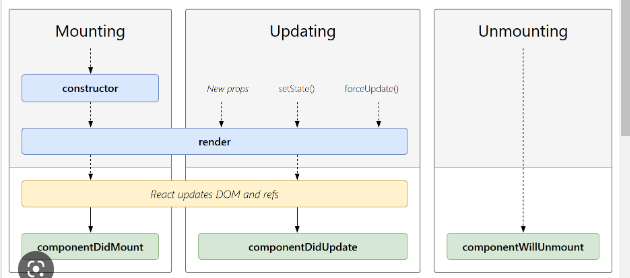
1 - Review the job description and requirements: Make sure you understand the specific skills, qualifications, and experience required for the job. Then, tailor your responses to demonstrate how you meet those requirements.

2 - Research the company: Familiarize yourself with the company's products, services, and culture. You can use this information to show your enthusiasm and alignment with the company's mission and values.

3 - Brush up on technical skills: Review the technical skills required for the job and practice your coding and problem-solving abilities. You can use online resources such as coding challenges or online courses to practice.

* What is JSX? Is a syntax extension of JS that allows to write HTML in JS. Is a syntactic sugar over React.createElement
* Why is class, classname in React? Class is a reserved keywork in JS, so it will be a conflict if use this work
* Describe Data Flow in React? The data flow ins unidirectional. This means that all of the components in React have a parent-child relationship, passing data down from above. The most basic way to pass information down is through Props. A good option if you have a prop drilling problem, for example if you want to share data multiple layers deep is to use Context, Global State or Redux.
* How would you delay an API call until a component has mounted? In a Class would be using componentDidMount( ), which is a function that runs after the component is mounted that allows you to run other code after that point. This will make sure that you are not making your API call before the component has mounted.

How the component lifecycle works:



Another way to do that is using the useEffect Hook, passing an empty array as the second argument which will mimics the componentDidMount. Inside of the useEffect we can write the API call and it will work. Now that we have functional components and hooks, we don’t need to write classes anymore.

* Should you use Ternaries or && operator to conditionally render react components? We should use Ternaries because if you use the && Operator and you are checking for example the length of an array and want to render things out, it will return 0 if the array starts with 0.

4 - Be prepared to talk about your experience: Prepare specific examples of projects you have worked on, technical challenges you have faced, and how you overcame them. Use the STAR (Situation, Task, Action, Result) method to structure your responses.

* I work as a main developer, developing new features of the system and maintaining the code. I have as highlights, the development of a homologation environment, in which there wasn’t before generating many problems in the production environment. I also optimized analysts' processes where they performed extensive manual tasks, and now they are all performed automatically.
* Before that I worked as a freelancer developing front-end pages with React.JS and Redux.

5 - Practice common interview questions: Review common IT interview questions and practice your responses. You can find these questions online or from career services centers.

* Tell me about yourself and your background.
* What technical skills do you possess that are relevant to this role?
* Can you explain your experience with [specific programming language, software, or technology?
* How do you keep up-to-date with the latest industry trends and technologies?
* Can you walk me through a project you worked on and your role in it?
* Have you worked in a team environment before? How did you contribute to the team's success?
* How do you troubleshoot technical issues?
* Can you explain the process you follow when you encounter a problem you don't know how to solve?
* Can you discuss your experience with database management systems?

6 - Show your enthusiasm and passion: IT is a constantly evolving field, and interviewers look for candidates who are passionate about technology. Show your interest by discussing technology trends, the latest tools, or industry events.

7 - Ask questions: Don't be afraid to ask questions about the company, the team, or the job. It shows your interest in the position and can also help you determine if it's the right fit for you.

8 – Which are the main React Concepts?

* Components: React allows to create reusable UI components that can be used in multiple parts of an application. Components are the building blocks of a React Application.
* JSX: Is a syntax extension of JS that allows to write HTML in JS. It allows to create React components in a familiar and concise syntax.
* State: Represents the data that is manage by a component. When state changes, react re-renders the component to reflect the updated state.
* Props: Are used to pass data and configuration between components. Props allow to create reusable components that can be easily customized.
* Virtual DOM: React uses a virtual representation of the actual DOM to improve performance. The virtual DOM is a lightweight copy of the actual DOM and allows React to efficiently update the DOM when there are changes.
* Lifecycle methods: React components have several lifecycles methods that are called at different points in the component’s lifecycle. These methods allow developers to perform actions at specific points in the component’s lifecycle, such as when a component is mounted or updated.
* Events: React supports a range of DOM events, such as onClick, onChange and onSubmit. These events can be used to trigger actions in a React component.
* Hooks: They are a new addition to React that allow to use state and other React features in functional components. Hooks enable to write more concise and reusable code.
* React Router: Is a popular library for implementing routing in a React application. It allows to define different routes in an application and handle navigation between them.

9 - What is React Redux?

* Is a state management library. It is based on the Flux architecture pattern and provides a predictable way of managing the application state.
* The state is stored in a central location called “store”. The store is created using the Redux Library and contains the state of the application. The store can be accessed by any component and can be update using “actions”.
* Actions are plain JS objects that describe the type of event that has occurred in the application. When an action is dispatched, it is sent to the store and the store updates its state based on the action.
* Reducers are functions that update the state of the application based on the actions that are dispatched. Reducers take the current state and an action as input and return a new state.
* In a React Redux application, components can subscribe to changes in the store using the "connect" function. The "connect" function is a higher-order component that connects a React component to the Redux store. When the state of the store changes, the connected component will be re-rendered with the updated state.
* React Redux provides a number of benefits for managing state in a React application. It makes it easier to reason about the state of the application, provides a predictable way of updating the state, and enables components to subscribe to changes in the state without having to pass props down through multiple levels of the component hierarchy.

11 – Qual a diferença entre DOM e Virtual DOM?

* O DOM é a representação em tempo real da estrutura do documento HTML, que é criada pelo navegador quando a página é carregada. Ele é usado para manipular a estrutura, o conteúdo e o estilo da página em tempo real, através de linguagens de programação como JavaScript.
* Por outro lado, o Virtual DOM é uma técnica que envolve a criação de uma cópia virtual do DOM em memória, que é atualizada e comparada com a versão real do DOM para determinar quais partes da página precisam ser atualizadas. Isso ajuda a melhorar a performance da aplicação, pois a atualização do DOM é um processo custoso em termos de processamento.

12 – O que é o Event Loop do Node.JS?

* É um recurso do Node.js, que permite que ele seja capaz de lidar com um grande número de operações de entrada/saída (I/O) assíncronas e simultâneas sem bloquear a execução do código.
* É uma estrutura de loop infinito que verifica a fila de eventos do Node.js e executa as tarefas pela ordem que foram adicionadas a fila. Essas tarefas podem ser call-back functions, requisições HTTP, acesso a arquivos, conexões com banco de dados...
* Com esse recurso, o Node.js consegue lidar com uma grande carga de entradas/saídas simultâneas, mantendo uma alta velocidade de processamento e uma baixa latência.

13 – Quais são os fundamentos do Node.JS?

* Event-Driven Programming: Node.js utiliza uma arquitetura orientada a eventos, onde as ações são disparadas por eventos e não por um fluxo de controle linear.
* Módulos: Em Node.js, os módulos são blocos de código encapsulados, que podem ser usados para adicionar funcionalidades à aplicação ou para isolar diferentes partes do código.
* Fluxo de controle assíncrono: Node.js é projetado para lidar com muitas solicitações simultâneas de entrada/saída (I/O) e, portanto, utiliza um modelo de fluxo de controle assíncrono para garantir que o código não fique bloqueado enquanto aguarda uma resposta.
* Manipulação de arquivos, HTTP/HTTPS, Banco de dados, Gerenciamento de pacotes...

14 – Me explique sobre arquitetura de aplicações Node.JS e os padrões mais comuns?

* Geralmente é baseada em camadas, onde cada camada tem uma responsabilidade específica na aplicação.
  + Camada de infraestrutura: Responsável por lidar com a configuração do ambiente de aplicação, incluindo conexão com o banco de dados, configuração do servidor HTTP, gerenciamento de logs etc...
  + Camada de dados: Lida com a comunicação com o banco de dados, incluindo a criação e execução de consultas SQL ou operações CRUD.
  + Cama de negócios: Implementa a lógica de negócios da aplicação, como validação de dados, processamento de regras de negócio etc...
  + Camada de apresentação: Fornece a interface do usuário para aplicação, incluindo renderização HTML, resposta JSON etc...
* Os padrões mais comuns incluem:
  + MVC (model-view-controller): representa a estrutura dos dados da aplicação, a interface do usuário e a lógica de negócio.
  + Middleware: São funções intermediárias que são adicionadas à cadeia de execução para manipular a solicitação do usuário antes que seja enviada ao Controller.
  + Repositório: Utilizam bancos de dados, onde o repositório é responsável por lidar com a comunicação com o banco de dados e fornecer uma interface para o restante da aplicação.
  + Serviços: Aqui a lógica de negócio é encapsulada em um serviço, que é responsável por fornecer uma interface para a aplicação. Isso permite que a lógica de negócio seja reutilizada em diferentes partes da aplicação.
  + Injeção de dependências: As dependências são injetadas na aplicação em tempo de execução, em vez de serem definidas no código. Permite maior flexibilidade e facilidade na configuração da aplicação.