json`	Lists installed packages and scripts

3 Clean up App.jsx

Open **src/App.jsx** and replace everything inside with this minimal starting point. Then save the file. The browser will reload automatically.

JSX

```jsx
import React from "react";

function App() {
  return (
    <div style={{ padding: 20 }}>
      <h1>Pizza Manager</h1>
      <p>Coming soon...</p>
    </div>
  );
}

export default App;
```

SAVE AND WATCH

Vite has hot reload — every time you save, the browser updates instantly without losing data. Keep your browser and editor side by side while you work.

Part 3 • Display Pizzas (Read)

REACT

Show a list of pizzas using state and map()

The plan

Before we let users add pizzas, we will start by showing a hard-coded list. This teaches you two important things: how to store data in state, and how to render a list with the **map()** function.

Step 1 — Add state for the pizza list

JSX

```jsx
import React, { useState } from "react";

function App() {
  const [pizzas, setPizzas] = useState([
    { id: 1, name: "Margherita", price: 850, image: "" },
    { id: 2, name: "Pepperoni", price: 1200, image: "" },
    { id: 3, name: "BBQ Chicken", price: 1450, image: "" },
  ]);

  return (
    <div style={{ padding: 20 }}>
      <h1>Pizza Manager</h1>
    </div>
  );
}

export default App;
```

We start **pizzas** as an array of three objects. Each pizza has an **id** (unique number), **name**, **price**, and **image** (empty for now). **setPizzas** is the function we will use later to add, edit, or delete items.

Step 2 — Render the list with map()

map() takes each item in an array and returns a new piece of JSX for it. This is how React renders lists. Replace the return block with this:

```jsx
return (
  <div style={{ padding: 20, fontFamily: "Arial" }}>
    <h1>Pizza Manager</h1>
    <div style={{ display: "grid", gap: 15 }}>
      {pizzas.map((pizza) => (
        <div key={pizza.id} style={{
          border: "1px solid #ddd",
          padding: 15,
          borderRadius: 8,
        }}>
          <h3>{pizza.name}</h3>
          <p>Rs. {pizza.price}</p>
        </div>
      ))}
    </div>
  </div>
);
```

THE KEY PROP

Every item in a mapped list needs a unique key prop. React uses it to track which items changed when state updates. Forgetting it works but causes warnings — always include key.

What you should see now

Save the file. Your browser should display three pizza cards with names and prices. If you see them, your **read** step is complete. Now let us add a form.

Part 4 • Add a Pizza (Create)

REACT

A form with name, price, and image upload

What we need

A form with three inputs (name, price, image), state to track each input, and a function that pushes a new pizza into the list when the user submits.

Step 1 — Add state for the form fields

JSX

```jsx
const [name, setName] = useState("");
const [price, setPrice] = useState("");
const [image, setImage] = useState("");
```

Each input needs its own state. **name** and **price** start as empty strings. **image** will hold a base64 data URL after the user picks a file.

Step 2 — Handle the image upload

When the user picks a file, we read it as a base64 string and save it in state. This lets us preview the image and store it without a backend.

JSX

```jsx
function handleImageUpload(event) {
  const file = event.target.files[0];
  if (!file) return;

  const reader = new FileReader();
  reader.onloadend = () => {
    setImage(reader.result);
  };
  reader.readAsDataURL(file);
}
```

`event.target.files[0]` Gets the first file the user picked.

`FileReader` A browser API that reads files in JavaScript.

`reader.onloadend` Runs when the file is fully read.

`reader.result` The base64 data URL — looks like '...'

`readAsDataURL` Tells FileReader to read it as a base64 URL we can use as <img src>.

Step 3 — The addPizza function

```jsx
function addPizza() {
  if (!name || !price) {
    alert("Please enter name and price");
    return;
  }

  const newPizza = {
    id: Date.now(),
    name: name,
    price: Number(price),
    image: image,
  };

  setPizzas([...pizzas, newPizza]);

  setName("");
  setPrice("");
  setImage("");
}
```

Date.now() gives a unique number based on the current time — perfect for an id. **[...pizzas, newPizza]** creates a new array with all old pizzas plus the new one — this is called the spread operator. Then we clear the form fields.

Part 4 • The Form JSX

REACT

Wire the inputs to state

Step 4 — Add the form above the list

JSX

```jsx
<div style={{ marginBottom: 20, padding: 15,
              border: "2px solid #d4a017", borderRadius: 8 }}>
  <h2>Add New Pizza</h2>

  <input
    type="text"
    placeholder="Pizza name"
    value={name}
    onChange={(e) => setName(e.target.value)}
  />

  <input
    type="number"
    placeholder="Price"
    value={price}
    onChange={(e) => setPrice(e.target.value)}
  />

  <input
    type="file"
    accept="image/*"
    onChange={handleImageUpload}
  />

  {image && (
    <img src={image} alt="preview"
         style={{ width: 100, marginTop: 10 }} />
  )}

  <button onClick={addPizza}>Add Pizza</button>
</div>
```

Two-way binding explained

Notice each input has both **value** and **onChange**. This is called a **controlled component**. The value comes from state, and onChange updates state. This pattern is the React way of handling forms — memorize it.

THE IMAGE PREVIEW TRICK

{image && (<img />)} means: if image is not empty, show the img tag. This is conditional rendering — one of the most common React patterns.

Update the pizza card to show the image

JSX

```jsx
<div key={pizza.id} style={{
  border: "1px solid #ddd",
  padding: 15,
  borderRadius: 8,
}}>
  {pizza.image && (
    <img src={pizza.image} alt={pizza.name}
         style={{ width: 150, height: 150,
                  objectFit: "cover", borderRadius: 6 }} />
  )}
  <h3>{pizza.name}</h3>
  <p>Rs. {pizza.price}</p>
</div>
```

Part 5 • Delete a Pizza

REACT

Remove an item by id using filter()

How delete works in React

We never modify the original array directly. Instead, we use **filter()** to create a new array *without* the item we want to remove, and pass it to setPizzas. React sees the new array and removes that card from the screen.

Step 1 — The deletePizza function

JSX

```jsx
function deletePizza(id) {
  const confirmed = window.confirm("Delete this pizza?");
  if (!confirmed) return;

  setPizzas(pizzas.filter((pizza) => pizza.id !== id));
}
```

filter() keeps every item where the test returns true. Here we keep every pizza whose id is *not* equal to the one we want to delete. Simple and clean.

Step 2 — Add a delete button to each card

JSX

```jsx
<button
  onClick={() => deletePizza(pizza.id)}
  style={{
    background: "#c62828",
    color: "white",
    border: "none",
    padding: "6px 12px",
    borderRadius: 4,
    cursor: "pointer",
  }}
>
  Delete
</button>
```

WHY THE ARROW FUNCTION?

We write onClick={() => deletePizza(pizza.id)} instead of onClick={deletePizza(pizza.id)}. The second one would call deletePizza immediately when the page loads. The arrow function delays it until the user actually clicks.

Test it

Save the file, click Delete on any pizza, confirm in the popup, and watch it disappear smoothly. That is React reactivity in action — change the state, the UI updates itself.

Part 6 • Edit a Pizza

REACT

Toggle edit mode and save changes

The edit pattern

Editing is the trickiest of the four operations. We need to remember which pizza is being edited and load its values into the form. We will use a state variable called **editingId** that holds the id of the pizza being edited, or null when adding a new one.

Step 1 — Add editingId state

JSX

```jsx
const [editingId, setEditingId] = useState(null);
```

Step 2 — startEdit function

When the user clicks Edit on a card, copy that pizza's values into the form fields and remember the id we are editing.

JSX

```jsx
function startEdit(pizza) {
  setEditingId(pizza.id);
  setName(pizza.name);
  setPrice(pizza.price);
  setImage(pizza.image);
}
```

Step 3 — Update savePizza to handle both modes

Replace the old **addPizza** function with this smarter **savePizza**. If editingId is set, update an existing pizza. Otherwise, add a new one.

JSX

```jsx
function savePizza() {
  if (!name || !price) {
    alert("Please enter name and price");
    return;
  }

  if (editingId) {
    // Update existing pizza
    setPizzas(pizzas.map((pizza) =>
      pizza.id === editingId
        ? { ...pizza, name, price: Number(price), image }
        : pizza
    ));
  } else {
    // Add new pizza
    setPizzas([...pizzas, {
      id: Date.now(),
      name,
      price: Number(price),
      image,
    }]);
  }

  // Reset form
  setName("");
  setPrice("");
  setImage("");
  setEditingId(null);
}
```

TERNARY IN MAP

pizza.id === editingId ? newObject : pizza means: if this is the pizza we are editing, return the updated version; otherwise, return it unchanged. This is how you update one item in an array while keeping the others.

Part 6 • Edit Mode UI

REACT

Show edit/cancel buttons and dynamic title

Step 4 — Add Edit button to each card

JSX

```jsx
<button
  onClick={() => startEdit(pizza)}
  style={{
    background: "#1e88e5",
    color: "white",
    border: "none",
    padding: "6px 12px",
    borderRadius: 4,
    cursor: "pointer",
    marginRight: 8,
  }}
>
  Edit
</button>
```

Step 5 — Make the form title and button dynamic

The form should say **Edit Pizza** or **Add New Pizza** depending on the mode. Replace the form heading and the submit button:

JSX

```jsx
<h2>{editingId ? "Edit Pizza" : "Add New Pizza"}</h2>

// ... inputs go here ...

<button onClick={savePizza}>
  {editingId ? "Update Pizza" : "Add Pizza"}
</button>

{editingId && (
  <button onClick={() => {
    setEditingId(null);
    setName("");
    setPrice("");
    setImage("");
  }}>
    Cancel
  </button>
)}
```

Test the full edit flow

Click Edit on any pizza. The form should fill with that pizza's data and the heading should change to **Edit Pizza**. Change the name or price, click **Update Pizza**, and watch that one card update — the rest stay the same.

YOU DID IT

If your edit works, you have just built a real CRUD app. Add. Read. Update. Delete. These four operations power every business application in the world.

Part 7 • Complete App.jsx

REACT

All pieces combined into one file

Here is the entire **App.jsx** file with everything we built. Use this to double-check your code or to start over if anything broke along the way.

JSX

```jsx
import React, { useState } from "react";

function App() {
  const [pizzas, setPizzas] = useState([
    { id: 1, name: "Margherita", price: 850, image: "" },
    { id: 2, name: "Pepperoni", price: 1200, image: "" },
  ]);

  const [name, setName] = useState("");
  const [price, setPrice] = useState("");
  const [image, setImage] = useState("");
  const [editingId, setEditingId] = useState(null);

  function handleImageUpload(e) {
    const file = e.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onloadend = () => setImage(reader.result);
    reader.readAsDataURL(file);
  }

  function savePizza() {
    if (!name || !price) return alert("Fill all fields");
    if (editingId) {
      setPizzas(pizzas.map((p) =>
        p.id === editingId
          ? { ...p, name, price: Number(price), image }
          : p
      ));
    } else {
      setPizzas([...pizzas, {
        id: Date.now(), name,
        price: Number(price), image,
      }]);
    }
    setName(""); setPrice(""); setImage("");
    setEditingId(null);
  }

  function startEdit(p) {
    setEditingId(p.id);
    setName(p.name);
    setPrice(p.price);
    setImage(p.image);
  }

  function deletePizza(id) {
    if (window.confirm("Delete?"))
      setPizzas(pizzas.filter((p) => p.id !== id));
  }
```

Part 7 • Complete App.jsx (continued)

REACT

The JSX render block

JSX

```jsx
  return (
    <div style={{ padding: 20, fontFamily: "Arial",
                  maxWidth: 800, margin: "auto" }}>
      <h1>Pizza Manager</h1>

      <div style={{ padding: 15, marginBottom: 20,
                    border: "2px solid #d4a017",
                    borderRadius: 8 }}>
        <h2>{editingId ? "Edit Pizza" : "Add New Pizza"}</h2>
        <input type="text" placeholder="Name"
               value={name}
               onChange={(e) => setName(e.target.value)} />
        <input type="number" placeholder="Price"
               value={price}
               onChange={(e) => setPrice(e.target.value)} />
        <input type="file" accept="image/*"
               onChange={handleImageUpload} />
        {image && <img src={image} alt=""
                       style={{ width: 100, display: "block" }} />}
        <button onClick={savePizza}>
          {editingId ? "Update" : "Add"}
        </button>
      </div>

      <div style={{ display: "grid", gap: 15 }}>
        {pizzas.map((pizza) => (
          <div key={pizza.id} style={{
            border: "1px solid #ddd", padding: 15,
            borderRadius: 8, display: "flex", gap: 15,
          }}>
            {pizza.image && (
              <img src={pizza.image} alt={pizza.name}
                   style={{ width: 100, height: 100,
                            objectFit: "cover" }} />
            )}
            <div style={{ flex: 1 }}>
              <h3>{pizza.name}</h3>
              <p>Rs. {pizza.price}</p>
              <button onClick={() => startEdit(pizza)}>
                Edit
              </button>
              <button onClick={() => deletePizza(pizza.id)}>
                Delete
              </button>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
}

export default App;
```

Part 8 • Recap and Next Steps

REACT

What you learned and where to go next

What you have learned

[OK] Components Functions that return JSX and become reusable UI pieces.

[OK] State with useState How to store data that changes and trigger re-renders.

[OK] Events onClick and onChange — how React handles user interaction.

[OK] Controlled inputs Two-way binding with value and onChange.

[OK] Lists with map How to render arrays as JSX, and why keys matter.

[OK] Conditional rendering Using && and ternary operators to show or hide UI.

[OK] Spread operator How to add to or update an array immutably.

[OK] filter and map Two essential array methods for delete and update.

[OK] FileReader API How to read uploaded files as base64 in the browser.

Practice challenges

Try these to lock in what you learned:

1	**Add a category field**	Pizza type: Veg, Non-Veg, Vegan
2	**Add a search box**	Filter pizzas by name as you type
3	**Add total price**	Show the sum of all pizza prices below the list
4	**Save to localStorage**	Make the data persist across page reloads
5	**Sort by price**	Add buttons to sort low-to-high and high-to-low
6	**Validate price**	Reject negative numbers and show an error message

Where to go next

You now know enough React to learn anything else in the ecosystem. The next topics worth studying are: **useEffect** for side effects, **component splitting** for cleaner code, **props** for passing data between components, **React Router** for multi-page apps, and connecting to a real backend with **fetch** or **axios**. Visit **egotechworld.com** for the next tutorial in this series.

FINAL WORD FROM THE AUTHOR

The hardest part of learning React is starting. You just finished a complete CRUD app. Build five more small apps just like this one and React will feel as natural as HTML.