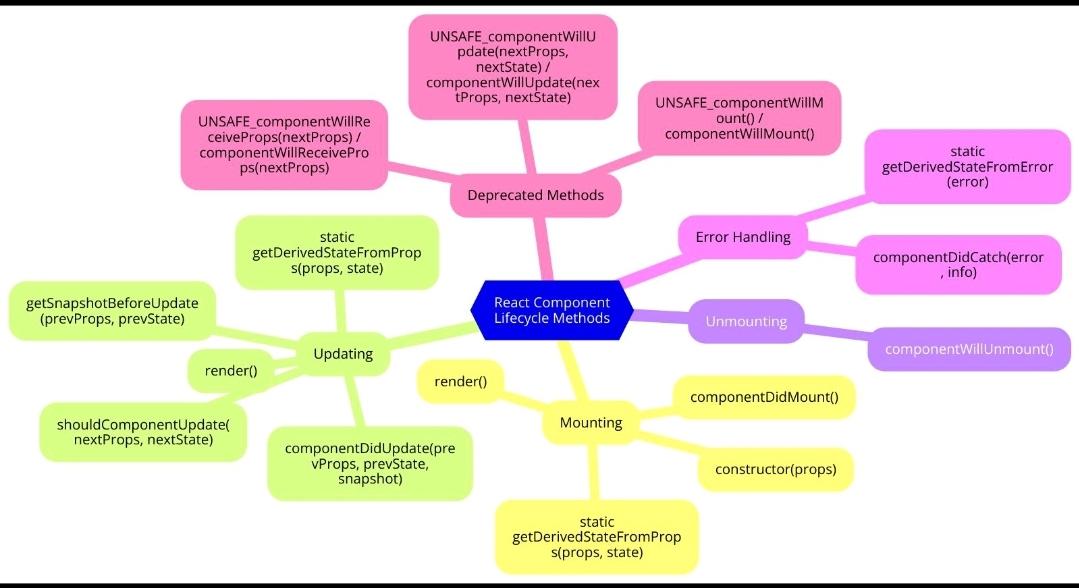
React Interview



# Links :

Road map : [React Developer Roadmap: Learn to become a React developer](https://roadmap.sh/react)

[Learn ReactJS – Complete Roadmap (freecodecamp.org)](https://www.freecodecamp.org/news/react-fundamentals-for-beginners/)

<https://www.linkedin.com/posts/manikmaity_ultimate-reactjs-roadmap-ugcPost-7192045223311499265-WtPP?utm_source=share&utm_medium=member_android>

Concepts: <https://www.linkedin.com/posts/gauravcode_react-reactjs-reactjs-activity-7184540079225262081-PyUU?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/omar-elshiref-programmer_react-router-in-details-ugcPost-7189623002982993920-azxV?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/codepapa360_react-server-components-activity-7191770908011716608-j2dE?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/feed/update/urn:li:activity:7184574012092489728?utm_source=share&utm_medium=member_android>

Road map from linked in post : <https://www.linkedin.com/posts/rehomanalif_react-js-roadmap-ugcPost-7170686723104890880-bH4C?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/codepapa360_react-roadmap-activity-7183844932242038784-2E3D?utm_source=share&utm_medium=member_android>

Full stack development: <https://www.linkedin.com/posts/abhushit-chaudhary-88266a167_fullstack-developer-roadmap-activity-7171418909995155457-wXel?utm_source=share&utm_medium=member_android>

interview:

<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7191820317881729024-7VWW?utm_source=share&utm_medium=member_desktop>

<https://www.linkedin.com/posts/amankumar1in_react-js-questions-ugcPost-7171344245209903104-pgoO?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/sai-ram-somanaboina_i-spent-the-last-week-collecting-these-50-activity-7186340369083596800-Lp_i?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/rajatgajbhiye_softwaredevelopment-systemdesign-js-activity-7193210393517813760-79rj?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/harshit-chopra7_frontend-react-seo-activity-7177881001069559808-_AW3?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/mohitsehrawat_react-questions-to-crack-interviews-ugcPost-7177516117635002368-zZ3Q?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/aliyan-ansari_interview-questions-ugcPost-7178271289181171712-boFf?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7186763350612099072-Kbt5?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7187859450911682560-e2H5?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/parthib-mohanta_react-top-50-questions-ugcPost-7187159185493524480-bHHU?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/rajatgajbhiye_most-asked-javascript-interview-questions-ugcPost-7187658889176244226-qKC8?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/codepapa360_javascript-interview-questions-with-answers-activity-7189203993657552897-GGSQ?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/yogesh-chavan97_javascript-reactjs-nodejs-activity-7190197052360847360-Y2Nu?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/codepapa360_react-interview-questions-ugcPost-7191479490013765632-e8OI?utm_source=share&utm_medium=member_android>

React learning topics : <https://www.linkedin.com/posts/rehomanalif_react-js-in-20-days-ugcPost-7173963616642162689-HRCd?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/saikrishnanangunuri_javascript-javascriptdeveloper-reactjs-activity-7183349478836375552-l_Bb?utm_source=share&utm_medium=member_android>

super dopper plan for full stack : <https://www.linkedin.com/posts/saurav-singh-08_full-stack-development-roadmap-activity-7180054954190004224-ir9B?utm_source=share&utm_medium=member_android>

Libraries :

For animation - [Homepage | GSAP](https://gsap.com/) , three. js, react three fiber

For form - [How to Create Forms in React using react-hook-form (freecodecamp.org)](https://www.freecodecamp.org/news/how-to-create-forms-in-react-using-react-hook-form/)

<https://www.linkedin.com/feed/update/urn:li:activity:7194068564473962498?utm_source=share&utm_medium=member_android>

Css shape & gradient website: <https://www.linkedin.com/posts/shubhamkamble89_killer-tools-for-frontend-developers-ugcPost-7181577463511072769-AYsO?utm_source=share&utm_medium=member_android>

practice interview website:<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-angular-javascript-activity-7186377406155296769-tCGs?utm_source=share&utm_medium=member_android>

Redux : [Redux Tutorial - Learn Redux from Scratch (youtube.com)](https://www.youtube.com/watch?v=poQXNp9ItL4&t=1637s)

Redux toolkit : [ReactJS Course [14] - Redux Toolkit Tutorial (youtube.com)](https://www.youtube.com/watch?v=yZqBVLcWSn8)

Library: <https://www.linkedin.com/posts/codepapa360_top-react-libraries-tools-you-need-ugcPost-7190090991733727232-uIeO?utm_source=share&utm_medium=member_android>

# Learning YouTube:

1. <https://www.linkedin.com/posts/sahil-chopra-56a63b191_javascript-activity-7203359275459481602-mXpd?utm_source=share&utm_medium=member_android>

# ChatGBT guidline:

To become proficient in React.js, you should focus on mastering the following concepts:

1. \*\*JSX\*\*: Understand how to write JSX, which is a syntax extension for JavaScript used with React to describe what the UI should look like.

2. \*\*Components\*\*: Learn how to create and manage components, which are reusable pieces of UI.

3. \*\*State and Props\*\*: Understand the concepts of state and props, which are used to manage data and communication between components.

4. \*\*Lifecycle Methods\*\*: Familiarize yourself with component lifecycle methods, which allow you to perform actions at different points in a component's lifecycle.

5. \*\*React Router\*\*: Learn how to use React Router for handling navigation in single-page applications.

6. \*\*Hooks\*\*: Master the usage of hooks like useState, useEffect, useContext, etc., introduced in React 16.8 for managing state and side effects.

<https://www.linkedin.com/posts/muhammad-mohsin-029649178_controlled-vs-uncontrolled-components-react-ugcPost-7199304102650998784-VYrJ?utm_source=share&utm_medium=member_android>

7. \*\*Redux (or other state management libraries)\*\*: Understand how to use Redux or similar libraries for managing application state in larger applications.

8. \*\*Forms\*\*: Learn how to work with forms in React, including controlled and uncontrolled components.

9. \*\*Ajax and Fetch\*\*: Understand how to make AJAX requests using fetch or other libraries like Axios for data fetching and updating.

10. \*\*Error Handling\*\*: Learn about error boundaries and how to handle errors gracefully in React applications.

11. \*\*Testing\*\*: Familiarize yourself with testing frameworks like Jest and Enzyme for testing React components.

12. \*\*Performance Optimization\*\*: Learn about performance optimization techniques such as code splitting, memoization, and virtualization to improve the speed and efficiency of React applications.

Mastering these concepts will give you a solid foundation to become a proficient React.js developer.

# Best interview answers:

1. <https://youtu.be/jCTcvbgoKCo?si=5WfTqMVnnD_3WR6D>
2. <https://youtu.be/Etpdn4N9-d4?si=Gv0iChtU0WsrW00H>
3. <https://youtu.be/xRD0pkTJ-Zg?si=p1r7hvqVWfKbPTRy>
4. <https://youtu.be/ibV9Z7VOjbE?si=BttgnIEErds0kDIQ>
5. <https://youtu.be/fKWwlXzwA54?si=SceISYb773Epaq9d>
6. <https://youtu.be/6qERg1Yt1QQ?si=Ylz5gVmc-KUH6jg7> - Turning

# Questions :

## <https://www.linkedin.com/posts/yogeshkumar8434_react-js-roadmap-ugcPost-7217548016440766464-BaJt?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/rajatgajbhiye_react-roadmap-activity-7206150682460983296-ztns?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/bhoomikasahu_reactjs-frontend-web-activity-7216282470571212800-ysJX?utm_source=share&utm_medium=member_android> - 𝗥𝗲𝗰𝗼𝗻𝗰𝗶𝗹𝗶𝗮𝘁𝗶𝗼𝗻

## interview website:

<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-angular-javascript-activity-7192882103804387329-HVbz?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/codepapa360_top-30-react-interview-questions-ugcPost-7193318512205967361-V7bo?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/rajatgajbhiye_knowing-react-increases-your-value-in-the-activity-7193828213960151040-uoIH?utm_source=share&utm_medium=member_android>

1. why we choose react over other framework: [Bhoomika Sahu 🚀 on LinkedIn: #reactjs](https://www.linkedin.com/posts/bhoomikasahu_reactjs-activity-7211571374383849472-TIhG?utm_source=share&utm_medium=member_android)
2. redux recent interview: <https://www.linkedin.com/posts/ayushi1sharma_webdevelopment-reactjs-redux-activity-7214908384997777408-AW3R?utm_source=share&utm_medium=member_android>
3. [Hafijur Rahman on LinkedIn: #reactjs #webdevelopment #frontenddevelopment](https://www.linkedin.com/posts/hafijur-rahman0211_reactjs-webdevelopment-frontenddevelopment-activity-7213423709334716416-xUVi?utm_source=share&utm_medium=member_android) - concepts
4. <https://www.linkedin.com/posts/dimple-kumari_react-webdevelopment-interview-activity-7216797520591536128-1ry4?utm_source=share&utm_medium=member_android>
5. <https://www.linkedin.com/posts/parthib-mohanta_react-50-important-questions-ugcPost-7213774540101890048-XuA4?utm_source=share&utm_medium=member_android>
6. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7215728767011282946-xZCh?utm_source=share&utm_medium=member_android>
7. <https://www.linkedin.com/posts/yogesh-chavan97_javascript-reactjs-webdevelopment-activity-7213036283806310400-hyW7?utm_source=share&utm_medium=member_android>
8. <https://www.linkedin.com/posts/ranjan-kumar-635311296_top-notch-react-interview-questions-activity-7214907283179233282-vsno?utm_source=share&utm_medium=member_android>
9. props vs State <https://www.linkedin.com/posts/parthib-mohanta_state-vs-props-ugcPost-7212325108428722176-WUfL?utm_source=share&utm_medium=member_android>
10. <https://www.linkedin.com/posts/rasheed-dev_react-frontenddevelopment-softwareengineering-activity-7215769669985570816-eCyc?utm_source=share&utm_medium=member_android>
11. <https://www.linkedin.com/feed/update/urn:li:activity:7212363835398057985?utm_source=share&utm_medium=member_android>
12. <https://www.linkedin.com/posts/rani-dhage_js-coding-eventloop-activity-7211961500767309824-j1px?utm_source=share&utm_medium=member_android>
13. <https://www.linkedin.com/posts/sahil-chopra-56a63b191_frontend-react-seo-activity-7209400370589011970-Qw7T?utm_source=share&utm_medium=member_android>
14. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7213079756387307520-KUR4?utm_source=share&utm_medium=member_android>
15. <https://www.linkedin.com/posts/harisahmad59_frontend-interview-questions-activity-7211965504259833856-dO28?utm_source=share&utm_medium=member_android>
16. <https://www.linkedin.com/posts/srikanth-tekumudi-95b47216b_if-you-are-preparing-for-frontend-interviews-activity-7213036058232467456-rHd_?utm_source=share&utm_medium=member_android>
17. <https://www.linkedin.com/posts/harihara-kumar-b193761b4_react-javascript-webdevelopment-activity-7215024668674596864-NTXy?utm_source=share&utm_medium=member_android>
18. <https://www.linkedin.com/posts/prasanya-shankar_reactjs-interview-questions-and-answer-ugcPost-7213223524801396737-srIZ?utm_source=share&utm_medium=member_android>
19. <https://www.linkedin.com/posts/sai-ram-somanaboina_youre-not-clearing-your-front-end-interview-activity-7210620679841357824-FI0w?utm_source=share&utm_medium=member_android>
20. <https://github.com/priya42bagde/JavaScriptCodingInterviewQuestions>
21. <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7206337334550622211-8x-L?utm_source=share&utm_medium=member_android>
22. <https://www.linkedin.com/posts/prakash-shukla_reactjs-activity-7211603270207696896-ajzz?utm_source=share&utm_medium=member_android>
23. <https://www.linkedin.com/posts/korada-saikiran-910a621a8_frontend-interviews-reactjs-activity-7212363766569590785-l8MR?utm_source=share&utm_medium=member_android>
24. <https://www.linkedin.com/posts/saurabhpandey-_reactjs-interview-questions-ugcPost-7212067746157608960-V8Ol?utm_source=share&utm_medium=member_android>
25. <https://www.linkedin.com/posts/hafijur-rahman0211_react-js-interview-questions-activity-7211896212621320193-tR39?utm_source=share&utm_medium=member_android>
26. Redux questions:

What is Redux, and why is it used in React applications?

🌲 Explain the core principles of Redux (Actions, Reducers, Store).

🌲 What is a Redux Store? How is it different from React component state?

🌲 Describe the flow of data in a Redux application.

🌲 Why is immutability important in Redux, and how is it achieved?

🌲 What are Redux Actions and Action Creators?

🌲 Explain the role of Reducers in Redux.

🌲 What is a Redux Selector? Why and when would you use it?

🌲 Discuss the difference between Redux Thunk and Redux Saga for handling async actions.

🌲 What is a Redux Middleware? Provide examples of commonly used middlewares.

🌲 Explain the purpose of a middleware like redux-thunk in Redux.

🌲 How does a middleware handle actions in the Redux flow?

🌲 What are the main features of Redux Toolkit?

🌲 Explain the concept of 'slices' in Redux Toolkit.

🌲 Why might someone prefer using RTK over traditional Redux?

🌲 Discuss the benefits of using createSlice() in Redux Toolkit.

🌲 How does RTK simplify the Redux setup compared to traditional Redux?

🌲 Discuss the concept of Redux state normalization and its advantages.

🌲 Explain the idea of time-travel debugging with Redux DevTools.

🌲 How does memoization play a role in Redux selectors for performance optimization?

🌲 Discuss the potential drawbacks of overusing Redux in small to medium-sized applications.

🌲 Explain the differences between action creators and action types in Redux.

1. <https://www.linkedin.com/posts/dimple-kumari_learnwithdimple-html-css-activity-7211955040305958912-b84J?utm_source=share&utm_medium=member_android>
2. <https://www.linkedin.com/posts/alpnap_interview-redux-reactjs-activity-7211941729178963971--U5K?utm_source=share&utm_medium=member_android> - redux questions
3. <https://www.linkedin.com/posts/duvvurukishore_virtualization-ugcPost-7211699668517310464-6i3i?utm_source=share&utm_medium=member_android> - virtualization
4. problem <https://www.linkedin.com/posts/sudiptasaha2_machinecoding-frontend-interviewpreparation-activity-7211770011856556034-vjxt?utm_source=share&utm_medium=member_android>
5. <https://www.linkedin.com/posts/bhoomikasahu_happycoding-reactjs-activity-7211941330887794688-okgW?utm_source=share&utm_medium=member_android> - claas component life cycle
6. git command
7. <https://www.linkedin.com/posts/shanudubey_usedeferredvalue-ugcPost-7211397121726431232-K7ml?utm_source=share&utm_medium=member_android> - useDefferedValue
8. pure component and impure
9. state, stateless component
10. how to get value from without using .then.
11. <https://www.linkedin.com/posts/vivek-goswami-680a91203_optimize-your-react-application-ugcPost-7206097090064662529-h9Mo?utm_source=share&utm_medium=member_android>
12. <https://www.linkedin.com/posts/shubhamkamble89_redux-interview-questions-ugcPost-7211325456321531906-B0Hh?utm_source=share&utm_medium=member_android>
13. interview questions attendender post : <https://www.linkedin.com/posts/srikanth-tekumudi-95b47216b_i-have-been-attending-interviews-recently-activity-7205916655716032515-Ga7E?utm_source=share&utm_medium=member_android>
14. <https://www.linkedin.com/feed/update/urn:li:activity:7211364290988007425?utm_source=share&utm_medium=member_android>
15. <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7207985797092777984-Aj52?utm_source=share&utm_medium=member_android>
16. <https://www.linkedin.com/posts/hafijur-rahman0211_react-interview-questions-activity-7207951995805016064-j3_e?utm_source=share&utm_medium=member_android>
17. <https://www.linkedin.com/posts/saikrishnanangunuri_javascriptdeveloper-reactjs-reactjsdeveloper-activity-7211020577774403585-rFrq?utm_source=share&utm_medium=member_android>
18. <https://www.linkedin.com/posts/ayushi1sharma_react-frontenddevelopment-javascript-activity-7210624376948289536-V4IO?utm_source=share&utm_medium=member_android>
19. <https://www.linkedin.com/posts/sai-ram-somanaboina_15-react-component-principles-that-helped-activity-7206991524516024320-gv4m?utm_source=share&utm_medium=member_android>
20. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-angular-javascript-activity-7206323536460738560-Q3UE?utm_source=share&utm_medium=member_android> - interview practice websites
21. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascriptdeveloper-reactjs-activity-7203388950613299200-wmo4?utm_source=share&utm_medium=member_android>
22. <https://www.linkedin.com/posts/prakash-shukla_react-js-activity-7207351078995009536-DZWg?utm_source=share&utm_medium=member_android>
23. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascriptdeveloper-reactjs-activity-7210641124917399552-4kU1?utm_source=share&utm_medium=member_android>
24. <https://www.linkedin.com/posts/hafijur-rahman0211_react-interview-questions-ugcPost-7203378672190341121-cjOn?utm_source=share&utm_medium=member_android>
25. <https://www.linkedin.com/posts/hafijur-rahman0211_react-interview-questions-activity-7206487076924493824-Iq7I?utm_source=share&utm_medium=member_android>
26. <https://www.linkedin.com/posts/shanudubey_react-optimising-tips-ugcPost-7203393947547623424---mo?utm_source=share&utm_medium=member_android>
27. middle ware
28. <https://www.linkedin.com/posts/sai-ram-somanaboina_8-tech-blogs-on-reach-every-web-developer-activity-7203735514649161729-fcYG?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/vivek-shinde-a36591230_reactjs-reactnative-nextjs-activity-7203049589485035520-QX0_?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7206882024463204352-cDTS?utm_source=share&utm_medium=member_android> - react life cycle methods
2. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7206883096258859009-s3Ud?utm_source=share&utm_medium=member_android> - usememo vs useCallback
3. <https://www.linkedin.com/posts/akashsinnghh_here-are-top-30-interview-questions-for-mastering-activity-7205080983757373440-uOzm?utm_source=share&utm_medium=member_android>
4. <https://www.linkedin.com/feed/update/urn:li:activity:7205395002418479104?utm_source=share&utm_medium=member_android>
5. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascript-javascriptdeveloper-activity-7206707769217843202-982e?utm_source=share&utm_medium=member_android>
6. <https://www.linkedin.com/feed/update/urn:li:activity:7206831687958196224?utm_source=share&utm_medium=member_android>
7. <https://www.linkedin.com/posts/thealiraza_react-nextjs-javascript-ugcPost-7204480856982188032-ejNB?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/shivam907_interview-rtcampinterview-interviewexperience-activity-7201961794569920512-a6wp?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/franciscomoretti_intro-to-react-compiler-what-if-you-could-activity-7204792048388816896-P72y?utm_source=share&utm_medium=member_android>
2. <https://www.youtube.com/watch?v=PYHBHK37xlE&t=816s>
3. <https://www.linkedin.com/posts/vivek-shinde-a36591230_authorization-rbac-reactjs-activity-7204756153447231488-1gy1?utm_source=share&utm_medium=member_android>
4. <https://www.linkedin.com/posts/alpnap_react-interview-question-ugcPost-7204327016144265216--nZm?utm_source=share&utm_medium=member_android>
5. <https://youtube.com/playlist?list=PLKhlp2qtUcSZiWKJTi5-5r6IRdHhxP9ZU&si=7H4Dznj0iQcOkGOK>

## <https://www.linkedin.com/posts/careerwithvasanth_reactjs-javascript-mock-activity-7201228026397822977-8jQQ?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascriptdeveloper-reactjs-activity-7209574275186323456-KzRP?utm_source=share&utm_medium=member_android>
2. <https://www.linkedin.com/posts/careerwithvasanth_interview-junior-developer-activity-7204354008491130880-NFAD?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/anchal-sharma-902904171_react-reactjs-javascript-activity-7201052089563693056-UrJC?utm_source=share&utm_medium=member_android>

1. fiber react

## <https://www.linkedin.com/posts/kashif-ur-rahman-662132207_interview-repost-like-activity-7198928844924342274-f-0d?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/shubham-soni-374a86175_frontend-interview-preparation-activity-7201077963042283520-l6p1?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/dimple-kumari_react-webdevelopment-interviewquestions-activity-7204239215218282496-gi6Q?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7201101623497003008-mLYK?utm_source=share&utm_medium=member_android>

## ways to style a react component:

<https://www.linkedin.com/posts/codepapa360_most-asked-react-interview-questions-activity-7196738641375707138-ahFT?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascriptdeveloper-reactjs-activity-7202569840409075712-hvf8?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/vivek-goswami-680a91203_frontend-interview-question-activity-7209728187612778498-sfB0?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/dimple-kumari_react-interview-questions-activity-7202194709635555328-Y9RH?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/saikrishnanangunuri_javascript-javascriptdeveloper-reactjs-activity-7200890077588176896-VbP0?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/careerwithvasanth_interview-reactjs-important-activity-7197469041022623744-veQG?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7200071813547139072-RP2d?utm_source=share&utm_medium=member_android>

## router hooks

<https://www.linkedin.com/posts/codepapa360_react-router-hooks-in-details-ugcPost-7195800620220526592-VJu6?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/santhosh-kumar-b-424785305_top-30-react-js-activity-7198706620464267265-MyEh?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/sai-ram-somanaboina_yesterday-i-sat-down-for-2-hours-and-looked-activity-7199742629541027840-_eHd?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/careerwithvasanth_reactjs-mock-interview-activity-7200156172350365696-pvtU?utm_source=share&utm_medium=member_android>

1. <https://www.linkedin.com/posts/amit-sharma-67b03a138_performance-ugcPost-7197978293025869824-J_zE?utm_source=share&utm_medium=member_android>

## react virtualized

## Popup Drilling :

State should be held by the highest parent component in the stack that requires access to the state.

To illustrate, we have many nested components. The component at the top and bottom of the stack need access to the state.

To do this without Context, we will need to pass the state as "props" through each nested component. This is called **"prop drilling"**.

**Ex :**

import { useState } from "react";

import ReactDOM from "react-dom/client";

function Component1() {

const [user, setUser] = useState("Jesse Hall");

return (

<>

<h1>{`Hello ${user}!`}</h1>

<Component2 user={user} />

</>

);

}

function Component2({ user }) {

return (

<>

<h1>Component 2</h1>

<Component3 user={user} />

</>

);

}

function Component3({ user }) {

return (

<>

<h1>Component 3</h1>

<Component4 user={user} />

</>

);

}

function Component4({ user }) {

return (

<>

<h1>Component 4</h1>

<Component5 user={user} />

</>

);

}

function Component5({ user }) {

return (

<>

<h1>Component 5</h1>

<h2>{`Hello ${user} again!`}</h2>

</>

);

}

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

root.render(<Component1 />);

Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.

The solution is to create context.

## proxy

1. Style the components - <https://www.linkedin.com/posts/theabdul-rehman345_style-your-react-app-ugcPost-7186844774258143232-f0Bi?utm_source=share&utm_medium=member_android>

## Implement code more slices

1. <https://www.linkedin.com/posts/huzaifa-khan-officia_reactjs-interview-questions-ugcPost-7195733385338613760-q8Vq?utm_source=share&utm_medium=member_android>

## throttle & denouncing

## Micro frontend - <https://www.linkedin.com/posts/priya-bagde_frontend-javascript-react-activity-7189889801548943360-m3ZX?utm_source=share&utm_medium=member_desktop>

1. <https://www.linkedin.com/posts/sumitsoni0226_reactjs-interview-activity-7197223864727166976-_4Bn?utm_source=share&utm_medium=member_android>

## new react feature: ( React 19)

<https://www.linkedin.com/posts/shanudubey_useoptimistic-ugcPost-7191268620990095360-kx2K?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/slobodan-gajic_codewithsloba-programming-reactjs-activity-7190657336183607298-nLeY?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/parthib-mohanta_react-19-ugcPost-7191503465993363456-kzuT?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/mollerjorge_reactjs-ugcPost-7191823506743967747-cw-5?utm_source=share&utm_medium=member_desktop>

<https://www.linkedin.com/feed/update/urn:li:activity:7192590946641866754?utm_source=share&utm_medium=member_android>

<https://www.linkedin.com/posts/wajid-ali-28ab63122_react-19-use-hook-activity-7193363025255448577-uhDF?utm_source=share&utm_medium=member_android>

## React import / export :

<https://www.linkedin.com/posts/shanudubey_import-export-in-react-activity-7193080358228103168-1OKp?utm_source=share&utm_medium=member_android>

## Performance - <https://www.linkedin.com/posts/parthib-mohanta_optimizing-performance-in-react-application-activity-7192729630292770818-_gr0?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/theabdul-rehman345_react-interview-questions-and-answers-ugcPost-7189062382181109760-sVs4?utm_source=share&utm_medium=member_android>

## performance improvement

<https://www.linkedin.com/posts/saikrishnanangunuri_reactjs-javascriptdeveloper-reactjs-activity-7195296021998944256-df50?utm_source=share&utm_medium=member_android>

## <https://www.linkedin.com/posts/codepapa360_when-to-use-closures-in-javascript-ugcPost-7195761369684533248-ych6?utm_source=share&utm_medium=member_android> - closure it's looks like reducer and it acts as memorization

## 1. What is React?

## <Strict mode>

## 2. What is useMemo ?

## 3. What are the features of React ?

## 4. What is JSX ?

## 5. What is DOM ?

## 6. What is Virtual Dom ?

## 7. What is component life cycle of React class component ?

## 8. What are fragments in react ?

## 9. What are props in react ?

## 10. What are synthetic events in react ?

## 11. What are the difference between Package.json and Package.lock.json ?

## 12. What are the differences between client side and server side rendering ?

## 13. What is state in Reactjs?

## 14. What are props ?

## 15. What are the differences between State and Props in react ?

## 16. What is props drilling ?

## 17. What are the disadvantages of props drilling and How we can avoid props drilling ?

## 18. What are Pure components in React ?

## 19. Whatare Ref’s in React?

## 20. What is meant by forward ref ?

## 21. What are Error boundaries ?

## 22. What are Higher order components in react ?

## 23. What are the differences between controlled and uncontrolled components ?

## 24. What is useCallback ?

## 25. What are the differences between useMemo and useCallback ?

## 26. What are keys in React ?

## 27. What is Lazy loading in React ?

## 28. What is suspense in React ?

## 29. What are custom hooks ?

## 30. What is useReducer hook ?

## 31. What are Portals in react ?

## 32. What is context in react ?

## 33. Practical question: Give an example of context api usage ?

## 34. What is the purpose of callback function as an argument of setState()?

## 35. Practical question: create a custom hook for increment/decrement counter ?

## 36. Which lifecycle hooks in class component are replaced with useEffect in functional components ?

## 37. What is Strict mode in react ?

## 38. What are the different ways to pass data from child component to parent component in react ?

## 39. Practical question: How to send data from child to parent using callback functions ?

## 40. Practical question: How to send the data from child component to parent using useRef ?

## 41. How do you optimize your react application ?

## 42. How would you consume a RESTful JSON API in reactjs?

## 43.different design patterns used in react?

## 44. context api vs redux

## 45. prop types in react(How to apply validation on props in react)

## 46. What are React Mixins ?

## 47. what are the different hooks you have used ?

## 48. What are render props in react ?

## 49. What are the different types of exports and imports ?

## 50. What are the differences between create element vs clone element in react ?

## 51. When to use useState and useReducer?

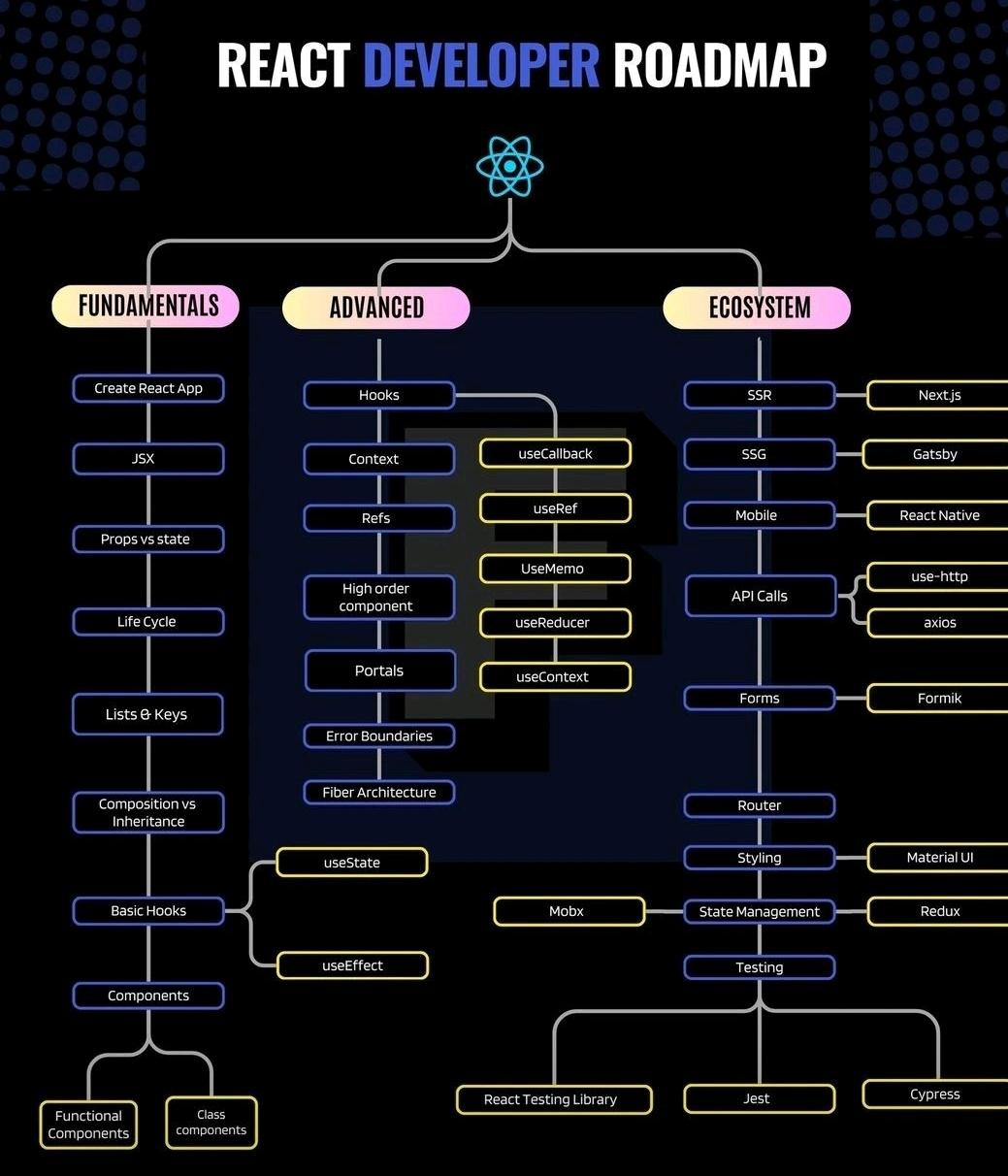
## 52. What is flushSync in react ?

## Reconciliation

1. <https://www.linkedin.com/posts/saikrishnanangunuri_javascript-javascriptdeveloper-reactjs-activity-7193646072928350209-rDpv?utm_source=share&utm_medium=member_android>
2. <https://www.linkedin.com/posts/alifuzzaman-limu-4a20b71bb_top-30-react-interview-questions-ugcPost-7193597453437145089-Gkls?utm_source=share&utm_medium=member_android>
3. <https://www.linkedin.com/posts/parthib-mohanta_react-js-interview-questions-activity-7193612607113564160-EsFF?utm_source=share&utm_medium=member_android>
4. <https://www.linkedin.com/posts/rajatgajbhiye_knowing-react-increases-your-value-in-the-activity-7193828213960151040-uoIH?utm_source=share&utm_medium=member_android>
5. <https://www.linkedin.com/posts/duvvurukishore_react-interviewquestions-interviewquestions-activity-7150327830889816064-0Pdv?utm_source=share&utm_medium=member_android>
6. <https://thetshaped.dev/p/15-react-component-principles-for-better-design>
7. <https://www.linkedin.com/posts/sakhamurinaveen_react-design-patterns-container-components-activity-7205901831514857474-CIA3?utm_source=share&utm_medium=member_android>
8. <https://www.linkedin.com/posts/rajatgajbhiye_dont-overwhelm-to-learn-reactjs-reactjs-activity-7201076470310436865-zGHb?utm_source=share&utm_medium=member_android>
9. <https://www.linkedin.com/posts/srikanth-tekumudi-95b47216b_reactabrinabrdepth-webdevelopment-javascript-activity-7207049783734067200-wnIR?utm_source=share&utm_medium=member_android>
10. Ndnndndn

# Answers :

Interview questions & answers: <https://learndepth.com/>



## React JS:

React is a popular JavaScript library for building user interfaces, primarily for single-page applications. It allows developers to create reusable UI components, making the development process more efficient and maintainable. React follows a component-based architecture, where the UI is broken down into small, isolated pieces that manage their own state. Developed by Facebook.

## 𝗥𝗲𝗰𝗼𝗻𝗰𝗶𝗹𝗶𝗮𝘁𝗶𝗼𝗻 𝗶𝗻 𝗥𝗲𝗮𝗰𝘁

Reconciliation is the process React uses to update the web page efficiently. Instead of reloading the whole page when something changes, React only updates the parts that need to change.

𝗘𝘅𝗮𝗺𝗽𝗹𝗲: Imagine you have a list of items on a webpage. If you add a new item, you don't want the whole list to reload, just the new item to appear. This is what reconciliation helps with.

𝗩𝗶𝗿𝘁𝘂𝗮𝗹 𝗗𝗢𝗠: Think of the virtual DOM as a blueprint of your webpage. React keeps this blueprint and compares it to the actual webpage. When you make changes, React updates the blueprint first.

𝗗𝗶𝗳𝗳𝗶𝗻𝗴 𝗔𝗹𝗴𝗼𝗿𝗶𝘁𝗵𝗺: React looks at the differences between the old and new blueprints. It then figures out the smallest number of changes needed to update the actual webpage to match the new blueprint.

𝗛𝗼𝘄 𝗜𝘁 𝗪𝗼𝗿𝗸𝘀:

When you change something in your React component, React:

• Compares the old and new blueprints.

• Finds what has changed (like a text update or a new item).

• Updates only those specific parts on the actual webpage.

𝗖𝗼𝗱𝗲 𝗘𝘅𝗮𝗺𝗽𝗹𝗲: In the example, when you click the button, only the message inside the <h1> changes from "Hello, world!" to "Hello, React!" instead of reloading the entire page.

𝗖𝗼𝗻𝗰𝗹𝘂𝘀𝗶𝗼𝗻: Reconciliation helps React update web pages quickly and efficiently by only changing what needs to be changed. Understanding this can help you write better and faster React apps.

## React JS over other frameworks :

Component-Based Architecture: React lets you build reusable, modular components, making your code more maintainable and scalable.

🔹 Virtual DOM: It enhances performance by updating only the parts of the DOM that have changed, ensuring a smooth user experience.

🔹 Rich Ecosystem and Community: With a vast array of libraries, tools, and a strong community, React makes development easier. Integrations like Redux and React Router add powerful features.

🔹 Developer Experience: JSX syntax makes your code intuitive and readable. React's declarative style simplifies the UI development process, making your code predictable and easier to debug.

🔹 Flexibility: Ideal for both single-page apps (SPAs) and complex enterprise solutions. React Native also lets you extend your web development skills to mobile apps.

🔹 Stability and Support: React is consistently updated with a strong commitment to backward compatibility, thanks to Facebook's ongoing support.

🔹 Performance Optimization: React Hooks and tools like useMemo and useCallback help manage state and prevent unnecessary re-renders.

🔹 SEO-Friendly: Improve SEO and initial load times with server-side rendering (SSR) using frameworks like Next.js.

## JSX

JSX (JavaScript XML) is a syntax extension for JavaScript used with React to describe what the UI should look like. It allows you to write HTML-like code within JavaScript, making it easier to create and visualize the structure of your components.

const element = <h1>Hello, world!</h1>;

JSX is not HTML, but it looks similar. Under the hood, it gets transformed into JavaScript objects by tools like Babel, creating React.createElement calls.

### **DOM?**

The DOM (Document Object Model) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content. The DOM represents the document as nodes and objects, allowing languages like JavaScript to interact with and manipulate the document dynamically.

## Virtual DOM

The Virtual DOM is a concept used in libraries like React to optimize performance. It is a lightweight copy of the actual DOM kept in memory. When the state of an application changes, React creates a new Virtual DOM tree and compares it with the previous one. This process, known as "reconciliation," determines the most efficient way to update the actual DOM, minimizing direct manipulations and enhancing performance.

## What is the Component Lifecycle of a React Class Component?

The lifecycle of a React class component can be divided into three main phases:

1. **Mounting**: When a component is being inserted into the DOM.
   1. constructor()
   2. static getDerivedStateFromProps()
   3. render()
   4. componentDidMount()
2. **Updating**: When a component is being re-rendered due to changes in props or state.
   1. static getDerivedStateFromProps()
   2. shouldComponentUpdate()
   3. render()
   4. getSnapshotBeforeUpdate()
   5. componentDidUpdate()
3. **Unmounting**: When a component is being removed from the DOM.
   1. componentWillUnmount()
4. constructor

The constructor is called when the component is initialized. It is used to set up initial state and bind event handlers.

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

this.handleClick = this.handleClick.bind(this);

}

}

1. static getDerivedStateFromProps(props, state)

This method is called right before rendering, both on the initial mount and on subsequent updates. It allows the state to be updated based on the props.

static getDerivedStateFromProps(nextProps, prevState) {

if (nextProps.someValue !== prevState.someValue) {

return { someValue: nextProps.someValue };

}

return null;

}

1. render()

The render method is required in every class component. It examines this.props and this.state and returns one of the following types: React elements, arrays, fragments, portals, strings, numbers, or booleans (null).

render() {

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

}

1. componentDidMount()

This method is invoked immediately after a component is mounted (inserted into the tree). Initialization that requires DOM nodes should go here. It's a good place to make network requests.

componentDidMount() {

fetch('/api/data')

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

.then(data => this.setState({ data }));

}

1. shouldComponentUpdate(nextProps, nextState)

This method allows you to prevent re-rendering based on changes in props or state. By default, it returns true. If you return false, the component will not update.

shouldComponentUpdate(nextProps, nextState) {

return nextProps.someValue !== this.props.someValue;

}

1. getSnapshotBeforeUpdate(prevProps, prevState)

This method is called right before the most recently rendered output is committed to the DOM. It allows you to capture some information from the DOM (e.g., scroll position) before it potentially changes.

getSnapshotBeforeUpdate(prevProps, prevState) {

if (prevProps.list.length < this.props.list.length) {

return this.listRef.scrollHeight;

}

return null;

}

1. componentDidUpdate(prevProps, prevState, snapshot)

This method is called immediately after updating occurs. It is a good place to perform operations that require the DOM to be in place, such as making network requests based on updated props or updating the DOM in response to prop or state changes.

componentDidUpdate(prevProps, prevState, snapshot) {

if (snapshot !== null) {

this.listRef.scrollTop = this.listRef.scrollHeight - snapshot;

}

}

1. componentWillUnmount()

This method is called immediately before a component is destroyed. It is used to clean up any side effects or subscriptions that were created in componentDidMount.

componentWillUnmount() {

clearInterval(this.timerID);

}

## virtual dom / reconciliation / diff algorithm / fiber

<https://www.linkedin.com/posts/cybersecboy_react-reactjs-javascript-activity-7225387265101897728-ZqFm?utm_source=share&utm_medium=member_android>

## Fragment

Fragments are a way to group a list of children without adding extra nodes to the DOM. This is useful when rendering multiple elements in a component without wrapping them in an additional element like a div.

import React from 'react';

function MyComponent() {

return (

<React.Fragment>

<h1>Heading</h1>

<p>Paragraph</p>

</React.Fragment>

);

}

Short syntax for fragments:

function MyComponent() {

return (

<>

<h1>Heading</h1>

<p>Paragraph</p>

</>

);

}

## Props

Props (short for "properties") are read-only attributes passed from a parent component to a child component. They allow data to be passed and reused across components, promoting modularity and reusability.

**Key features:**

* Immutability: Cannot be modified by the child component.
* Data Flow: Unidirectional, from parent to child.
* Usage: Accessed via this.props in class components or directly in function parameters in functional components.

function Greeting(props) {

return <h1>Hello, {props.name}!</h1>;

}

function App() {

return <Greeting name="Alice" />;

}

### **Synthetic Events in React?**

Synthetic Events are objects that act as a cross-browser wrapper around the browser's native event system. They are designed to provide a consistent interface for handling events in React applications, ensuring that events work the same across all browsers.

Key features:

1. Normalization: Provides a consistent API for different browsers.
2. Performance: Pooled for performance, meaning event objects are reused.
3. Properties: Include all the properties of native events, like event.target, event.preventDefault(), and event.stopPropagation().

function handleClick(event) {

console.log(event.type); // "click"

event.preventDefault();

}

<button onClick={handleClick}>Click me</button>

## package.json vs package-lock.json

**package.json**:

1. Describes the project: Contains metadata like the project’s name, version, author, and scripts.
2. Lists dependencies: Specifies which packages the project needs.
3. Used for: High-level information, package management, and project configuration.

**package-lock.json**:

1. Exact versions: Captures the exact versions of dependencies and their dependencies installed at the time of running npm install.
2. Ensures consistency: Guarantees that the same dependencies are installed across different environments.
3. Used for: Dependency tree snapshot, ensuring reproducible builds.

## Client-Side vs Server-Side Rendering

**Client-Side Rendering (CSR)**:

1. Rendering: Happens in the browser using JavaScript.
2. Initial load: Slower, as the browser needs to download and execute JavaScript before rendering.
3. Subsequent interactions: Faster, as only the data is fetched and the DOM is updated.
4. SEO: Challenging, as search engines may struggle with JavaScript-heavy content.

**Server-Side Rendering (SSR)**:

1. Rendering: Happens on the server, and the HTML is sent to the browser.
2. Initial load: Faster, as the browser receives fully rendered HTML.
3. Subsequent interactions: May require full-page reloads or dynamic fetching.
4. SEO: Better, as the content is fully rendered HTML when the page loads.

## State

State is an object that holds data that may change over the lifecycle of the component. It is managed within the component and can be used to control its behavior and render logic.

Key features:

1. Initialization: Set in the constructor in class components, or using the useState hook in functional components.
2. Updates: Trigger re-rendering of the component.
3. Scope: Local to the component, but can be passed down to child components as props.

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState({ count: this.state.count + 1 });

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

## State and Props in React

**State**:

1. **Definition**: State is a built-in object that holds data that may change over the lifecycle of the component.
2. **Modifiability**: State is mutable and can be changed within the component using this.setState in class components or the useState hook in functional components.
3. **Scope**: Local to the component where it is defined.
4. **Usage**: Used to manage dynamic data and handle user interactions within the component.
5. **Updates**: Triggers a re-render when updated.

**Props**:

1. **Definition**: Props (short for "properties") are read-only attributes passed from a parent component to a child component.
2. **Modifiability**: Props are immutable and cannot be changed by the child component.
3. **Scope**: Passed from parent to child components.
4. **Usage**: Used to pass data and event handlers to child components, enabling reusable and modular components.
5. **Updates**: Do not trigger re-renders by themselves but cause re-renders when the parent component re-renders with new props.

## Props Drilling

Props drilling refers to the process of passing data through multiple levels of nested components by passing props from one component to another. This can lead to cumbersome and hard-to-maintain code, especially when the data needs to be accessed by deeply nested components.

**Disadvantages**:

1. **Redundant Code**: Involves passing props through components that do not need the data, leading to cluttered and repetitive code.
2. **Maintenance**: Makes components tightly coupled, increasing the difficulty of maintenance and refactoring.
3. **Readability**: Decreases code readability and makes it harder to understand the flow of data

**Avoiding Props Drilling**:

1. **Context API**: Use React's Context API to create a context and provide data to the entire component tree without passing props through every level.

const MyContext = React.createContext();

function App() {

return (

<MyContext.Provider value={someValue}>

<DeeplyNestedComponent />

</MyContext.Provider>

);

}

function DeeplyNestedComponent() {

const value = React.useContext(MyContext);

return <div>{value}</div>;

}

## Pure Components

Pure Components in React are components that do not re-render if the props and state have not changed. They implement a shouldComponentUpdate method with a shallow comparison of current and next props and state, improving performance by preventing unnecessary re-renders.

class MyPureComponent extends React.PureComponent {

render() {

return <div>{this.props.value}</div>;

}

}

## Refs in react

Refs (short for references) provide a way to access and interact with DOM nodes or React elements created in the render method. They are useful for:

1. **Accessing DOM elements**: When you need to interact directly with the DOM, such as focusing an input field.
2. **Storing mutable values**: Keeping any mutable value that does not cause a re-render when updated.
3. **Integrating with third-party libraries**: When using libraries that require direct DOM manipulation.

function MyComponent() {

const myRef = React.useRef(null);

React.useEffect(() => {

myRef.current.focus();

}, []);

return <input ref={myRef} />;

}

## Forward Ref

Forward Ref is a feature in React that allows a component to pass a ref through it to a child component. It enables parent components to directly access a child component's DOM node or React instance. Forward Refs are typically used in reusable component libraries to give the parent component a way to interact with the child component's DOM.

const MyInput = React.forwardRef((props, ref) => (

<input ref={ref} {...props} />

));

const ParentComponent = () => {

const inputRef = React.useRef();

React.useEffect(() => {

inputRef.current.focus();

}, []);

return <MyInput ref={inputRef} />;

};

## Error Boundaries

Error Boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. They help in gracefully handling runtime errors in React applications.

To create an error boundary, implement either static getDerivedStateFromError() or componentDidCatch() in a class component.

Note: Error Boundaries can only be implemented using class components. However, functional components can use error boundary components to handle errors.

**Example of a class-based Error Boundary and its usage in a functional component:**

class ErrorBoundary extends React.Component {

constructor(props) {

super(props);

this.state = { hasError: false };

}

static getDerivedStateFromError(error) {

return { hasError: true };

}

componentDidCatch(error, errorInfo) {

console.log(error, errorInfo);

}

render() {

if (this.state.hasError) {

return <h1>Something went wrong.</h1>;

}

return this.props.children;

}

}

// Usage in a functional component

const MyComponent = () => {

// Component logic

return <div>Component Content</div>;

};

const App = () => (

<ErrorBoundary>

<MyComponent />

</ErrorBoundary>

);

## Higher-Order Components (HOCs)

Higher-Order Components (HOCs) are functions that take a component and return a new component. They are used to enhance or modify the behavior of the original component, enabling code reuse, logic abstraction, and the manipulation of props.

## Controlled vs Uncontrolled Components

**Controlled Components**:

1. **State Management**: The form data is handled by the React component's state.
2. **Two-way Binding**: The component controls the value of the input field, which means the input's value is driven by the state.
3. **Event Handling**: Requires an onChange event handler to update the state.

const ControlledForm = () => {

const [inputValue, setInputValue] = React.useState('');

const handleChange = (event) => {

setInputValue(event.target.value);

};

return (

<input

type="text"

value={inputValue}

onChange={handleChange}

/>

);

};

**Uncontrolled Components**:

1. **State Management**: The form data is handled by the DOM itself.
2. **One-way Binding**: The component does not control the input's value; the DOM maintains its own state.
3. **Refs**: Typically, refs are used to access the input value.

const UncontrolledForm = () => {

const inputRef = React.useRef(null);

const handleSubmit = (event) => {

event.preventDefault();

console.log(inputRef.current.value);

};

return (

<form onSubmit={handleSubmit}>

<input type="text" ref={inputRef} />

<button type="submit">Submit</button>

</form>

);

};

## useCallback

useCallback is a React hook that returns a memoized version of a callback function. It is used to optimize performance by preventing the creation of new instances of a function on every render, which can be useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

**Syntax:**

const memoizedCallback = useCallback(

() => {

// function body

},

[dependencies]

);

const MyComponent = () => {

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

const handleClick = useCallback(() => {

setCount(count + 1);

}, [count]);

return <button onClick={handleClick}>Increment</button>;

};

### **useMemo vs useCallback**

**Purpose**:

1. **useMemo**: Memoizes a computed value, optimizing performance by caching expensive calculations.
2. **useCallback**: Memoizes a callback function, preventing unnecessary re-creations of functions on each render.

**Return Value**:

1. **useMemo**: Returns a memoized value.
2. **useCallback**: Returns a memoized function.

**Usage**:

1. **useMemo**: Used for memoizing the results of expensive calculations.
2. **useCallback**: Used for memoizing functions to prevent re-creating them, often used when passing callbacks to child components.

const MyComponent = ({ items }) => {

const expensiveCalculation = useMemo(() => {

return items.reduce((acc, item) => acc + item.value, 0);

}, [items]);

return <div>Total: {expensiveCalculation}</div>;

};

const MyComponent = () => {

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

const handleIncrement = useCallback(() => {

setCount(count + 1);

}, [count]);

return <button onClick={handleIncrement}>Increment</button>;

};

## Keys in React

**Keys** are special attributes you need to include when creating lists of elements in React. They help React identify which items have changed, are added, or are removed, thus optimizing the rendering process.

1. **Purpose**: Improve performance by enabling efficient updates of the rendered list.
2. **Requirement**: Should be unique among siblings.
3. **Common Usage**: Used in lists generated by map().

const MyList = ({ items }) => (

<ul>

{items.map(item => (

<li key={item.id}>{item.name}</li>

))}

</ul>

);

## Lazy Loading in React

**Lazy Loading** is a technique in React that delays the loading of components until they are needed, which can improve the initial load time of an application.

1. **Usage**: Use React.lazy() and Suspense to dynamically import components.
2. **Benefit**: Reduces the initial bundle size by splitting code into smaller chunks.

const LazyComponent = React.lazy(() => import('./LazyComponent'));

const App = () => (

<div>

<Suspense fallback={<div>Loading...</div>}>

<LazyComponent />

</Suspense>

</div>

);

## Suspense in React

**Suspense** is a React component that allows you to display a fallback while waiting for a lazy-loaded component to load.

1. **Usage**: Wrap lazy-loaded components with Suspense and provide a fallback UI.
2. **Benefit**: Enhances user experience by showing a loading indicator or placeholder.

const LazyComponent = React.lazy(() => import('./LazyComponent'));

const App = () => (

<div>

<Suspense fallback={<div>Loading...</div>}>

<LazyComponent />

</Suspense>

</div>

);

## Custom Hooks

**Custom Hooks** are functions in React that start with use and allow you to extract and reuse stateful logic between components.

1. **Purpose**: Encapsulate reusable logic.
2. **Creation**: Combine existing hooks (like useState, useEffect, etc.) to create your own hooks.

const useCounter = (initialValue = 0) => {

const [count, setCount] = React.useState(initialValue);

const increment = () => setCount(count + 1);

const decrement = () => setCount(count - 1);

return { count, increment, decrement };

};

const Counter = () => {

const { count, increment, decrement } = useCounter();

return (

<div>

<button onClick={decrement}>-</button>

<span>{count}</span>

<button onClick={increment}>+</button>

</div>

);

};

## Reducer Hook

**useReducer** is a React hook that provides an alternative to useState for managing complex state logic or multiple related state transitions.

1. **Usage**: Takes a reducer function and an initial state, returning the current state and a dispatch function.
2. **Benefit**: Better for managing state with multiple sub-values or state transitions that depend on the previous state.

const initialState = { count: 0 };

const reducer = (state, action) => {

switch (action.type) {

case 'increment':

return { count: state.count + 1 };

case 'decrement':

return { count: state.count - 1 };

default:

return state;

}

};

const Counter = () => {

const [state, dispatch] = React.useReducer(reducer, initialState);

return (

<div>

<button onClick={() => dispatch({ type: 'decrement' })}>-</button>

<span>{state.count}</span>

<button onClick={() => dispatch({ type: 'increment' })}>+</button>

</div>

);

};

## Portals

**Portals** provide a way to render children into a DOM node that exists outside the hierarchy of the parent component.

1. **Purpose**: Useful for rendering elements like modals, tooltips, and dropdowns that need to visually break out of the parent component's container.
2. **Creation**: Use ReactDOM.createPortal(child, container).

import ReactDOM from 'react-dom';

const Modal = ({ isOpen, children }) => {

if (!isOpen) return null;

return ReactDOM.createPortal(

<div className="modal">

{children}

</div>,

document.getElementById('modal-root') // Assuming you have a div with id 'modal-root' in your HTML

);

};

const App = () => {

const [isModalOpen, setIsModalOpen] = React.useState(false);

return (

<div>

<button onClick={() => setIsModalOpen(true)}>Open Modal</button>

<Modal isOpen={isModalOpen}>

<button onClick={() => setIsModalOpen(false)}>Close Modal</button>

</Modal>

</div>

);

};

## Context

**Context** provides a way to pass data through the component tree without having to pass props down manually at every level.

1. **Purpose**: Useful for global data like themes, user information, or settings.
2. **Creation**: Use React.createContext() to create a context.

**Create Context**:

const ThemeContext = React.createContext('light');

**Provide Context Value**:

const App = () => {

const [theme, setTheme] = React.useState('light');

return (

<ThemeContext.Provider value={{ theme, setTheme }}>

<Toolbar />

</ThemeContext.Provider>

);

};

**Consume Context Value**:

const Toolbar = () => {

return (

<div>

<ThemedButton />

</div>

);

};

const ThemedButton = () => {

const { theme, setTheme } = React.useContext(ThemeContext);

return (

<button

style={{ background: theme === 'light' ? '#fff' : '#333', color: theme === 'light' ? '#000' : '#fff' }}

onClick={() => setTheme(theme === 'light' ? 'dark' : 'light')}

>

Toggle Theme

</button>

);

};

## Purpose of Callback Function as an Argument of setState()

In class components, the setState method can take a callback function as its second argument. This callback function is executed once the state has been updated and the component has re-rendered.

* **Purpose**: Ensure code runs after the state has been updated.
* **Use Case**: Useful for actions that depend on the updated state or DOM, like making API calls, updating other state values, or manipulating the DOM.

Example:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0,

};

}

incrementCount = () => {

this.setState({ count: this.state.count + 1 }, () => {

console.log('Count updated:', this.state.count);

});

};

render() {

return (

<div>

<button onClick={this.incrementCount}>Increment</button>

<p>Count: {this.state.count}</p>

</div>

);

}

}

In functional components, this behavior is typically handled with useEffect:

const MyComponent = () => {

const [count, setCount] = React.useState(0);

React.useEffect(() => {

console.log('Count updated:', count);

}, [count]);

const incrementCount = () => {

setCount(count + 1);

};

return (

<div>

<button onClick={incrementCount}>Increment</button>

<p>Count: {count}</p>

</div>

);

};

## Lifecycle Hooks Replaced by useEffect in Functional Components

In class components, there are several lifecycle methods that manage side effects. These can be replaced by useEffect in functional components:

1. **componentDidMount**: Executed once after the initial rendering.
2. **componentDidUpdate**: Executed after every update, except the initial one.
3. **componentWillUnmount**: Executed before the component is removed from the DOM.

In functional components, useEffect can handle these lifecycle phases:

import React, { useEffect } from 'react';

const MyComponent = () => {

useEffect(() => {

// This is equivalent to componentDidMount and componentDidUpdate

console.log('Component mounted or updated');

return () => {

// This is equivalent to componentWillUnmount

console.log('Component will unmount');

};

}, []); // Empty dependency array means this effect runs only once, like componentDidMount

return <div>My Component</div>;

};

## Strict Mode in React

**Strict Mode** is a tool for highlighting potential problems in an application. It does not render any visible UI but activates additional checks and warnings for its descendants.

1. **Purpose**: Identify unsafe lifecycles, legacy API usage, and unexpected side effects.
2. **Usage**: Wrap part of your application with <React.StrictMode>.

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

## Different Ways to Pass Data from Child Component to Parent Component in React

1. **Callback Functions**: Child component calls a function passed down from the parent.
2. **Refs**: Child component modifies a ref passed down from the parent.
3. **Context API**: Both parent and child access shared data via context.
4. **State Management Libraries**: Use libraries like Redux or Zustand to manage shared state.

### **Sending Data from Child to Parent Using Callback Functions**

const ParentComponent = () => {

const [data, setData] = React.useState('');

const handleDataFromChild = (childData) => {

setData(childData);

};

return (

<div>

<ChildComponent sendDataToParent={handleDataFromChild} />

<p>Data from child: {data}</p>

</div>

);

};

const ChildComponent = ({ sendDataToParent }) => {

const sendData = () => {

sendDataToParent('Hello from Child');

};

return <button onClick={sendData}>Send Data</button>;

};

### **Sending Data from Child to Parent Using useRef**

const ParentComponent = () => {

const childDataRef = React.useRef('');

const handleShowData = () => {

alert(childDataRef.current);

};

return (

<div>

<ChildComponent dataRef={childDataRef} />

<button onClick={handleShowData}>Show Data from Child</button>

</div>

);

};

const ChildComponent = ({ dataRef }) => {

React.useEffect(() => {

dataRef.current = 'Data from Child';

}, [dataRef]);

return <div>Child Component</div>;

};

### **Sending Data from Child to Parent Using contextApi**

1. **Create a Context**

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

// Create a Context

const DataContext = createContext();

1. **Provide Context Value**

const DataProvider = ({ children }) => {

const [data, setData] = useState('');

return (

<DataContext.Provider value={{ data, setData }}>

{children}

</DataContext.Provider>

);

};

export { DataContext, DataProvider };

1. **Consume Context Value in Child Component**

import React, { useContext } from 'react';

import { DataContext } from './DataProvider';

const ChildComponent = () => {

const { setData } = useContext(DataContext);

const sendDataToParent = () => {

setData('Hello from Child');

};

return (

<button onClick={sendDataToParent}>Send Data to Parent</button>

);

};

export default ChildComponent;

1. **Consume Context Value in Parent Component**

import React, { useContext } from 'react';

import { DataContext } from './DataProvider';

import ChildComponent from './ChildComponent';

const ParentComponent = () => {

const { data } = useContext(DataContext);

return (

<div>

<p>Data from child: {data}</p>

<ChildComponent />

</div>

);

};

export default ParentComponent;

1. **Wrap the App Component with the DataProvider**

import React from 'react';

import ReactDOM from 'react-dom';

import { DataProvider } from './DataProvider';

import ParentComponent from './ParentComponent';

const App = () => (

<DataProvider>

<ParentComponent />

</DataProvider>

);

ReactDOM.render(<App />, document.getElementById('root'));

### **Optimizing a React Application**

1. **Code Splitting**: Use dynamic imports and React.lazy to split code and load components only when needed.

const LazyComponent = React.lazy(() => import('./LazyComponent'));

1. **Memoization**: Use React.memo, useMemo, and useCallback to memoize components, values, and functions to prevent unnecessary re-renders.

const MemoizedComponent = React.memo(MyComponent);

const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);

const memoizedCallback = useCallback(() => doSomething(a, b), [a, b]);

1. **Pure Components**: Use React.PureComponent or React.memo for components that rely on immutable props.

const PureComp = React.memo(MyComponent);

1. **Avoiding Inline Functions**: Define functions outside of the render method to avoid creating new instances on every render.

const handleClick = () => {

// handle click

};

return <button onClick={handleClick}>Click Me</button>;

1. **Optimizing State**: Use local component state instead of global state where possible. Avoid deep nested state objects.

const [value, setValue] = useState(initialValue);

1. **Virtualization**: Use libraries like react-window or react-virtualized to efficiently render large lists.

import { FixedSizeList as List } from 'react-window';

1. **Using Key Properly**: Ensure unique keys for list items to help React optimize rendering.

items.map(item => <ListItem key={item.id} item={item} />);

## Consuming a RESTful JSON API in ReactJS

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

const App = () => {

const [data, setData] = useState(null);

useEffect(() => {

fetch('https://api.example.com/data')

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

.then(data => setData(data));

}, []);

return (

<div>

{data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}

</div>

);

};

export default App;

**Using axios:**

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

import axios from 'axios';

const App = () => {

const [data, setData] = useState(null);

useEffect(() => {

axios.get('https://api.example.com/data')

.then(response => setData(response.data));

}, []);

return (

<div>

{data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}

</div>

);

};

export default App;

## Context API vs Redux

**Context API**:

1. **Use Case**: Simple state management or global data like themes, user info.
2. **Setup**: Minimal setup, integrated with React.
3. **Performance**: Can lead to unnecessary re-renders if not optimized.
4. **API**: createContext, Provider, useContext.

**Redux**:

1. **Use Case**: Complex state management with multiple actions and reducers.
2. **Setup**: Requires setup of store, actions, reducers, and middleware.
3. **Performance**: More optimized for large applications with fine-grained control over updates.
4. **API**: createStore, Provider, connect, useSelector, useDispatch.

## Prop Types in React

**PropTypes** are used to validate the props passed to a component to ensure they are of the expected type.

import React from 'react';

import PropTypes from 'prop-types';

const MyComponent = ({ name, age, isActive }) => (

<div>

<p>Name: {name}</p>

<p>Age: {age}</p>

<p>Active: {isActive ? 'Yes' : 'No'}</p>

</div>

);

MyComponent.propTypes = {

name: PropTypes.string.isRequired,

age: PropTypes.number,

isActive: PropTypes.bool,

};

MyComponent.defaultProps = {

age: 0,

isActive: false,

};

export default MyComponent;

**Common PropTypes**:

1. PropTypes.string
2. PropTypes.number
3. PropTypes.bool
4. PropTypes.array
5. PropTypes.object
6. PropTypes.func
7. PropTypes.node (anything that can be rendered)
8. PropTypes.element (React element)

## Default Props

You can define default values for props using the defaultProps property.

const Greeting = ({ name }) => <div>Hello, {name}!</div>;

Greeting.defaultProps = {

name: 'Guest',

};

## React mixins

**React Mixins** were a way to share code between components in older versions of React, primarily with class components. They allowed you to define a mixin containing methods and state, which could be shared across multiple components.

**Key Points**:

1. **Usage**: Mixins are included in the mixins array when defining a React component class.
2. **Deprecation**: Mixins have been deprecated in favor of higher-order components (HOCs), render props, and hooks due to issues with state and method conflicts.

const MyMixin = {

componentDidMount() {

console.log('Component did mount');

},

myMethod() {

return 'Hello from mixin!';

}

};

const MyComponent = React.createClass({

mixins: [MyMixin],

render() {

return <div>{this.myMethod()}</div>;

}

});

## Render Props in React

**Render Props** is a technique for sharing code between React components using a prop whose value is a function.

**Key Points**:

1. **Purpose**: Enables flexible and reusable component logic.
2. **Usage**: The render prop function receives props and returns React elements.

const MouseTracker = ({ render }) => {

const [position, setPosition] = useState({ x: 0, y: 0 });

const handleMouseMove = event => {

setPosition({ x: event.clientX, y: event.clientY });

};

return (

<div style={{ height: '100vh' }} onMouseMove={handleMouseMove}>

{render(position)}

</div>

);

};

const App = () => (

<MouseTracker render={({ x, y }) => (

<h1>Mouse position: ({x}, {y})</h1>

)} />

);

## Different Types of Exports and Imports

1. **Named Exports**: Exports multiple values from a module.

// Named Export

export const value1 = 'value1';

export const value2 = 'value2';

// Named Import

import { value1, value2 } from './module';\

1. **Default Exports**: Exports a single value from a module.

// Default Export

const myDefault = 'default value';

export default myDefault;

// Default Import

import myDefault from './module';

1. **Aggregating Modules**: Re-export from another module.

// ModuleA.js

export const valueA = 'A';

// ModuleB.js

export \* from './ModuleA'; // Re-export everything from ModuleA

// Importing in another file

import { valueA } from './ModuleB';

## Differences Between createElement and cloneElement in React

**React.createElement**:

1. **Purpose**: Creates and returns a new React element of the given type.
2. **Usage**: Used internally by JSX but can be used directly for dynamic component creation.
3. **Syntax**: React.createElement(type, [props], [...children]).

const element = React.createElement('div', { className: 'my-div' }, 'Hello, World!');

**React.cloneElement**:

1. **Purpose**: Clones and returns a new React element using the element passed as the first argument.
2. **Usage**: Useful for passing new props to existing elements or children.
3. **Syntax**: React.cloneElement(element, [props], [...children]).

const originalElement = <div className="original">Original</div>;

const clonedElement = React.cloneElement(originalElement, { className: 'cloned' });

## Routing

Routing in React is typically managed using libraries that provide the ability to handle navigation, URL matching, and rendering of components based on the current URL. The most commonly used routing library for React is react-router-dom.

**Core Components :**

1. **BrowserRouter**: A router that uses the HTML5 history API to keep UI in sync with the URL.
2. **Routes**: A container component that holds route definitions.
3. **Route**: Defines a mapping between a URL path and a component.
4. **Link**: Provides navigation between routes.
5. **Outlet**: Renders nested routes.
6. **Navigate**: Programmatically navigates to a different route.

**Basic setup :**

// App.js

import React from 'react';

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

const Home = () => <h2>Home Page</h2>;

const About = () => <h2>About Page</h2>;

const Contact = () => <h2>Contact Page</h2>;

const App = () => {

return (

<Router>

<nav>

<ul>

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

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

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

</ul>

</nav>

<Routes>

<Route path="/" element={<Home />} />

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

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

</Routes>

</Router>

);

};

export default App;

**Nested Routes :**

You can define nested routes using Outlet:

// App.js

import React from 'react';

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

const Home = () => (

<div>

<h2>Home Page</h2>

<Outlet />

</div>

);

const About = () => <h2>About Page</h2>;

const Contact = () => <h2>Contact Page</h2>;

const SubContact = () => <h2>Sub Contact Page</h2>;

const App = () => {

return (

<Router>

<nav>

<ul>

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

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

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

<li><Link to="/contact/sub">Sub Contact</Link></li>

</ul>

</nav>

<Routes>

<Route path="/" element={<Home />}>

<Route path="sub" element={<SubContact />} />

</Route>

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

<Route path="/contact/\*" element={<Contact />} />

</Routes>

</Router>

);

};

export default App;

**Programmatic Navigation:**

import React from 'react';

import { useNavigate } from 'react-router-dom';

const NavigateButton = () => {

const navigate = useNavigate();

const handleClick = () => {

navigate('/about');

};

return <button onClick={handleClick}>Go to About Page</button>;

};

export default NavigateButton;

**Route Parameters:**

// UserProfile.js

import React from 'react';

import { useParams } from 'react-router-dom';

const UserProfile = () => {

const { userId } = useParams();

return <h2>User Profile for User ID: {userId}</h2>;

};

export default UserProfile;

// App.js

import React from 'react';

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

import UserProfile from './UserProfile';

const App = () => {

return (

<Router>

<nav>

<ul>

<li><Link to="/user/1">User 1</Link></li>

<li><Link to="/user/2">User 2</Link></li>

</ul>

</nav>

<Routes>

<Route path="/user/:userId" element={<UserProfile />} />

</Routes>

</Router>

);

};

export default App;

**Redirects :**

To redirect from one route to another, use the Navigate component:

import React from 'react';

import { Navigate } from 'react-router-dom';

const RedirectToHome = () => {

return <Navigate to="/" />;

};

export default RedirectToHome;

**Protected Routes :**

// PrivateRoute.js

import React from 'react';

import { Navigate } from 'react-router-dom';

const PrivateRoute = ({ element, isAuthenticated }) => {

return isAuthenticated ? element : <Navigate to="/login" />;

};

export default PrivateRoute;

// App.js

import React from 'react';

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

import PrivateRoute from './PrivateRoute';

const App = () => {

const isAuthenticated = true; // Determine authentication state here

return (

<Router>

<nav>

<ul>

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

<li><Link to="/profile">Profile</Link></li>

<li><Link to="/login">Login</Link></li>

</ul>

</nav>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/login" element={<Login />} />

<Route path="/profile" element={<PrivateRoute element={<Profile />} isAuthenticated={isAuthenticated} />} />

</Routes>

</Router>

);

};

export default App;

## useEffect Hook

**Purpose**: Perform side effects in functional components, such as data fetching, subscriptions, or manually changing the DOM.

**Behavior**:

* 1. Runs after the render is committed to the screen.
  2. Optionally takes a dependency array to control when the effect should re-run.

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

const MyComponent = () => {

const [data, setData] = useState(null);

useEffect(() => {

// Perform data fetching or any side effect

fetch('https://api.example.com/data')

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

.then(data => setData(data));

// Cleanup function (optional)

return () => {

// Cleanup code, e.g., unsubscribe from a service

};

}, []); // Empty array means this effect runs only once, similar to componentDidMount

return (

<div>

{data ? <pre>{JSON.stringify(data, null, 2)}</pre> : 'Loading...'}

</div>

);

};

export default MyComponent;

## useEffect Cases

1. Basic Side Effect

A basic useEffect that runs after every render.

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

const BasicEffect = () => {

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

useEffect(() => {

console.log('Component rendered');

});

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

};

export default BasicEffect;

1. Effect with Dependencies

A useEffect that runs only when specific dependencies change.

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

const EffectWithDependencies = () => {

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

useEffect(() => {

console.log(`Count changed to: ${count}`);

}, [count]);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

};

export default EffectWithDependencies;

1. Effect with Cleanup

A useEffect that includes a cleanup function, often used for subscriptions or removing event listeners.

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

const EffectWithCleanup = () => {

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

useEffect(() => {

console.log('Setting up an interval');

const interval = setInterval(() => {

setCount(prevCount => prevCount + 1);

}, 1000);

// Cleanup function

return () => {

console.log('Cleaning up the interval');

clearInterval(interval);

};

}, []);

return (

<div>

<p>Count: {count}</p>

</div>

);

};

export default EffectWithCleanup;

1. Effect that Runs Once

A useEffect that runs only once, similar to componentDidMount.

import React, { useEffect } from 'react';

const EffectRunsOnce = () => {

useEffect(() => {

console.log('Effect runs once');

// Fetch data or perform a one-time operation

}, []);

return (

<div>

<p>This effect runs only once</p>

</div>

);

};

export default EffectRunsOnce;

## useLayoutEffect Hook

**Purpose**: Similar to useEffect, but it fires synchronously after all DOM mutations. It can be used to read layout from the DOM and synchronously re-render.

**Behavior**:

1. Runs synchronously after the DOM updates but before the browser has painted.
2. Useful for reading layout and making changes before the browser performs a repaint.

import React, { useLayoutEffect, useRef } from 'react';

const LayoutComponent = () => {

const divRef = useRef(null);

useLayoutEffect(() => {

// Perform side effects that require reading layout

const { height } = divRef.current.getBoundingClientRect();

console.log('Div height:', height);

// Cleanup function (optional)

return () => {

// Cleanup code

};

}, []); // Empty array means this effect runs only once, similar to componentDidMount

return (

<div ref={divRef} style={{ height: '200px' }}>

Layout Effect

</div>

);

};

export default LayoutComponent;

## useMemo

useMemo is a hook in React that allows you to optimize performance by memoizing expensive calculations. It returns a memoized value that only recalculates when one of its dependencies changes. This can help prevent unnecessary recalculations on every render, improving the performance of your application.

import React, { useMemo } from 'react';

function MyComponent({ a, b }) {

const memoizedValue = useMemo(() => {

// Some expensive calculation

return a + b;

}, [a, b]);

return <div>{memoizedValue}</div>;

}

In this example, the expensive calculation a + b is only recomputed when either a or b changes. If neither changes, the previously memoized value is returned, avoiding unnecessary recalculations and improving performance.

## Style the react components

Styling React components can be done in several ways:

1. **CSS Modules**:

import styles from './Button.module.css';

function Button() {

return <button className={styles.button}>Click Me</button>;

}

1. **Styled Components** (a CSS-in-JS library):

import styled from 'styled-components';

const Button = styled.button`

background-color: blue;

color: white;

border: none;

padding: 10px 20px;

cursor: pointer;

`;

function App() {

return <Button>Click Me</Button>;

}

1. **Inline Styles**:

function Button() {

return (

<button style={{ backgroundColor: 'blue', color: 'white', padding: '10px 20px' }}>

Click Me

</button>

);

}

1. **Tailwind CSS**:

function Button() {

return (

<button className="bg-blue-500 text-white p-2">

Click Me

</button>

);

}

## Children props:

The { children } prop in the AuthProvider component is a special prop in React that represents the child components nested inside the AuthProvider when it is used in your application. This allows you to wrap other components with the AuthProvider so that these child components can access the context values provided by AuthProvider.

When you define a component that receives children, you're essentially saying that this component will render whatever elements are passed between its opening and closing tags.

<AuthProvider>

<Router>

<Switch>

<Route path="/login" component={Login} />

<PrivateRoute path="/home" component={Home} />

</Switch>

</Router>

</AuthProvider>

Here, <Router>, <Switch>, <Route>, and <PrivateRoute> are all children of AuthProvider. The AuthProvider component will render these children.

## Fiber in react

React Fiber is a reimplementation of the React reconciliation algorithm to improve its performance and capabilities. Here are the key points:

1. **Incremental Rendering**: React Fiber breaks rendering work into chunks and spreads it over multiple frames. This helps in maintaining a responsive UI by not blocking the main thread for long periods.
2. **Prioritization**: Fiber allows React to assign priority to different types of updates. For example, animations and user interactions can be given higher priority than data fetching.
3. **Concurrency**: With Fiber, React can pause and resume work, allowing it to handle complex rendering tasks more efficiently. This means that while some updates are in progress, others can be started or continued without waiting for the previous ones to complete.

## Middleware

In the context of Redux (a state management library for React):

1. Middleware provides a third-party extension point between dispatching an action and the moment it reaches the reducer.
2. Examples include logging, crash reporting, and performing asynchronous tasks like API calls.

const logger = store => next => action => {

console.log('dispatching', action);

let result = next(action);

console.log('next state', store.getState());

return result;

};

const store = createStore(reducer, applyMiddleware(logger));

## state, stateless component

1. **Stateful Component: Manages state internally using this.state and this.setState.**

In modern React, we often use functional components with hooks to manage state.

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

increment = () => {

this.setState({ count: this.state.count + 1 });

};

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

1. **Stateless Component**: Doesn't manage state, uses props.

function Display({ count }) {

return <p>Count: {count}</p>;

}

function App() {

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

return (

<div>

<Display count={count} />

<button onClick={() => setCount(count + 1)}>Increment</button>

</div>

);

}

## useDefferedValue in react 18

useDeferredValue is a React hook introduced in React 18. It allows you to defer a value until the next render to avoid blocking the main thread with expensive computations.

**Key Points**:

1. Helps in deferring updates that are not critical to keep the UI responsive.
2. Useful for scenarios like search inputs where the results can be computed with a slight delay.

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

function SearchResults({ query }) {

const deferredQuery = useDeferredValue(query);

const results = performSearch(deferredQuery); // Assume performSearch is an expensive operation

return (

<ul>

{results.map(result => (

<li key={result.id}>{result.name}</li>

))}

</ul>

);

}

function App() {

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

return (

<div>

<input

type="text"

value={query}

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

/>

<SearchResults query={query} />

</div>

);

}

## Outlet vs <component />

**<Outlet />**:

1. Used in nested routes to render child routes.
2. Acts as a placeholder for where the child routes will be rendered.

import { Outlet } from 'react-router-dom';

function Dashboard() {

return (

<div>

<h1>Dashboard</h1>

<Outlet /> {/\* Nested routes will be rendered here \*/}

</div>

);

}

**<Component />**:

1. Directly renders a component.
2. Not used for nested routing but for normal routing

import { Route } from 'react-router-dom';

function App() {

return (

<Routes>

<Route path="/" element={<Home />} />

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

</Routes>

);

}

## react. memo vs use memo

**React.memo:**

React.memo is a higher-order component (HOC) used to optimize functional components. It helps to prevent unnecessary re-renders by memoizing the component. When a component is wrapped in React.memo, React will only re-render it if its props have changed.

import React from 'react';

const MyComponent = ({ value }) => {

console.log('Rendering MyComponent');

return <div>{value}</div>;

};

export default React.memo(MyComponent);

In this example:

* MyComponent will only re-render if its value prop changes.
* If the parent component re-renders but the value prop remains the same, MyComponent will not re-render.

**useMemo :**

useMemo is a React hook that memoizes the result of a function. It helps to optimize performance by caching the result of expensive computations and only recomputing when dependencies change.

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

function ExpensiveComponent({ number }) {

const computeFactorial = (n) => {

console.log('Computing factorial');

return n <= 1 ? 1 : n \* computeFactorial(n - 1);

};

const factorial = useMemo(() => computeFactorial(number), [number]);

return <div>Factorial of {number} is {factorial}</div>;

}

function App() {

const [number, setNumber] = useState(1);

return (

<div>

<input

type="number"

value={number}

onChange={(e) => setNumber(parseInt(e.target.value))}

/>

<ExpensiveComponent number={number} />

</div>

);

}

export default App;

In this example:

* computeFactorial is an expensive function.
* useMemo memoizes the result of computeFactorial and only recomputes it when the number dependency changes.
* This prevents unnecessary re-execution of computeFactorial when number has not changed.

### **When to Use Which**

1. **React.memo**:
   1. Use it to memoize a whole component.
   2. Helps to avoid re-rendering of the entire component when props haven't changed.
   3. Best for components that render frequently with the same props.
2. **useMemo**:
   1. Use it to memoize the result of a computation inside a component.
   2. Helps to avoid re-executing expensive functions when dependencies haven't changed.
   3. Best for optimizing costly calculations or expensive operations that are executed during render.

## How do you safeguard your reactjs application ?

1. By Mitigating Cross site scripting (XSS)

2. Will Prevent Cross-site Request Forgery(CSRF)

3. Secure/sanitize the user inputs with libraries like DOMPurify.

4. Regularly updating the dependencies and 3rd party components.

5. Good authentication and authorization mechanisms - JWTs for session management

6. Ensuring secure communication using HTTPS

7. Proper CORS configuration

8. Implementing Content security policy.

## Condition style :

export default function LoginModal() {

// Function to apply conditional styles based on the state

const getModalStyle = () => {

if (login || signup) {

return {

background: '#fff',

};

};

return (

<div style={getModalStyle()}>

{/\* Rest of your modal content \*/}

</div>

);

}

## Conditionally add a class :

If only have conditional class :

const MyComponent = ({ condition }) => {

return (

<div className={condition ? 'class1' : 'class2'}>

{/\* Content \*/}

</div>

);

};

If add conditional classes with other classes :

const MyComponent = ({ condition }) => {

return (

<div className={`base-class ${condition ? 'class1' : 'class2'} class3`}>

{/\* Content \*/}

</div>

);

};

## MouseHover event in react :

<ul>

<li

onMouseEnter={handleMouseEnter}

onMouseLeave={handleMouseLeave}

</li>

</ul>

## Get the position of element :

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

function MyComponent() {

const elementRef = useRef(null);

useEffect(() => {

if (elementRef.current) {

const rect = elementRef.current.getBoundingClientRect();

console.log("Element's position from the top of the page:", rect.top);

console.log("Element's position from the left of the page:", rect.left);

}

}, []);

return <div ref={elementRef}>Your Element</div>;

}

export default MyComponent;

## Redux vs redux toolkit :

| Redux | 1. redux toolkit |
| --- | --- |
| We create a store using createStore() | Use StoreConfiguration |
| We didn’t use slice | Use Slice with name,initialState and reducers. |
| Here, we mention reducer in switch case. | Here, we use it inside slice (reducers property), so no need to use switch case. |
|  |  |

## What is Redux, and why is it used in React applications?

**Redux** is a predictable state management library commonly used with React applications. It helps manage the state of an application in a predictable way, making it easier to debug and maintain. Redux is particularly useful in large applications where managing state across multiple components can become complex.

## Micro frontend

**Command :** **npx create-mf-app**

## 🌲 Explain the core principles of Redux (Actions, Reducers, Store).

**Core Principles of Redux:**

1. **Single Source of Truth**: The state of the entire application is stored in a single object called the **store**.
2. **State is Read-Only**: The only way to change the state is to dispatch an action, an object describing what happened.
3. **Changes are Made with Pure Functions**: To specify how the state tree is transformed by actions, you write pure functions called **reducers**.

## 🌲 What is a Redux Store? How is it different from React component state?

**Redux Store**:

* The store is a single object that holds the entire state of the application.
* It provides methods to get the current state, dispatch actions, and subscribe to changes.
* It's a global state management system, independent of the React component tree.

**React Component State**:

1. State is local to the component in which it is declared.
2. Each component can manage its own state and pass it down as props to child components.
3. React's component state is not inherently shared or globally accessible without additional mechanisms.

## 🌲 Describe the flow of data in a Redux application.

**Dispatching an Action**: The application dispatches an action. An action is a plain JavaScript object that describes what happened (e.g., { type: 'INCREMENT' }).

**Reducer**: The action is sent to the reducer, a pure function that takes the current state and action as arguments and returns a new state.

**Store**: The reducer updates the store with the new state.

**State Update**: The application reads the new state from the store and re-renders any affected components.

## 🌲 Why is immutability important in Redux, and how is it achieved?

**Immutability** ensures that state changes are predictable and traceable, making debugging easier. In Redux:

* **Immutability** allows for efficient change detection in the state, improving performance.
* **Immutability** ensures the original state is never modified directly, preventing side effects.

**How it's Achieved**:

1. Use immutable data structures (e.g., arrays, objects).
2. Use spread operator or methods like Object.assign to create copies of state.

const initialState = { count: 0 };

function counterReducer(state = initialState, action) {

switch (action.type) {

case 'INCREMENT':

return { ...state, count: state.count + 1 };

default:

return state;

}

}

## 🌲 What are Redux Actions and Action Creators?

**Redux Actions**:

1. Actions are plain JavaScript objects that represent an intention to change the state.
2. Actions must have a type property that indicates the type of action being performed.

Example of an action:

const incrementAction = { type: 'INCREMENT' };

**Action Creators**:

* Action creators are functions that create and return action objects.
* They encapsulate the process of creating actions and can also include additional payload data.

Example of an action creator

function increment() {

return { type: 'INCREMENT' };

}

function addTodo(text) {

return { type: 'ADD\_TODO', payload: text };

}

## 🌲 Explain the role of Reducers in Redux.

**Reducers** are pure functions that take the current state and an action as arguments and return a new state. They specify how the application's state changes in response to actions sent to the store. The reducer is the heart of Redux state management.

1. **Pure Function**: Reducers must be pure functions, meaning they should always return the same output for the same input and produce no side effects.
2. **Action Handling**: They handle different action types using a switch statement or conditional logic.
3. **State Initialization**: Reducers often provide initial state for the application.

const initialState = { count: 0 };

function counterReducer(state = initialState, action) {

switch (action.type) {

case 'INCREMENT':

return { ...state, count: state.count + 1 };

case 'DECREMENT':

return { ...state, count: state.count - 1 };

default:

return state;

}

}

## 🌲 What is a Redux Selector? Why and when would you use it?

**Redux Selectors** are functions that extract specific pieces of data from the Redux store state. They help to encapsulate the logic for deriving data from the state, making it reusable and easier to maintain.

1. **Encapsulation**: They keep the state shape consistent and abstract away the details of state structure.
2. **Performance**: Selectors can be memoized using libraries like Reselect to optimize performance by avoiding unnecessary recalculations.

const getCount = (state) => state.counter.count;

**Memoized Selector Example with Reselect**:

import { createSelector } from 'reselect';

const getCount = (state) => state.counter.count;

const getDoubleCount = createSelector(

[getCount],

(count) => count \* 2

);

## 🌲 Discuss the difference between Redux Thunk and Redux Saga for handling async actions.

**Redux Thunk**:

1. **Function Middleware**: Allows you to write action creators that return a function instead of an action.
2. **Simple**: Good for basic asynchronous logic like simple API calls.

const fetchData = () => {

return async (dispatch) => {

const response = await fetch('/api/data');

const data = await response.json();

dispatch({ type: 'FETCH\_DATA\_SUCCESS', payload: data });

};

};

**Redux Saga**:

1. **Generator Functions**: Uses ES6 generators to manage complex asynchronous workflows.
2. **More Powerful**: Handles complex side effects like concurrency, race conditions, and more.

import { call, put, takeEvery } from 'redux-saga/effects';

function\* fetchData() {

try {

const response = yield call(fetch, '/api/data');

const data = yield response.json();

yield put({ type: 'FETCH\_DATA\_SUCCESS', payload: data });

} catch (error) {

yield put({ type: 'FETCH\_DATA\_FAILURE', error });

}

}

function\* mySaga() {

yield takeEvery('FETCH\_DATA\_REQUEST', fetchData);

}

## 🌲 What is a Redux Middleware? Provide examples of commonly used middlewares.

**Redux Middleware**:

* Middleware provides a third-party extension point between dispatching an action and the moment it reaches the reducer.
* It can intercept actions, perform side effects, and dispatch new actions.

**Common Middlewares**:

1. **redux-thunk**: Allows action creators to return functions for handling async logic.
2. **redux-saga**: Uses generator functions to handle complex asynchronous workflows.
3. **redux-logger**: Logs actions and state changes to the console for debugging purposes.

import { createStore, applyMiddleware } from 'redux';

import thunk from 'redux-thunk';

const store = createStore(rootReducer, applyMiddleware(thunk));

## 🌲 Explain the purpose of a middleware like redux-thunk in Redux.

**redux-thunk**:

1. **Async Logic**: Enables action creators to return functions instead of actions, allowing for asynchronous operations such as API calls within the action creators.
2. **Dispatch Multiple Actions**: The returned function can dispatch multiple actions, useful for handling success and failure states of async operations.
3. **Access to State**: The function receives dispatch and getState as arguments, providing access to the state and dispatch capabilities for more complex logic.

const fetchData = () => {

return async (dispatch, getState) => {

dispatch({ type: 'FETCH\_DATA\_REQUEST' });

try {

const response = await fetch('/api/data');

const data = await response.json();

dispatch({ type: 'FETCH\_DATA\_SUCCESS', payload: data });

} catch (error) {

dispatch({ type: 'FETCH\_DATA\_FAILURE', error });

}

};

};

In this example, redux-thunk allows the fetchData action creator to handle asynchronous API calls, dispatching different actions based on the outcome of the API call.

1. 🌲 How does a middleware handle actions in the Redux flow?
2. 🌲 What are the main features of Redux Toolkit?
3. 🌲 Explain the concept of 'slices' in Redux Toolkit.
4. 🌲 Why might someone prefer using RTK over traditional Redux?
5. 🌲 Discuss the benefits of using createSlice() in Redux Toolkit.
6. 🌲 How does RTK simplify the Redux setup compared to traditional Redux?
7. 🌲 Discuss the concept of Redux state normalization and its advantages.
8. 🌲 Explain the idea of time-travel debugging with Redux DevTools.
9. 🌲 How does memoization play a role in Redux selectors for performance optimization?
10. 🌲 Discuss the potential drawbacks of overusing Redux in small to medium-sized applications.
11. 🌲 Explain the differences between action creators and action types in Redux.
12. <https://www.linkedin.com/posts/alpnap_frontend-react-seo-activity-7220990999194460162-7BtP?utm_source=share&utm_medium=member_android>

## What are the major new features introduced in React 19?

## Super inside constructor

In React, the super keyword is used in class components to call the constructor of the parent class, React.Component. This is necessary when you want to use this in the constructor, as it initializes the parent class and makes this available.

import React, { Component } from 'react';

class MyComponent extends Component {

constructor(props) {

super(props); // Call the constructor of React.Component

this.state = {

// Initialize state

count: 0,

};

// Binding methods (if necessary)

this.incrementCount = this.incrementCount.bind(this);

}

incrementCount() {

this.setState((prevState) => ({

count: prevState.count + 1,

}));

}

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.incrementCount}>Increment</button>

</div>

);

}

}

export default MyComponent;

In this example:

1. **Constructor Definition**: The constructor is defined to initialize the component.
2. **Calling super(props)**: The super(props) call is necessary to call the constructor of React.Component with props. This is required to set up the component correctly, especially when using this.props.
3. **Initializing State**: The component's state is initialized using this.state = { count: 0 };.
4. **Binding Methods**: If methods need to access this, they can be bound in the constructor (though using arrow functions is more common in modern React to avoid binding).
5. <https://www.linkedin.com/posts/shubhamsingh15_testing-in-reactjs-ugcPost-7226462360620646400-nh4F?utm_source=share&utm_medium=member_android> - JEST
6. fdsafafa