React

CREATE APPLICATION

npx create-react-app my-react-prepare

cd my-react-prepare

npm start

Open http://localhost:3000 to view it in the browser.

ADDING i18n APPLICATION

npm install i18next react-i18next i18next-browser-languagedetector --legacy-peer-deps

npm i i18next-http-backend --legacy-peer-deps

ADDING react routing APPLICATION

npm install react-router-dom --legacy-peer-deps

Add new translation JSON files per page

We'll keep your existing common.json and myTranslation.json,

and add home.json and about.json for each language.

Update src/i18n.js to include page namespaces,

Add "home" and "about" to your ns and keep common as defaultNS.

src/App.js – combine Router + i18next + pages

Lazy loading namespaces per route (e.g., only load about namespace when visiting /about)

ADDING react draggable

add draggable

npm install react-beautiful-dnd --legacy-peer-deps

formik and yup

npm install formik --legacy-peer-deps

npm install yup --legacy-peer-deps

Programming react questions

-- code snippet

https://dev.to/allenarduino/live-coding-react-interview-questions-2ndh

input

<input value={text} onChange={(e) => setText(e.target.value)} placeholder="Add a to-do" />

button toggle

<button onClick={() => setIsOn(!isOn)}>

useState for query

const [query, setQuery] = useState('');

delete item

.splice(index,1) -- is for delete

consider only few records

.slice(0, 5)

add item

setTodos([...todos, { text, completed: false }])

update item

const newTodos = [...todos]; newTodos[index].completed = !newTodos[index].completed; setTodos(newTodos);

useEffect/ fetch data from API

useEffect(() => { fetch('https://api.example.com/data') .then(response => response.json()) .then(data => { setData(data); setLoading(false); }); }, []);

filter items

items.filter(item => item.toLowerCase().includes(query.toLowerCase()));

iterative over items

return (

- {data.map(item => ({item.name}

))});
iterative over items - 1 return (

- {data.map(item => ({item.name}

))});

iterative over items - 2

return (

<button onClick={() => setIsOpen(!isOpen)}>Menu {isOpen && (
- {items.map((item, index) => ({item}

))}})}
);

iterative over items - 3

return (

{tabs.map((tab, index) => (<button key={index} className={index === activeTab ? 'active' : ''} onClick={() => setActiveTab(index)} > {tab.label}))}
{tabs[activeTab].content}
);

iterative over items - 4 Tabs

const Tabs = ({ tabs }) => { const [activeTab, setActiveTab] = useState(0);

return (

{tabs.map((tab, index) => (<button key={index} className={index === activeTab ? 'active' : ''} onClick={() => setActiveTab(index)} > {tab.label}))}
{tabs[activeTab].content}
); };
const App = () => { const tabs = [{ label: 'Tab 1', content:

Content of Tab 1
}, { label: 'Tab 2', content:
Content of Tab 2
}, { label: 'Tab 3', content:
Content of Tab 3
},];
return ; };

Multstep form

const Step1 = ({ next }) => (

Step 1

Next
);
const Step2 = ({ next, previous }) => (

Step 2

Previous Next
);
const Step3 = ({ previous }) => (

Step 3

Previous Submit
);
const MultiStepForm = () => { const [step, setStep] = useState(1);

const nextStep = () => setStep(step + 1); const previousStep = () => setStep(step - 1);

const handleSubmit = (e) => { e.preventDefault(); console.log('Form submitted'); };

return (

{step === 1 && } {step === 2 && } {step === 3 && }); };
const App = () => { return (

Multi-Step Form

); };

socket.io-client

npm install socket.io-client --legacy-peer-deps

language switcher

i18n.changeLanguage(lng); const { t } = useTranslation(['myTranslation']); return (

{t("hello")}

{t("welcome")}

);

Summary Table

Step Command / Action Create App npx create-react-app my-i18n-app Install i18n npm install i18next react-i18next ... Add code & JSON files src/i18n.js, translation files Run app npm start or npm run dev View in browser localhost:3000 or localhost:5173

map, filter, reduce

Convert number to binary

const arr = [5, 1, 3, 2, 6]; function binary(x) { return x.toString(2); // convert number to binary } const output = arr.map(binary);

Filter > 4 → using filter

const arr = [5, 1, 3, 2, 6]; const greaterThan4 = arr.filter(x => x > 4);

reduce : find max value in array arr.

const max = arr.reduce(function (acc, curr) { if (curr > acc) { acc = curr; } return acc; }, 0); // initial value — can also use `-Infinity`

console.log(max); // ✅ Output: 6

You Now Know:

✔ .map() → Transform array ✔ .filter() → Filter values ✔ .reduce() → Sum / Max / Any single result ✔ DRY Principle + Higher Order Functions ✔

filter + map OR reduce + map

const users = [{ firstName: "akshay", lastName: "saini", age: 26 }, { firstName: "donald", lastName: "trump", age: 75 }, { firstName: "elon", lastName: "musk", age: 50 }, { firstName: "deepika", lastName: "padukone", age: 26 }];

// ["akshay", "deepika"] const output = users .filter((x) => x.age < 30) .map((x) => x.firstName);

console.log(output);

---

const output = users.reduce((acc, user) => { if (user.age < 30) { acc.push(user.firstName); } return acc; }, []);

console.log(output);

---

npm install styled-components --legacy-peer-dep

Hooks must be called in the same order on every render.

React hooks must be called at the top level because React relies on the order of hook calls to correctly associate state with components. Calling hooks conditionally or in loops breaks this order.

Summary Table ❌ Not Allowed ✅ Allowed Inside if At top level Inside loops At top level Inside functions Inside component Inside callbacks Inside component Conditional hook calls Conditional logic inside hook

---

🟦 QUICK SUMMARY TABLE (Interview Ready)

Hook: Purpose Returns Used For

useState: local state [value, setValue] UI changes

useEffect: side effects nothing API calls, listeners

useContext: global state context value avoid props drilling

useMemo: memoize values cached value heavy calculations

useCallback: memoize functions cached function stop re-renders

useReducer: advanced state [state, dispatch] complex logic

useRef: mutable ref ref object DOM access, storing values

useLayoutEffect: sync layout effect nothing before painting UI

useImperativeHandle: expose custom ref API nothing child → parent actions

React.memo: memoize component component optimization

- useLayoutEffect runs synchronously after DOM mutations but before the browser paints, making it ideal for reading or synchronizing layout to avoid visual inconsistencies.
- useMemo is used to memoize the result of expensive calculations so they don’t re-run on every render, improving performance.
- useCallback is used to memoize functions so they don’t get recreated on every render,

useMemo vs useCallback (Quick Interview Tip)

useMemo useCallback

Memoizes value Memoizes function

Returns result Returns function

Heavy calculations Passing callbacks

🟦 High-Level Difference (One Line Each)

useMemo → Memoizes a value

useCallback → Memoizes a function

React.memo → Memoizes a component render

🧠 Comparison Table (Interview-Ready)

Feature useMemo useCallback React.memo

Memoizes: Value Function Component

Used in: Component Component Component definition

Prevents: Recalculation New function ref Re-render

Returns: Value Function Component

Works alone?: Yes Usually with memo Needs stable props

Dependency array: Yes Yes Optional compare fn

---

useMemo memoizes values,

useCallback memoizes functions, and

React.memo memoizes component renders to prevent unnecessary recalculations and re-renders.

---

npm install -g json-server json-server --watch db.json --port 3001

---

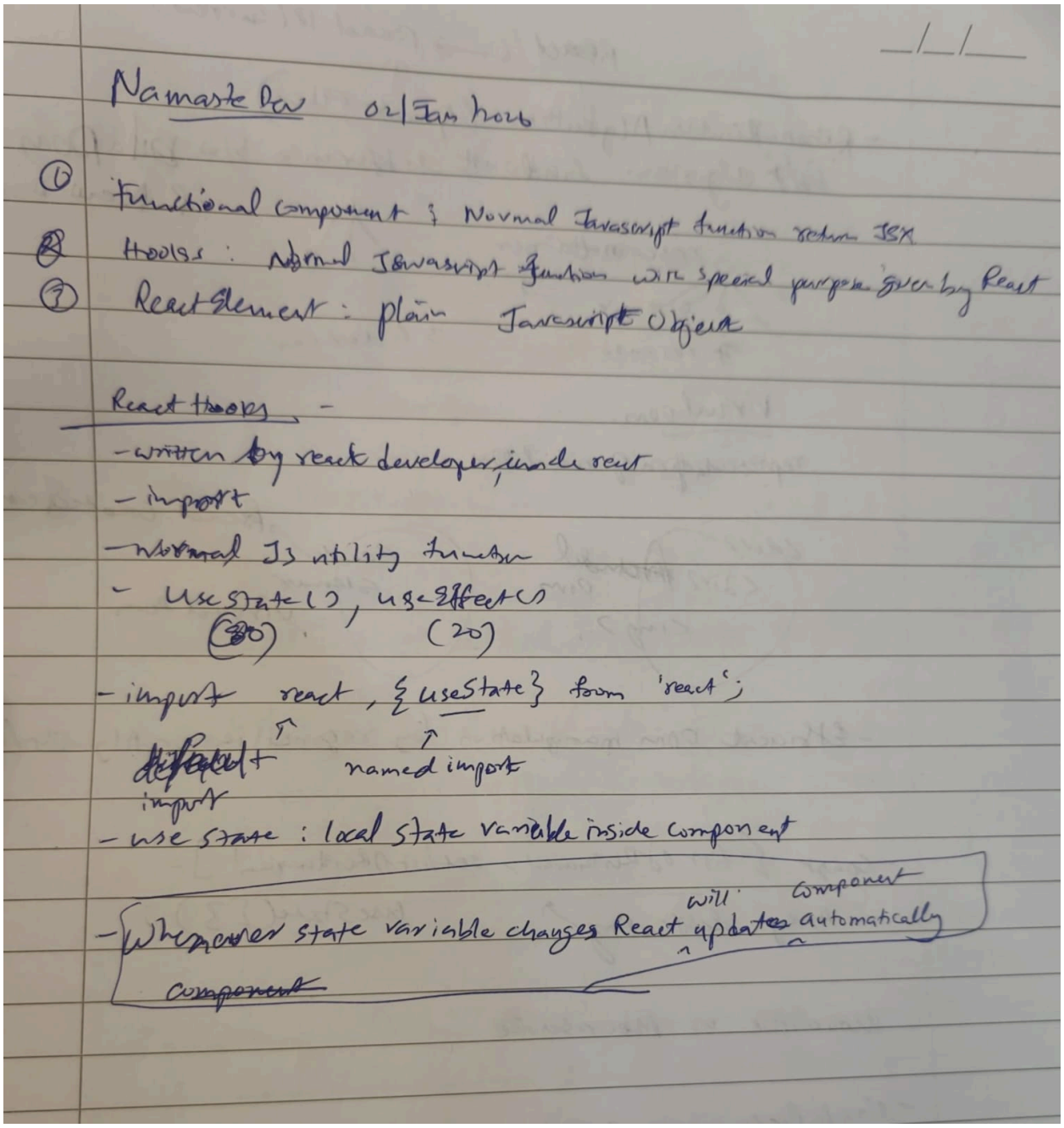
npm install @tanstack/react-query --legacy-peer-dep npm install @tanstack/react-query-devtools --legacy-peer-dep

React Query simplifies server-state management by handling caching, retries, background refetching, and error states automatically.

Interview-Ready Comparison

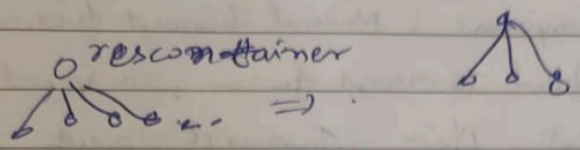
Feature	useEffect	React Query
Caching	❌	✅
Retry	❌	✅
Refetch	Manual	Built-in
Loading	Manual	Built-in
Error	Manual	Built-in
Background sync	❌	✅

Interview questions preparation

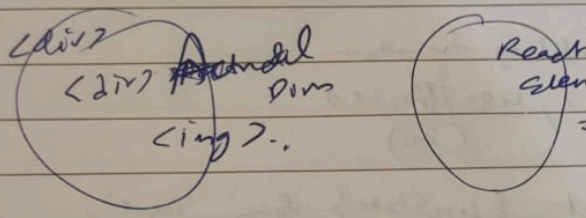


React 16 → React 18 (current) —/—/—

- Reconciliation Algorithm (React Fiber)
  Diff algorithm: find out difference b/w Old DOM & New DOM



Virtual DOM.
representation of actual DOM.



- Efficient DOM manipulation by reconciliation Algorithm (React)
-

- const [listOfRestaurant , setListOfRestaurant] = useState([3])
  Array destructuring

- Monolithic vs Microservice

- Single Responsibility principle, separation of concern

__/__/____

2 Approach

Loads → API → Render

500ms

Blank Page & Sudden Render

await

Loads → Render UI → API → Re Render with data from API

React

Better UX

React render Last.

useEffect: call this callback function after render of component

useEffect(() => { }, []);

const fetchData = async () => {

const data = await fetch("")

capability by browser

CORS policy

Promise .then(response =>

const json = await data.json();

c.l (json);

Allow CORS – Access control - Allow-Origin

↑ browser extension

__/__/__

5 Rounds

1. machine coding
2. DS / Algo.
3. Web Technology
4. UI / System Design
5. Hiring Manager

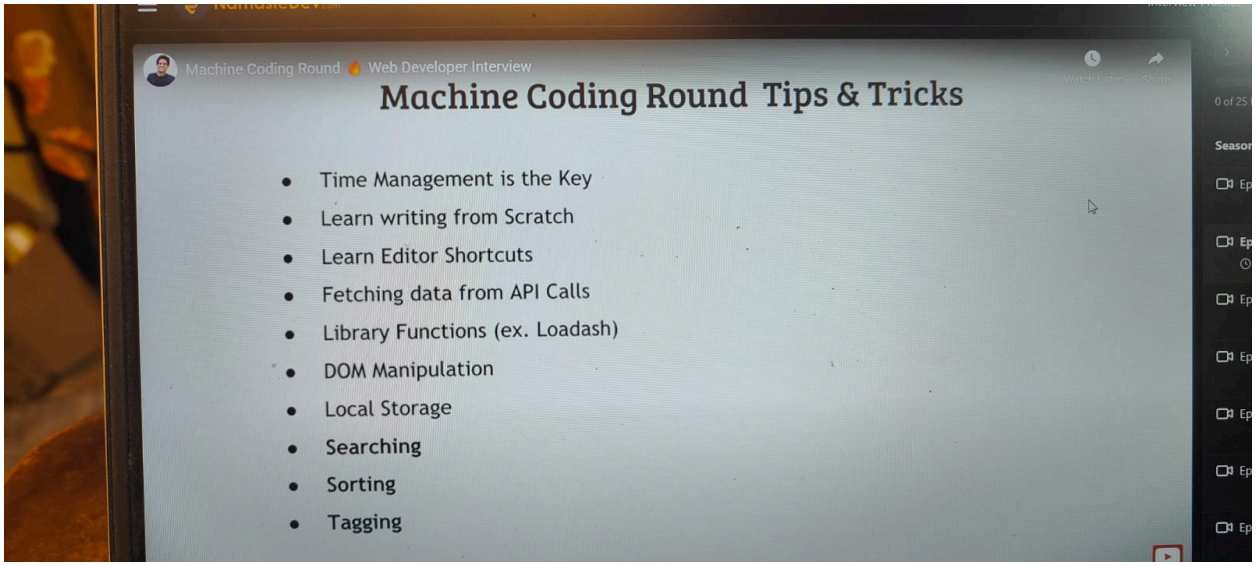
1. Tech discussion
2. Question discussion
3. Web Developer friends

1) Machine coding

problem statement
Time frame
Vanilla Javascript

⇒ Javascript Vanilla
Mental model
Tricky part of the problem is in
Revisit the progress, skip feature.
Realign yourself

⇒ Time management: 1-2 hour. 1) learn
5 minutes are crucial planning 2) fetch data from API call
list all things down
What to use library
lodash, underscore JS
planning well is the key
Draw the user interface/Mental method
Requirement understanding
time



Thinking Recursively | Microsoft Interview Question | Software Engineer UI/Frontend
https://www.youtube.com/watch?v=Vi4Pr8bUMZs