UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE CIÊNCIAS EXATAS E TECNOLOGIA DA INFORMAÇÃO
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Gabriel Barbosa dos Santos

Desenvolvimento web com React: Principais funcionalidades, limitações e bibliotecas auxiliares mais utilizadas

ANGICOS
2024

Gabriel Barbosa dos Santos

Desenvolvimento web com React: Principais funcionalidades, limitações e bibliotecas auxiliares mais utilizadas

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Sistemas de Informação.

Orientadora: Valquíria Melo Souza Correia, Prof.ª Dr.ª

ANGICOS

2024

S237d Santos, Gabriel Barbosa dos.
Desenvolvimento web com React: Principais funcionalidades, limitações e bibliotecas auxiliares mais utilizadas / Gabriel Barbosa dos Santos. - 2024.
45 f. : il.

Orientadora: Valquíria Melo Souza Correia.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Sistemas de Informação, 2024.

1. React. 2. Desenvolvimento. 3. Enquetes. 4. Ferramentas. I. Correia, Valquíria Melo Souza, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo autor(a).
Biblioteca Campus Angicos
Bibliotecário: Helder Romero Maia Duarte
CRB: 15/673

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

Gabriel Barbosa dos Santos

Desenvolvimento web com React: Principais funcionalidades, limitações e bibliotecas auxiliares mais utilizadas

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Sistemas de Informação.

Defendida em: 10 / 04 / 2024

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
VALQUIRIA MELO SOUZA CORREIA
Data: 22/04/2024 13:53:38-0300
Verifique em https://validar.iti.gov.br

Valquíria Melo Souza Correia, Prof.ª Dr.ª (UFERSA)
Presidente

Documento assinado digitalmente
gov.br
ADRIANA MARA GUIMARAES DE FARIAS
Data: 22/04/2024 08:10:43-0300
Verifique em https://validar.iti.gov.br

Adriana Mara Guimarães de Farias, Prof.ª Dr.ª (UFERSA)
Membro Examinador

Documento assinado digitalmente
gov.br
SAIRO RAONI DOS SANTOS
Data: 20/04/2024 13:52:16-0300
Verifique em https://validar.iti.gov.br

Sairo Raoni dos Santos, Prof. Dr. (UFERSA)
Membro Examinador

RESUMO

O React é a ferramenta de construção de interfaces de usuário mais utilizada no desenvolvimento *front-end* moderno. A biblioteca se destacou por sua ideia de componentização e facilidade de adoção em relação aos seus concorrentes. Sua popularidade é refletida no tamanho de sua comunidade, bastante ativa, ela produz bibliotecas e pacotes de ferramentas que auxiliam em tarefas importantes. O React também conta com uma equipe que trabalha em atualizações que trazem melhorias e inovações por meio de novas funcionalidades. No entanto, existe o lado negativo em ter um grande acervo de ferramentas e atualizações frequentes: os desenvolvedores sofrem com dúvidas na escolha de ferramentas e podem facilmente ficar para trás em relação à última leva de atualizações. Com esses problemas em mente, o presente trabalho explora as ferramentas mais utilizadas no mercado e traz uma síntese acerca das principais capacidades e limitações da biblioteca em sua versão 18.2.0. Os resultados surgem a partir de pesquisas utilizando as documentações oficiais do React e das principais ferramentas encontradas em enquetes entre desenvolvedores.

Palavras-chave: React; desenvolvimento; enquetes; ferramentas.

ABSTRACT

React is the most widely used user interface building tool in modern front-end development. The library has stood out for its idea of componentization and ease of adoption compared to its competitors. Its popularity is reflected in the size of its highly active community, which produces libraries and tool packages that assist in important tasks. React also benefits from a team that works on updates, bringing improvements and innovations through new features. However, there is a downside to having a large arsenal of tools and frequent updates: developers struggle with doubts in choosing tools and can easily fall behind in the latest wave of updates. With these issues in mind, the present work explores the most used tools in the market and provides a synthesis of the main capabilities and limitations of the library in its version 18.2.0. The results stem from research using React's official documentation and the main tools found in surveys among developers.

Keywords: React; development; surveys; tools.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Várias plataformas de desenvolvimento web baseadas em JavaScript surgiram graças ao ganho de desempenho proporcionado pelo lançamento da V8 Engine, feito pelo Google em 2008 (Xing; Huang; Lai, 2019). Entre elas, a biblioteca React é a escolha mais popular para a criação da interface de usuários, ocupando o topo das enquetes feitas entre desenvolvedores, como a 2023 *Developer Survey* da Stack Overflow, onde a biblioteca ficou em primeiro lugar entre todas as tecnologias de desenvolvimento web, com 42,87% de uso (Stack Overflow, 2023).

A facilidade de adoção, podendo ser utilizada em partes de projetos que já estão em andamento sem comprometer sua integridade (Clark *et al*, 2021), e sua baixa curva de aprendizado em relação a seus concorrentes (Rawat; Mahajan, 2020) são alguns dos pontos-chave para a popularização do React. Entretanto, sua popularidade trouxe alguns problemas. Um deles é a dificuldade de encontrar especialistas na ferramenta, conforme demonstrado no trabalho de Ferreira, Borges e Valente (2021), sendo este o principal ponto fraco do React, segundo as respostas dos desenvolvedores entrevistados em sua pesquisa.

Devido ao tamanho de sua comunidade, o React dispõe de um enorme acervo de ferramentas que resolvem desde pequenos até grandes problemas de desenvolvimento. No entanto, essa quantidade de opções pode gerar dúvidas nas decisões tecnológicas de um projeto ou durante o processo de aprendizado. Segundo Vargas *et al*. (2020), a seleção da biblioteca certa é uma tarefa exigente para os desenvolvedores, dado o grande número de fatores que devem ser considerados durante o processo de escolha.

Para mitigar tais dificuldades, este trabalho visa auxiliar os desenvolvedores React a compreender melhor as capacidades de sua plataforma de trabalho, promovendo também o conhecimento de suas limitações e fornecendo um melhor entendimento das ferramentas externas mais utilizadas pelo mercado de desenvolvimento.

1.2 Objetivos

1.2.1 Objetivo Geral

Elaborar um quadro com recomendações tecnológicas baseado nas limitações e ferramentas mais utilizadas no desenvolvimento com React.

1.2.2 Objetivos Específicos

- Compreender as principais funcionalidades do React;
- Identificar as limitações do React;
- Identificar pontos de escolha tecnológica (PET) no desenvolvimento React e suas ferramentas mais utilizadas;
- Estudar as ferramentas identificadas;
- Elaborar recomendações para cada PET e limitação do React com considerando as ferramentas estudadas.

2 REVISÃO BIBLIOGRÁFICA

Esse capítulo contém uma revisão de literatura sobre evolução do front-end web, passando pela criação de cada tecnologia base até a chegada do React e também a revisão de trabalhos similares com uma análise das semelhanças e diferenças entre eles e o nosso.

2.1 Pilares Tecnológicos da web

De acordo com Xiaoshu (2020), o desenvolvimento web moderno é composto por três principais tecnologias: JavaScript, HTML e CSS. Juntas, elas são descritas por Levlin (2020) como os blocos fundamentais da construção de páginas web. Abaixo, forneceremos o contexto de criação de cada uma dessas tecnologias, juntamente com sua contribuição para o desenvolvimento web moderno.

2.1.1 Criação da web e HTML

Em 1990, Tim Berners-Lee criou o primeiro site da história da World Wide Web, ou simplesmente web. Este site pioneiro pretendia funcionar como um guia para a Internet nascente, fornecendo informações sobre os princípios fundamentais da web e instruções sobre como construir servidores e navegadores para acessá-la. O site era simples, composto principalmente de texto e hiperlinks, e foi criado utilizando uma das outras invenções de Berners-Lee, a Hypertext Markup Language, ou HTML (Raggett, 1998).

O HTML constitui a base de qualquer página da web, fornecendo a estrutura e a semântica necessárias para definir o conteúdo e o layout. Por meio das tags HTML, os desenvolvedores podem criar elementos como títulos, parágrafos, listas, tabelas e imagens incorporadas, que organizam informações baseadas em texto e multimídia. Além disso, o HTML possibilita a inclusão de outros elementos, como formulários, links e objetos de mídia, permitindo que os usuários interajam com a página e acessem recursos externos (W3Schools, 2024a).

2.1.2 Cascading Style Sheets

Cascading Style Sheets, ou CSS, foi concebido como uma resposta à necessidade de separar a estrutura do conteúdo (HTML) de sua apresentação (estilo). A primeira especificação CSS foi lançada em 1996 e, desde então, passou por várias revisões, culminando no CSS3. Cada nova versão trouxe recursos avançados, como posicionamento, sombras, animações e muito mais, permitindo maior criatividade e sofisticação no design web.

Usando CSS, os desenvolvedores podem aplicar estilos e formatação a elementos HTML, incluindo cores, fontes, layouts e animações. Essa separação entre conteúdo e apresentação permite maior flexibilidade no design de uma interface de usuário consistente e visualmente atraente em diversas páginas da web. Além disso, o CSS oferece suporte ao design responsivo, permitindo que os sites se adaptem e sejam exibidos corretamente em diferentes dispositivos e tamanhos de tela (W3Schools, 2024b).

2.1.3 JavaScript

JavaScript foi criado em 1995 por Brendan Eich, um programador que trabalhava para a Netscape Communications Corporation. A linguagem foi inicialmente chamada de Mocha e mais tarde renomeada para LiveScript antes de ser finalmente chamada de JavaScript. Seu principal objetivo era adicionar interatividade às páginas web, permitindo que os desenvolvedores web criassem sites mais dinâmicos e envolventes para os usuários (Rauschmayer, 2014, p. 43).

O JavaScript desempenha um papel crucial em tornar as páginas web interativas. Com sua capacidade de manipular e modificar o conteúdo das páginas web, o JavaScript permite que os desenvolvedores criem interfaces de usuário dinâmicas, lidem com interações do usuário e executem diversas operações do lado do cliente, como validação de formulários, efeitos de animação e manipulação de dados (MDN Web Docs, 2023).

2.1.4 Resumo

Através do uso de JavaScript, as páginas da web podem ser transformadas em experiências interativas para os usuários. Além disso, o HTML fornece a estrutura e a semântica necessárias para organizar e apresentar o conteúdo das

páginas da web, enquanto o CSS permite a customização e o estilo desses elementos.

2.2 Surgimento do React e seu impacto no desenvolvimento web

Desenvolvido pelo Facebook, o React é uma biblioteca JavaScript que capacita os desenvolvedores a construir interfaces de usuário para aplicações web. Criado inicialmente por Jordan Walke, engenheiro de software do Facebook, o React foi apresentado ao público pela primeira vez em 2013.

A arquitetura baseada em componentes do React permite aos desenvolvedores dividir os aplicativos em partes menores e reutilizáveis, promovendo a composição e modularidade do código. Esses componentes podem ser facilmente combinados para criar interfaces de usuário mais complexas (Cincović; Punt, 2020).

O React utiliza um DOM[1] virtual (uma cópia leve do DOM real) para otimizar a renderização. Isso permite que o React atualize eficientemente apenas as partes da UI que foram alteradas, resultando em melhor desempenho e uma experiência de usuário mais suave (Levlin, 2020, p. 45).

O foco na composição e modularidade do React promove as melhores práticas no design de componentes, incentivando os desenvolvedores a considerar como seus componentes serão escalonados e modificados conforme o aplicativo cresce. Além disso, o ecossistema do React é robusto, com uma vasta comunidade e uma variedade de pacotes e ferramentas de código aberto que aceleram a adoção da biblioteca e permitem aos desenvolvedores melhorar seus aplicativos.

O React não apenas transformou a maneira como os desenvolvedores abordam o desenvolvimento de interfaces de usuário, mas também elevou a qualidade e o desempenho das aplicações web, tornando-se uma ferramenta essencial no desenvolvimento de tecnologia web moderna.

2.3 Importância da tomada de decisão em projetos front-end

[1] O Modelo de Objeto de Documentos (do inglês DOM) é uma interface de programação que permite acesso dinâmico ao conteúdo, estrutura, e estilo de documentos de páginas da web HTML (LEVLIN, 2020).

O desenvolvimento de software é um campo complexo e dinâmico que requer tomadas de decisão cuidadosas em todas as etapas do processo. Um dos aspectos mais cruciais do desenvolvimento de software é a seleção das ferramentas adequadas para o trabalho. Bruckhaus (1996) afirma que as ferramentas apropriadas são uma maneira eficiente de aumentar a qualidade do desenvolvimento de software.

Boas decisões em relação à seleção de ferramentas podem melhorar a produtividade e eficiência, como demonstrado nos resultados do trabalho de Boehm (1984), e proporcionar um aumento na confiabilidade do produto (Copeland *et al*., 2000). Além disso, a seleção de ferramentas pode estar relacionada à redução de custos, pois o tempo de produção é reduzido com a melhoria da produtividade.

Em resumo, as ferramentas podem impactar positivamente a produtividade, qualidade, custos e confiabilidade do produto final de software. Portanto, é essencial tomar decisões informadas e estratégicas ao escolhê-las para cada fase do processo de desenvolvimento de software.

2.4 Revisão de estudos anteriores

Inicialmente, não foram encontrados trabalhos na literatura disponível que cobrissem todos os pontos deste estudo de uma só vez. Portanto, a estratégia de pesquisa adotada foi a de tratá-los separadamente. A pesquisa foi dividida em duas categorias principais: limitações do React e funcionalidades do React.

2.4.1 Limitações do React

Os trabalhos mais recentes sobre as limitações do React oferecem apenas uma abordagem superficial dessa questão. Bhalla, Garg e Singh (2020) mencionam brevemente a necessidade de ferramentas externas para completar projetos em React, devido à ênfase da biblioteca em questões relacionadas à interface do usuário. Eles também discutem as dificuldades de trabalhar com o código JSX (JavaScript e HTML em um mesmo arquivo) do React. Esses mesmos achados são corroborados por Maurya *et al*. (2023), que também destacam a dificuldade de manter-se atualizado com as constantes mudanças na biblioteca.

No entanto, ambos os trabalhos não se aprofundam na questão da necessidade de bibliotecas externas, nem especificam claramente quais problemas exigem o uso delas.

2.4.2 Funcionalidades do React

Ao estudar sobre as funcionalidades do React, nos deparamos com as distintas interpretações do que se entende por “funcionalidade”. Enquanto uns se debruçam mais nas fundações do funcionamento da biblioteca (baixo nível), como o trabalho de Komperla *et al.* (2022), outras trazem as funcionalidades mais próximas do que se é usado no dia-a-dia ao se trabalhar com o React (alto nível), assim como foca a seção de aprendizado da biblioteca oficial do React.

O objetivo do estudo sobre as funcionalidades do React desse trabalho é a de compreender as ferramentas programáticas que a biblioteca fornece nativamente. Porém, o entendimento dos dois tipos de funcionalidade (baixo e alto nível) traz uma visão mais ampla sobre o funcionamento do React e o que ele tem a oferecer.

2.4.3 Diferencial deste trabalho

Com a leitura da bibliografia disponível foi possível identificar a falta de aprofundamento nas limitações do React e pouca documentação sobre as funcionalidades programáticas (além da documentação oficial). Dito isso, esse trabalho se diferencia dos citados ao identificar quais pontos de limitação existem e quais soluções são empregadas para resolvê-los. Além disso, o trabalho discorre sobre cada capacidade programática que o React possui, estudando suas funcionalidades e casos de uso.

3 METODOLOGIA

3.1 Tipo de pesquisa

A pesquisa realizada tem caráter quantitativo, já que são principalmente analisados dados de pesquisas buscando a extração de percepções sobre os seus resultados. Além disso, a pesquisa bibliográfica foi principalmente realizada utilizados as documentações oficiais do React e das ferramentas encontradas através das enquetes selecionadas.

3.2 Coleta de dados

A coleta de dados foi feita de maneira a alcançar os objetivo específico deste trabalho. Em cada subseção abaixo temos os tipos de fontes utilizadas para cada e sua respectiva justificativa de escolha.

3.2.1 Compreender as principais funcionalidades do React e identificar suas limitações

A documentação oficial do React, hospedada no repositório do GitHub e acessível através do link para a página 'react.dev', é uma fonte de informação abrangente e confiável. Além de fornecer cursos para aprendizado da ferramenta, ela oferece referências detalhadas sobre suas funcionalidades e limitações. O conteúdo da documentação inclui definições claras, casos de uso, exemplos práticos e interativos, bem como comentários que facilitam a compreensão do 'idioma' do React. Por ser uma fonte oficial e constantemente atualizada, a documentação do React foi considerada suficiente como a única fonte de informação para alcançar os objetivos desta subseção.

3.2.2 Identificar pontos de escolha tecnológica no desenvolvimento React e suas ferramentas mais utilizadas

Os pontos de escolha tecnológica, ou PET, são momentos no desenvolvimento web com React nos quais os desenvolvedores recorrem a

ferramentas externas para auxiliar no trabalho, seja por uma limitação do React ou pelas funcionalidades que aquela ferramenta possui.

Optamos por utilizar enquetes de mercado realizadas entre desenvolvedores web para identificar as principais ferramentas e bibliotecas mais utilizadas no contexto do React. Essa escolha se justifica pelas percepções valiosas que essas enquetes proporcionam, revelando as tecnologias mais adotadas e populares no mercado atual. Ao analisarmos as ferramentas nomeadas e classificadas por porcentagem de uso, podemos agrupá-las conforme o tipo de solução que oferecem. Isso nos permite identificar áreas específicas onde a assistência de ferramentas externas pode ser necessária durante o desenvolvimento com React.

Para a seleção de enquetes, adotamos um conjunto de critérios. Consideramos apenas aquelas que não fossem mais antigas do que 2 anos, garantindo assim a contemporaneidade dos dados. Além disso, priorizamos enquetes com uma quantidade significativa de respondentes, na casa dos milhares, para garantir uma amostragem ampla e representativa. A relevância das perguntas para esta pesquisa também foi um ponto-chave, buscando enquetes que abordassem especificamente bibliotecas e ferramentas utilizadas no desenvolvimento web.

Outro critério importante foi a credibilidade da organização responsável pela enquete, priorizando aquelas conduzidas por entidades reconhecidas no segmento ou patrocinadas por referências do setor. Além disso, aplicamos filtragens nas enquetes para considerar apenas as respostas de desenvolvedores exclusivos da plataforma React e seus frameworks relacionados. Isso garante que os resultados contenham apenas as ferramentas utilizadas em projetos com essa plataforma.

A pesquisa pelas enquetes foi realizada utilizando o mecanismo de busca do Google, utilizando termos como 'Front-end', 'Survey' e 'web technologies'.

3.2.3 Estudar as ferramentas identificadas

As ferramentas consideradas foram aquelas que apareceram em comum nos resultados das enquetes selecionadas, buscando minimizar qualquer viés e aumentar a confiabilidade dos nossos achados, além daquelas que foram utilizadas por mais de 10% dos entrevistados.

Apenas as documentações oficiais de cada ferramenta identificada nas enquetes foram utilizadas, seguindo a mesma justificativa aplicada para a coleta de dados do React, conforme descrito na subseção 3.2.1.

Para encontrar as documentações oficiais, recorremos ao GitHub, o maior repositório de código-fonte aberto do mundo, onde realizamos uma pesquisa pelo nome de cada ferramenta e filtramos os resultados com base na quantidade de estrelas. Uma vez identificados os repositórios oficiais, procuramos por links que remetessem aos sites das ferramentas, onde normalmente encontramos a documentação necessária. Em casos em que os repositórios não estavam disponíveis no GitHub, ou para verificar a validade dos sites encontrados, utilizamos os links fornecidos pelas enquetes ou conduzimos uma busca no Google com os termos “nome da ferramenta” + “docs”.

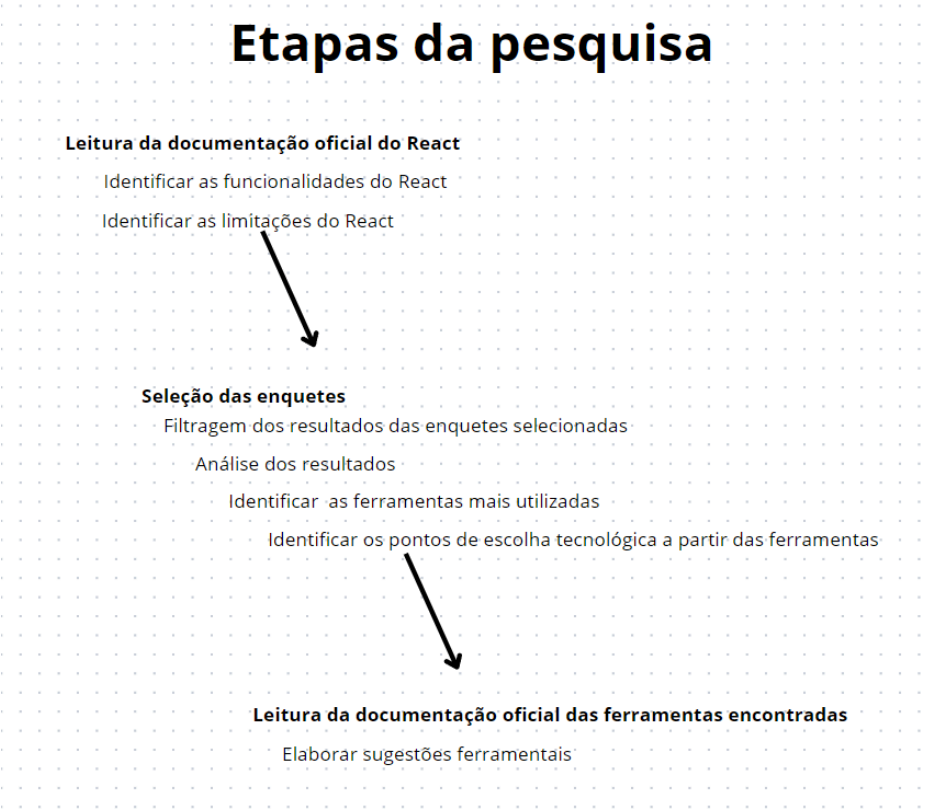
3.2.4 Elaborar recomendações para cada PET e limitação do React encontradas com base nas ferramentas estudadas

Para elaborar as recomendações, utilizamos nossa compreensão das capacidades do React, conforme descrito na subseção 3.2.1, juntamente com o conhecimento das ferramentas exploradas na subseção 3.2.3.

3.3 Etapas da pesquisa

Abaixo está uma imagem com o fluxo da pesquisa:

Figura 1 - Etapas da pesquisa



Fonte - Autoria própria (2024)

4 RESULTADOS E DISCUSSÕES

Os resultados da pesquisa foram divididos em quatro subseções: a primeira aborda as principais funcionalidades do React, a segunda identifica suas limitações, PET e ferramentas, a terceira explora as ferramentas mais utilizadas e faz recomendações com base nos achados, e por fim, a última apresenta um quadro resumindo os resultados.

4.1 Principais funcionalidades do React no desenvolvimento web

Com base na leitura da documentação oficial do React, identificamos quatro categorias de funcionalidades programáticas. Dessas, selecionamos apenas três - Hooks, Components e APIs - para esta subseção, pois a quarta categoria, Directives, continua em fase de testes (na versão 18.2.0 do React) e não está pronta para uso em produção.

4.1.1 *Hooks*

Hooks são ferramentas que permitem a utilização das funcionalidades do React em seus componentes. Abaixo, apresentamos um quadro com os principais hooks oferecidos pelo React, seus propósitos, exemplos de caso de uso e notas sobre sua utilização.

Quadro 1 - *Hooks* do React

Hook	**Propósito**	**Exemplos de caso de uso**	**Nota**
useState	Criar um *state* no componente atual. *state* é variável que salva valores entre re-renderizações.	Salvar algum valor e atualizar a UI quando esse valor for alterado. Exemplo: mostrar em tela a quantidade de cliques feitas em um botão.	Nem todos os valores exibidos em tela precisam ser *state*, para saber mais, leia: https://react.dev/learn/state-a-components-memory.
useReducer	Criar um *reducer* no componente atual. *reducer* é uma variável	Útil quando há muitas operações em cima de um *state*.	Um *state* é convertível em um *reducer* e vice-versa, vai da preferência do

	que salva valores entre re-renderizações e define como esses valores devem ser atualizados em uma função que pode ser separada do componente.	Ao separar a lógica de atualização do componente com o *state*, o código se torna mais legível e fácil de identificar erros.	desenvolvedor utilizar um ou outro.
useContext	Ler e se manter atualizado quanto ao valor do *context* passado. Um *context* permite que um componente "pai" passe dados para componentes "filhos" sem o uso de *props*. Mais explicações sobre *context* na subseção de *API*s.	Obter um valor de algum componente parente distante, evitando o *prop drilling*, que seria ter que passar esse valor de componente em componente, por meio de *props,* até chegar no componente que o utilizará. Pode ser utilizado juntamente com a *context API* para a criação de variáveis globais.	O *useContext* surgiu na versão 16.8 do React para resolver o problema muito comum de *prop drilling, q*ue antes era comumente resolvido com uso de ferramentas externas.
useRef	Referenciar um valor que não precisa ser mostrado em tela, mas que não pode ser reiniciado a cada nova renderização.	Referenciar um *interval* para que ele possa ser removido posteriormente. Referenciar um elemento HTML para poder ter acesso às suas propriedades.	Uma *ref* é uma mistura de uma variável comum e um *state*, ela mantém seu valor entre renderizações, como a última, mas não aciona uma nova quando é alterada, como a primeira.
useEffect	Obter informação de um sistema que não é controlado pelo React.	Gerenciar um temporizador com os métodos do JavaScript, Obter dados da janela do navegador ou realizar ações por meio da *API* de uma biblioteca externa.	Apesar de poder ser utilizado para fazer o *fetch* de dados, o *useEffect* sozinho não é a melhor opção. O próprio React sugere a utilização de ferramentas auxiliares para esta atividade ou uma implementação mais

			robusta em conjunto com esse *hook*.
useMemo	Evitar que cálculos sejam refeitos desnecessariamente durante novas renderizações.	Guardar o resultado de uma filtragem em um *array* com muitas posições. Evitar que um componente filho sofra uma nova renderização quando ocorrer alguma alteração no seu componente pai.	O *useMemo* é um *hook* de otimização, seu uso pode ser geralmente recomendado em cálculos que demoram mais de 1 milissegundo ou a depender das suas necessidades específicas de performance.
useCallback	Evitar que funções sejam declaradas novamente em uma nova renderização.	Impedir renderizações desnecessárias ao passar funções como *props* para componentes filhos.	Assim como o *useMemo* o *useCallback* também é um *hook* de otimização, ambos salvam dados em cache para evitar retrabalho (um o resultado, o outro a declaração) e muitas vezes são utilizados juntos na otimização de componentes.

Fonte: React (2024)

Além dos *hooks* mencionados anteriormente e daqueles que estão em fase de testes, existem outros, como o *useTransition, useLayoutEffect* e *useDebugValue*. O próprio React os descreve como “[…] variações raramente usadas […]” ou “[…] não são normalmente usados no código da aplicação.” (React, 2024a, tradução nossa). Por essa razão, optamos por não os incluir no quadro.

4.1.2 *Components*

O React possui três componentes embutidos, cada um servindo para um propósito específico:

1. *Fragments*: Um *Fragment* (<>…</> ou <Fragment>…</Fragment>) permite agrupar vários elementos juntos sem introduzir um nó DOM adicional. É útil quando é necessário retornar vários elementos de uma função sem envolvê-los em um elemento pai desnecessário, como uma div (...);
2. *Profiler*: O *Profiler* (<Profiler />) é utilizado para medir o desempenho de componentes React. Ao envolver uma seção da sua árvore de componentes com ele, você pode analisar quais partes estão demorando mais para renderizar, tornando a identificação de gargalos no desempenho mais rápida e precisa;
3. *Suspense*: O componente *Suspense* (<Suspense />) ajuda a gerenciar estados de carregamento. Ao envolver componentes que podem levar algum tempo para carregar (por exemplo, buscar dados de uma API) com o *Suspense*, você pode exibir uma UI de carregamento (texto ou animações) enquanto o conteúdo está sendo obtido. Isso melhora a experiência do usuário, evitando que a página inteira fique em branco durante o processo de carregamento.

4.1.3 APIs

O React possui algumas APIs que facilitam a criação de componentes e resolvem alguns problemas comuns enfrentados no desenvolvimento da UI. Abaixo, estão resumidas as funcionalidades de cada uma dessas APIs:

1. *createContext*: esta API possibilita resolver o problema de *prop drilling* criando um canal para transmitir valores para outros componentes da aplicação, independentemente de estarem no mesmo arquivo ou não. Em outras palavras, ela cria um comportamento semelhante ao de variáveis globais;
2. *forwardRef*: permite passar *refs* para componentes filhos, mesmo que sejam funções. É útil em situações em que é necessário acessar o DOM diretamente de um componente pai ou interagir com bibliotecas de terceiros que exigem *refs*;
3. *lazy*: possibilita o carregamento dinâmico de importações, fazendo com que o código do item importado seja carregado apenas quando necessário, como

quando precisa ser mostrado na tela. O uso da API *lazy* pode reduzir o tempo de carregamento inicial das páginas que utilizam *imports* custosos, como componentes que fazem *fetch* de dados ou demandam muito processamento computacional;

4. *memo*: evita que um componente seja desnecessariamente renderizado novamente quando seu “componente pai” sofre alguma alteração. Assim como o *hook useMemo*, esta API está relacionada à melhoria de desempenho da aplicação;
5. *startTransition*: utilizada para “quebrar” a renderização causada pela atualização do valor de um state. Normalmente, o React espera uma renderização acabar para iniciar outra, bloqueando a interação até que essa renderização termine. Para evitar esse bloqueio de interação, basta envolver a mudança do valor do state com a API *startTransition*.

4.1.4 Categorização

Podemos classificar os *hooks*, *components* e *APIs* em categorias relacionadas a sua funcionalidade, fazendo isso temos os principais pontos que o React cobre com suas ferramentas nativas. Segue a lista com as categorias:

- Estados: Gerenciamento e compartilhamento
    - *useState, useReducer, useContext, createContext*
- Renderização: Otimização, métricas de desempenho e manipulação
    - *useMemo, useCallback, memo, Profiler*, *startTransition*
- Carregamento de dados: Manipulação e *feedback*
    - *lazy, Suspense*
- DOM: Acesso a atributos e organização
    - *useRef, fowardRef, Fragments*
- Acesso a sistemas externos
    - *useEffect*

4.2 Limitações e pontos de escolha de ferramentas externas

Essa subseção apresenta o resultado das pesquisas por limitações do React, feitas na sua documentação oficial, e PET, feita por meio de enquetes de mercado.

4.2.1 Documentação oficial

A documentação destaca alguns pontos importantes nos quais é recomendado o uso de ferramentas auxiliares, principalmente para a realização de tarefas que não são triviais, como o *fetch* de dados, roteamento e *bundling*. Por resolverem esses problemas, o React recomenda a utilização de *frameworks* desde o início em novos projetos. A integração do React com esses *frameworks* é, segundo eles, a maior oportunidade para ajudar os desenvolvedores que utilizam React a construir melhores aplicações (React, 2024b).

A documentação apresenta três *frameworks* para desenvolvimento web com React: Next.js, Remix e Gatsby. O Next.js, desenvolvido pela Vercel, destaca-se como o mais proeminente devido ao trabalho conjunto com a equipe do React para trazer novas funcionalidades à plataforma. “A equipe do Next.js concordou em colaborar conosco na pesquisa, desenvolvimento, integração e teste de recursos React de ponta, independentemente da estrutura, como React Server Components” (REACT, 2024, tradução nossa).

Sumarizando as limitações e PET encontrados na documentação do React, temos: *Fetch* de dados, roteamento, *bundling* e *frameworks*.

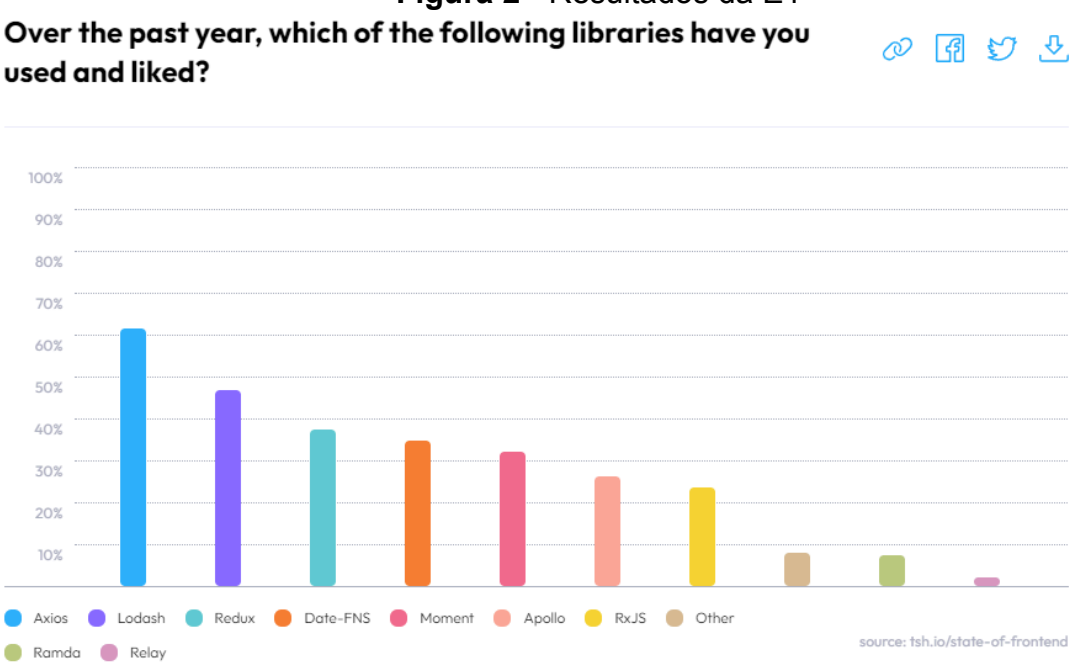
4.2.2 Pesquisas de Mercado

Foram encontradas duas enquetes que atenderam aos critérios deste trabalho: *State of* JavaScript 2022 (2022 STATE OF JS SURVEY, 2022) e *State of frontend 2022* (THE SOFTWARE HOUSE, 2022).

Na página da enquete da The Software House (E1) é disponibilizado uma planilha com os dados brutos (todas as respostas enviadas). Foram removidas na nossa filtragem as perguntas não relacionadas ao React e bibliotecas auxiliares, respostas em branco, respostas de desenvolvedores que não usaram o React no último ano e respostas de desenvolvedores que também utilizaram plataformas que não o React no último ano. Abaixo temos as figuras 2 e 3 que são, respectivamente,

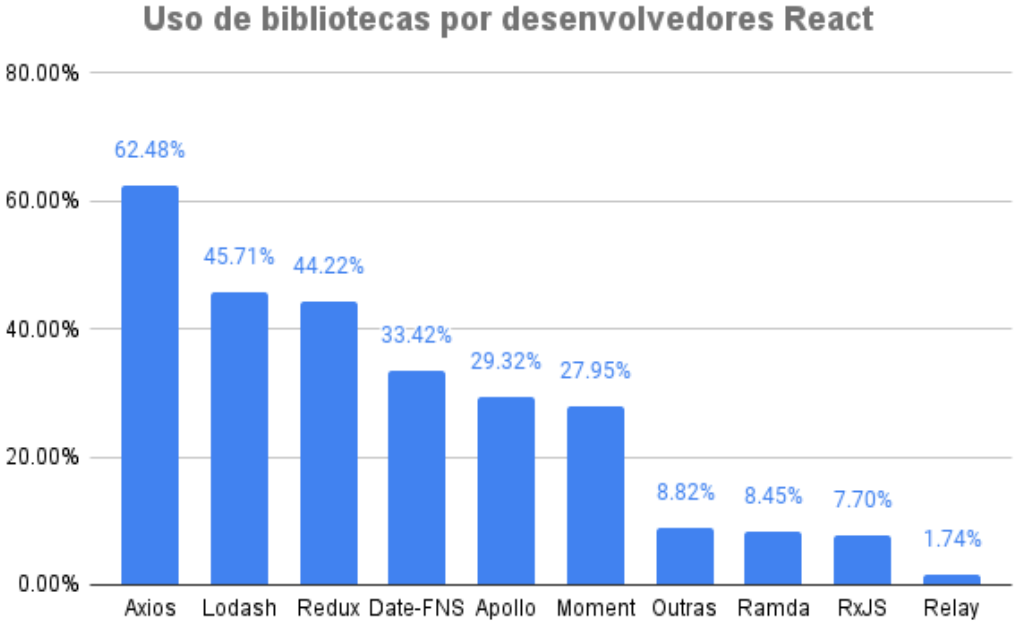
o gráfico das bibliotecas usadas e recomendadas da E1 e o nosso gráfico com os dados filtrados:

Figura 2 - Resultados da E1



Fonte: The Software House (2022)

Figura 3 - Resultados da filtragem da E1

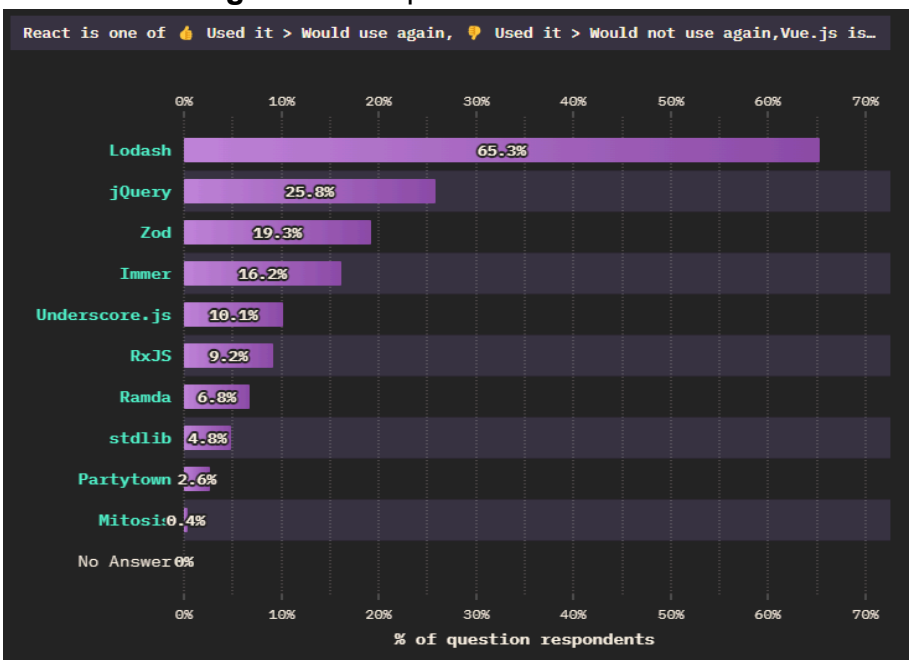


Fonte: Autoria própria (2024)

Além da alteração na porcentagem de cada ferramenta, podemos notar a troca de posições entre Apollo e Moment e a queda de duas posições da RxJS, indicando que muitos desenvolvedores fora do React a utilizam.

Na enquete da StateOfJS (E2) temos a opção de filtrar as repostas na própria página de resultados, como nos interessa apenas as repostas sobre as bibliotecas mais populares entre desenvolvedores React, aplicamos o filtro: utilizou React e não usaria novamente **ou** utilizou React e usaria novamente, **e** não utilizou Vue/Angular/Preact/Svelte (Plataformas concorrentes do React que pontuaram mais de 10% de uso na E2). Assim, eliminamos a possibilidade de alguma biblioteca aparecer nos resultados por ser utilizada em uma dessas plataformas. Abaixo, as figuras 4, 5 e 6, mostram a porcentagem de uso de cada ferramenta:

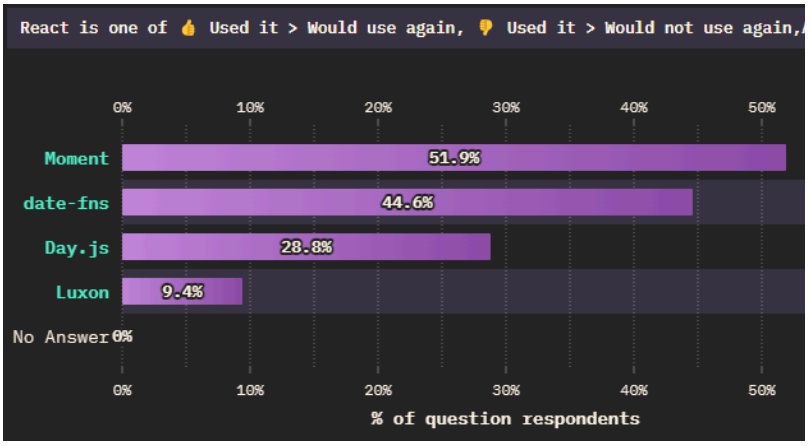
Figura 4 - Respostas filtradas da E2



Fonte: Stateofjs (2022)

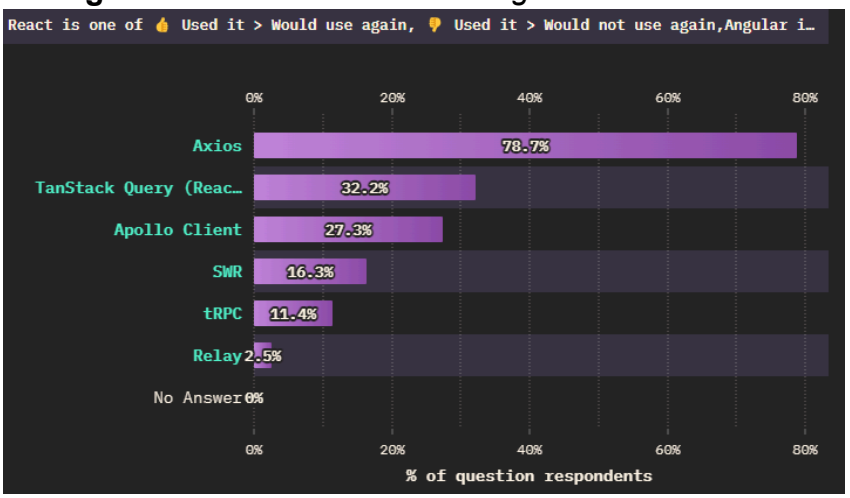
O mesmo foi feito para bibliotecas de gerenciamento de data (Figura 4) e *Fetching* de dados (Figura 5):

Figura 5 - Bibliotecas de gerenciamento de data filtradas



Fonte: Stateofjs (2022)

Figura 6 - Bibliotecas de *fetching* de dados filtradas



Fonte: Stateofjs (2022)

As ferramentas mais utilizadas (com mais de 10% de uso) comuns às duas enquetes foram: Axios, Lodash, Date-fns, Moment e Apollo. Apesar da biblioteca Redux ter aparecido como a terceira mais popular na pesquisa da The Software House, ela não consta nos principais resultados da StateOfJs (talvez por não ter sido listada como alternativa no formulário de respostas da E2), apenas aparece na seção de “outras respostas”, entretanto, ela entrará nas discussões deste trabalho pela sua popularidade entre os desenvolvedores React na E1.

4.2.3 Limitações e PET encontrados

Podemos categorizar as 7 bibliotecas citadas em relação ao tipo de solução que elas oferecem, e, a partir disso, identificar os pontos onde os desenvolvedores

optam por utilizar ferramentas externas no desenvolvimento de projetos React. Ao ler a documentação oficial de cada ferramenta, temos as seguintes categorias:

- Gerenciamento de data
    - Moment
    - Date-fns
- *Fetch* de dados
    - Axios
    - Apollo
- Funções utilitárias
    - Lodash
- Gerenciamento de estados
    - Redux

Unindo os achados na documentação e enquetes de mercado, temos os seguintes limitações e PET do React:

1. *Fetch* de dados;
2. Roteamento;
3. *Bundling;*
4. *Frameworks;*
5. Gerenciamento de data;
6. Gerenciamento de estados;
7. Funções utilitárias.

4.3 Explorando as ferramentas mais utilizadas

Nesta subseção serão abordados as limitações e PET encontrados. Para cada um, exploraremos as ferramentas (achadas nas enquetes e documentação oficial) e faremos recomendações quanto ao seu uso em projetos com React.

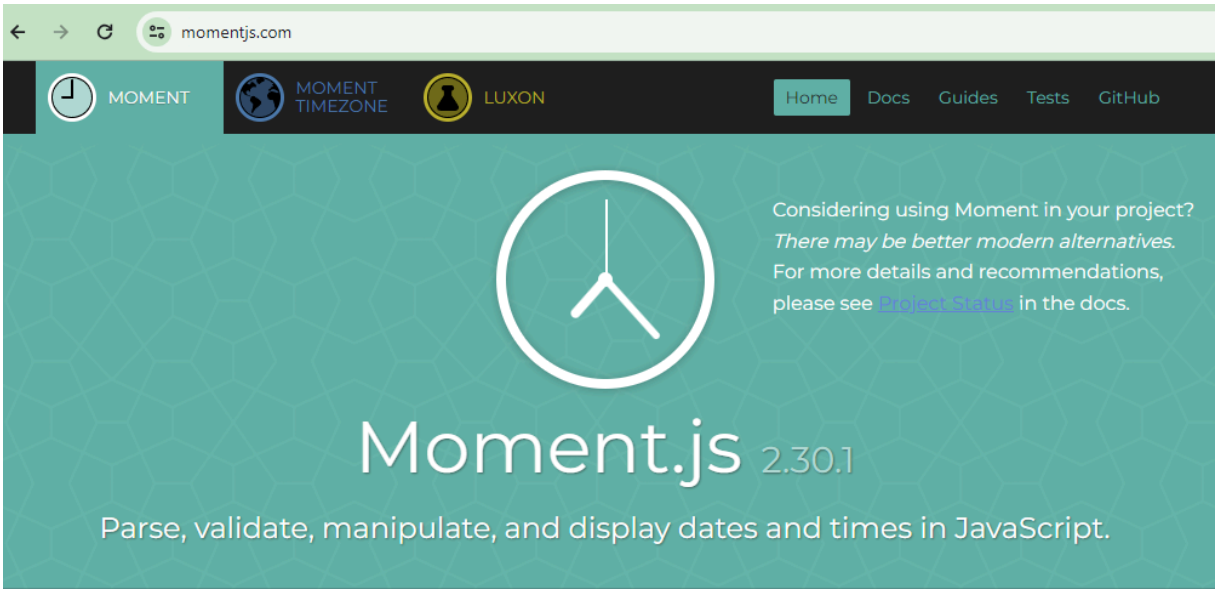
4.3.1 Gerenciamento de Data

Segundo o TC39[2] (2024), o tipo nativo do JavaScript utilizado para manipulação de data (*Date*) tem sido um problema antigo na linguagem. São algumas das maiores reclamações dos desenvolvedores a falta de suporte para fusos horários diferentes da hora local do usuário e a falta de suporte para calendários não gregorianos (Pint, 2017).

4.3.1.1 Moment.js

A biblioteca Moment.js, ou apenas Moment, surgiu em 2011 para melhorar a manipulação das datas no JavaScript, ela resolve os problemas citados anteriormente e também implementa outras funcionalidades. Entretanto, assim que chegamos na página de sua documentação oficial, nos deparamos com um aviso, nele está escrito (em tradução direta): “Está pensando em usar o Moment em seu projeto? Pode haver melhores alternativas modernas. Para mais detalhes e recomendações, consulte a situação do projeto nos documentos.”.

Figura 7 - Página inicial da Moment



Fonte: Momentjs (2024)

A documentação do Moment ressalta que o projeto está finalizado e não receberá atualização com mudanças para se adequar ao JavaScript moderno. A própria biblioteca também recomenda o uso de outras ferramentas para

[2] Grupo de desenvolvedores e pesquisadores que mantém e evoluem as definições do JavaScript.

gerenciamento de data em novos projetos. Dentre as recomendações aparece justamente a outra ferramenta encontrada em nossa pesquisa, a date-fns.

4.3.1.2 Date-fns

A date-fns se define como uma “Biblioteca moderna de utilitários de data em JavaScript” (date-fns, 2023, tradução nossa). Diferente da Moment, ela trabalha com objetos imutáveis e tamanhos de pacote pequenos, esses que são dois dos principais alvos de críticas à Moment citadas em sua documentação oficial.

4.3.1.3 Recomendações

Considerando o exposto, fica aparente que entre as duas bibliotecas de gerenciamento de tempo a mais recomendável é a date-fns, entretanto, talvez no futuro não seja necessária a utilização de bibliotecas externas para lidar com manipulação de data. Está em andamento uma proposta feita ao TC39 por uma implementação nativa do JavaScript que resolve os problemas discutidos anteriormente, chamada de “*Temporal*”. No futuro, antes de utilizar bibliotecas externas, é recomendável checar as atualizações do JavaScript para ver se tal proposta já foi aprovada.

4.3.2 Funções utilitárias

Bibliotecas utilitárias podem salvar horas de desenvolvimento em um projeto, elas oferecem funcionalidades complexas que foram rigorosamente testadas pelos seus desenvolvedores e pela comunidade.

4.3.2.1 Lodash

A biblioteca Lodash disponibiliza módulos que auxiliam na manipulação de *arrays*, objetos e *strings* JavaScript (Sirois; Hall, [s.d]). Porém, muitas das funcionalidades da Lodash já foram implementadas nas novas versões do JavaScript atual (YOU-DONT-NEED-LODASH, 2024), como *map* (*._map* na Lodash*), filter* (*._filter* na Lodash) e outras que o próprio React já implementa (com

as devidas adequações) como a “*._memo”* da Lodash, que no React é feito com o *hook useMemo*.

Muito da popularidade da biblioteca nas enquetes pode ser devido a projetos antigos que fazem uso dela, fazendo com que novos desenvolvedores precisem utilizá-la, como bem disse Andrzej Wysoczański, diretor de frontend na The Software House, em sua análise dos resultados da enquete da TSH (THE SOFTWARE HOUSE, 2022).

Ainda assim a Lodash pode ser recomendada, a biblioteca possui funcionalidades úteis cuja implementação pode ser custosa, como a “*._debounce”*, que, por exemplo, pode ser utilizada na otimização de caixas de pesquisa, servindo para atrasar o disparo da pesquisa causado por termos que estão sendo digitados.

4.3.2.2 Recomendações

Fica a mesma recomendação feita para o uso das bibliotecas de gerenciamento de data. Antes de aderir a Lodash, cheque se a funcionalidade que você está procurando já foi implementada nativamente no JavaScript ou React.

4.3.3 *Fetch* de Dados

O Axios e o Apollo Client são duas bibliotecas amplamente utilizadas para facilitar a comunicação com servidores e a integração de APIs.

4.3.3.1 Axios

Axios é uma biblioteca JavaScript amplamente utilizada para fazer requisições HTTP a servidores. Ela oferece uma lista de funcionalidades para interagir com REST APIs, como para envio de solicitações HTTP (GET, POST, PUT, DELETE), tratamento de erros, definição de cabeçalhos personalizados entre outras (Axios, 2024).

4.3.3.2 Apollo

O Apollo Client é uma biblioteca que simplifica o processo de busca, armazenamento em cache e modificação de dados da aplicação que utilizam o APIs GraphQL. Suas principais características incluem: busca declarativa de dados, suporte para TypeScript, adoção incremental e total compatibilidade com APIs GraphQL (Apollo, 2024).

4.3.3.3 Recomendações

O Axios é mais recomendado em cenários onde a aplicação precisa interagir com REST APIs tradicionais, fornecendo uma solução simples e direta para fazer requisições HTTP. Ele é útil em projetos menores ou mais simples, onde a complexidade do gerenciamento de estado é mínima e não há a necessidade de utilizar consultas GraphQL. Já o Apollo Client é melhor aproveitado em projetos que fazem uso de APIs GraphQL, oferecendo uma maneira de executar consultas e gerenciar o estado da aplicação.

É importante lembrar que a documentação do React sugere o uso de ferramentas auxiliares para o *fetch* de dados ou o uso das ferramentas disponibilizadas por um *framework* React. Utilizando o framework Next.js (versão 13 em diante) como exemplo, suas ferramentas nativas fornecem muitas soluções para o *data fetching* e talvez não seja necessário a utilização do Axios para interação com RESTful APIs. Para interação com APIs GraphQL com o Next.js, o Apollo criou uma biblioteca experimental para dar suporte as atualizações do Next.js 13 em diante.

4.3.4 Gerenciamento de Estado

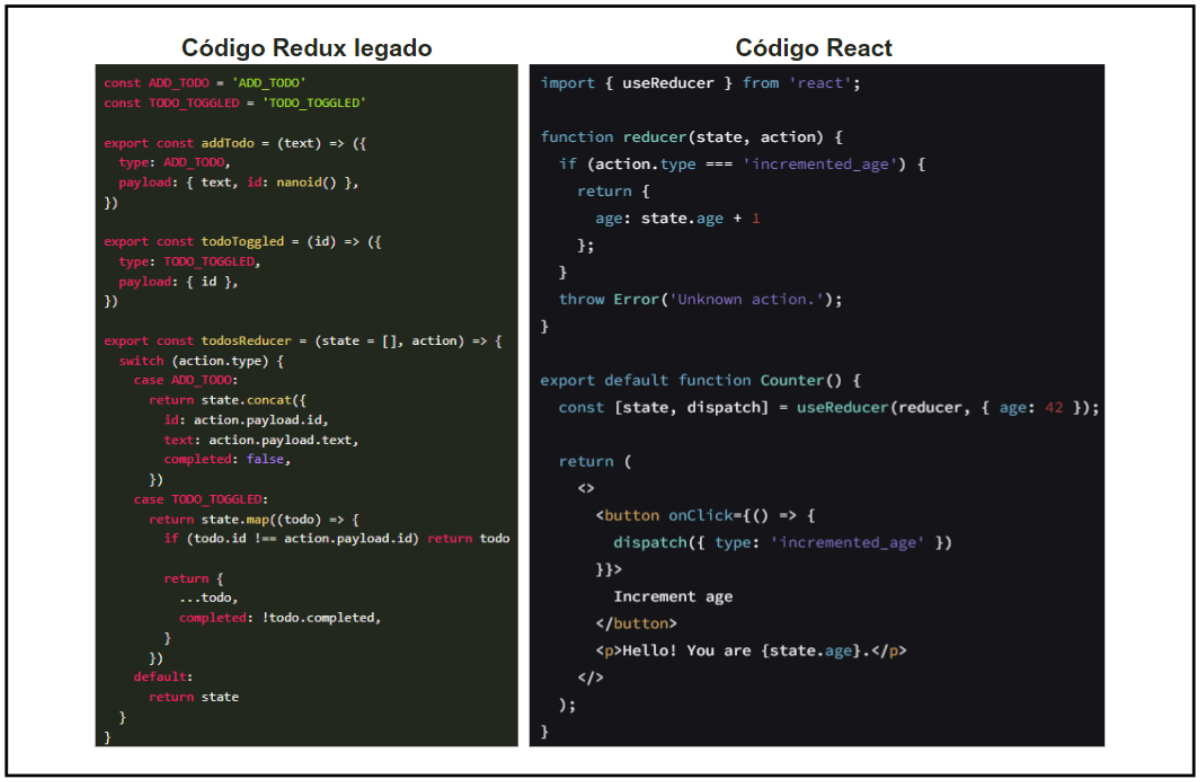
O gerenciamento de estado é uma parte importantes do React, é através dos estados de componentes que informações são exibidas e atualizadas na tela do usuário.

4.3.4.1 Redux

Resumidamente, o Redux oferece a criação de estados globais e atualização preditiva de acordo com ações definidas a esses estados (Redux, 2024). Isso quer dizer: a criação de um estado global que, por exemplo, pode ser passado entre

vários componentes sem a ocorrência do *prop drilling*, semelhante a *Context API* do React; a definição de ações que podem alterar o valor de um estado, semelhante ao *dispatch* (*useReducer*) do React; e a atualização do estado conforme a ação realizada, semelhante ao *reducer* (*useReducer)* do React. Abaixo podemos ver exemplos de uso retirados das documentações oficiais do Redux e React:

Figura 8 - Comparação entre código Redux e código React



Fonte: Autoria própria (2024)

Conforme as informações do parágrafo anterior, vemos que o React já implementa nativamente as funcionalidades principais do Redux, porém, com o lançamento do Redux Toolkit (RTK), uma nova maneira de utilizar Redux foi introduzida. A documentação afirma e reforça que o RTK é a abordagem oficial recomendada para escrever Redux e também que ele simplifica a maioria das tarefas, evita erros comuns e facilita a escrita de aplicações (Redux, 2024).

Com o uso do RTK, o código da figura x pode ser reescrito da seguinte forma:

Figura 9 - Código Redux com o RTK

features/todos/todosSlice.js

```
import { createSlice } from '@reduxjs/toolkit'

const todosSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    todoAdded(state, action) {
      state.push({
        id: action.payload.id,
        text: action.payload.text,
        completed: false
      })
    },
    todoToggled(state, action) {
      const todo = state.find(todo => todo.id === action.payload)
      todo.completed = !todo.completed
    }
  }
})

export const { todoAdded, todoToggled } = todosSlice.actions
export default todosSlice.reducer
```

Fonte: Redux.js.org (2024)

Além da melhor legibilidade do código, o RTK traz outras melhorias ao Redux:

- Elimina mutações acidentais;
- Facilita a escrita de código Redux em um único arquivo;
- Elimina a necessidade de escrever lógicas de atualização manualmente;
- Oferece excelente suporte ao TypeScript.

4.3.4.2 Recomendações

Um dos questionamentos que aparece na página de perguntas frequente do Redux é a de quando se deve usar a ferramenta. Para Dan Abramov, um dos criadores do Redux, não é necessário utilizar o Redux até se ter problemas com o React (Redux, 2024). Da mesma maneira, recomendamos a utilização do Redux apenas se as ferramentas nativas do React não forem suficientes.

4.3.5 Roteamento

Para esta funcionalidade a documentação do React recomenda fortemente o uso de *frameworks* com um sistema de roteamento próprio, tanto é que não é passada nenhuma recomendação de bibliotecas de roteamento na documentação do React (versão 18.2.0).

A única citação de bibliotecas de roteamento que encontramos nas páginas oficiais do React foi a feita na aba de “comunidade” do site legado do React, onde são listadas algumas opções de bibliotecas. Dentre as opções citadas, a única que consta nos resultados das enquetes (E1 e E2) é a React Router, que aparece na E2, na aba “outras bibliotecas”, com 3,4% de uso entre os desenvolvedores React.

4.3.5.1 Recomendações

A forte indicação do React por implementações de *frameworks* e a impopularidade de bibliotecas de roteamento entre as respostas de desenvolvedores sinalam que o uso de soluções isoladas de roteamento não são a opção mais recomendável. Entretanto, é, sim, possível realizar tal funcionalidade com bibliotecas como a React Router, principalmente em projetos mais simples.

4.3.6 *Bundlers*

O objetivo principal dos bundlers é organizar módulos e dependências em projetos front-end, otimizar o desempenho do site e reduzir o tamanho dos arquivos.

Na documentação oficial do React são citados o Vite e Parcel, ambos aparecem na E2 (na seção de *Build Tools*) com 48% e 27% de uso, respectivamente.

O Vite é o compilador dos *frameworks* Remix e Astro (Vite, 2024) e foi desenvolvido por Evan You, o criador do Vue.js (concorrente do React). Ele apresenta maior velocidade de compilação e maior capacidade de customização que o Parcel, que, por outro lado, demanda menos configuração na utilização e suporta navegadores mais antigos (Antipov, 2024).

4.3.6.1 Recomendações

Se for necessário ter mais controle sobre as escolha em um novo projeto, o Vite é a melhor opção por sua alta capacidade de configuração. Já o Parcel é mais recomendável se for necessária uma ferramenta com um suporte mais abrangente a navegadores e maior facilidade de adoção.

4.3.7 *Frameworks*

Como dito na subseção 4.2.1, o React recomenda o uso de *frameworks* desde o início de um novo projeto, já que ele pode se tornar muito complexo a medida que for crescendo. Os *frameworks* React resolvem questões como roteamento, *fetch* de dados e *bundling*. Next.js, Remix e Gatsby são as recomendações da documentação do React e também são os *frameworks* da plataforma que aparecem nos relatórios das enquetes utilizadas neste trabalho.

Diferentemente das ferramentas dos outros pontos de escolha, os *frameworks* apresentam soluções para múltiplas questões, de maneira que fazer uma comparação entre eles (em suas versões mais atuais) rende um trabalho por si só dada a quantidade de funcionalidades e diferenças em suas implementações.

4.3.6.1 Recomendações

Para a nossa recomendação, nos baseamos nas opiniões dos milhares de desenvolvedores que responderam as enquetes selecionadas. Na E1 temos uma seção que mostra a taxa de uso e aprovação de cada framework, enquanto na E2 temos a “taxa de retenção”, que mede o interesse em usar novamente tal framework entre os desenvolvedores já o utilizaram.

A E1 mostra o Next.js com 43%, Gatsby com 11,6% e Remix com 8,8% de uso e aprovação. Na E2 temos o Next.js com 90%, Remix com 81,8% e Gatsby com 38,4% de retenção. Com esses dados e sabendo que a equipe do React trabalha em conjunto com a do Next.js em pesquisa e inovação de novas funcionalidades (subseção 4.2.1), podemos concluir que a opção de Framework mais recomendada é a da Vercel, o Next.js.

4.4 Quadro com os resultados encontrados

Quadro 2 - Sumário dos resultados

Limitações e PET	**Tipo**	**Recomendação**	**Alternativas**
Framework	PET	Next.js	Remix ou Gatsby
Fetch de dados	Limitação	Solução do *framework*	Axios para REST ou Apollo para GraphQL
Roteamento	Limitação	Solução do *framework*	React Router
Bundling	Limitação	Solução do *framework*	Vite
Gerenciamento de estados	PET	React Reducer + React Context	Redux ToolKit
Funções utilitárias	PET	Soluções do JavaScript	Lodash
Gerenciamento de data	PET	Date-fns	*Temporal* (caso lançada)

Fonte: Autoria própria (2024)

5 CONSIDERAÇÕES FINAIS

O trabalho realizado tem uma importância significativa no contexto do desenvolvimento web, especialmente na área do front-end, onde o React tem se destacado como uma das principais tecnologias. A motivação por trás deste estudo foi de auxiliar os desenvolvedores a compreender as limitações e capacidades do React, identificar os pontos de adoção de ferramentas auxiliares e, finalmente, recomendar as melhores opções com base em evidências objetivas.

É fundamental destacar que o trabalho foi conduzido sem qualquer viés de opinião, priorizando fontes confiáveis e abertas ao público, como documentações oficiais e dados provenientes da comunidade de desenvolvedores. No entanto, durante a pesquisa, foi identificada uma lacuna significativa de enquetes disponíveis que abordam o uso de bibliotecas no front-end. Isso evidencia a necessidade de mais estudos nessa área para embasar futuras recomendações de forma mais abrangente e precisa.

Apesar das limitações encontradas, os objetivos do trabalho foram alcançados. No entanto, reconhecemos haver espaço para melhorias, principalmente no que diz respeito à quantidade e qualidade do material disponível. A inclusão de mais enquetes e pesquisas de mercado específicas para o React pode fornecer percepções valiosas para aprimorar ainda mais esse estudo.

É importante ressaltar que algumas questões cruciais, como o uso do TypeScript, soluções de estilização e ferramentas de teste, não puderam ser abordadas, seja por limitações nas enquetes ou devido a sua natureza subjetiva e opcional. No entanto, esses aspectos podem ser explorados em estudos futuros, contribuindo para uma compreensão mais completa do ecossistema React.

Para trabalhos futuros, recomendamos a utilização de enquetes específicas para o React, como a State of React 2023, enquete não teve seus resultados disponibilizados a tempo para entrar nesse trabalho. Além disso, é fundamental estar atento às atualizações e propostas de novas funcionalidades do React e do JavaScript, uma vez que essas mudanças podem impactar significativamente as práticas de desenvolvimento web.

Por fim, é importante destacar que este trabalho representa um esforço contínuo e evolutivo. O desenvolvimento web é um campo dinâmico e em constante mudança, e, portanto, a pesquisa e aprimoramento deste estudo devem ser vistos

como um processo contínuo para acompanhar as inovações e necessidades da comunidade de desenvolvedores.

REFERÊNCIAS

2022 STATE OF JS SURVEY. **The 2022 State of JS survey**. [*S. l.*]: Sacha Greif, 2022. Disponível em: https://2022.stateofjs.com/en-US/. Acesso em: 4 mar. 2024.

ANTIPOV, A. **Head-to-Head: Parcel vs Vite Analysis**. In: MOIVA. Head-to-Head: Parcel vs Vite Analysis. [S. l.], 2024. Disponível em: https://moiva.io/?npm=@parcel/core+vite#:~:text=As%20a%20zero%2Dconfiguration%20build,build%20process%20to%20their%20needs. Acesso em: 4 abr. 2024.

BHALLA, A; GARG, S; SINGH, P. **Present Day Web-Development Using Reactjs**. International Research Journal of Engineering and Technology (IRJET), [s. l.], v. 7, ed. 5, 5 maio 2020. Disponível em: https://www.irjet.net/archives/V7/i5/IRJET-V7I5223.pdf. Acesso em: 28 mar. 2024.

BOEHM, B. W. **Verifying and validating software requirements and design specifications**. IEEE Software, 1984.

BRUCKHAUS, T; MADHAVI, N.H; JANSSEN, I; HENSHAW, J. T. **The impact of tools on software productivity**. in: IEEE Software, vol. 13, no. 5, pp. 29-38, Sept. 1996, doi: 10.1109/52.536456.

CINCOVIĆ, J.; PUNT, M. **Comparison: Angular vs. React vs. Vue**. Which framework is the best choice?. In: Zdravković, M., Konjović, Z., Trajanović, M. (Eds.) ICIST 2020 Proceedings, pp.250-255, 2020

CLARK, A; VAUGN, B; ABERNATHY, C; ABRAMOV, D; NABORS, R; HANLON, R; MARKBÅGE, S; SETH, W. **The Plan for React 18**. React.dev, 2021. Disponível em: https://react.dev/blog/2021/06/08/the-plan-for-react-18#a-gradual-adoption-strategy. Acesso em: 31 mar. 2024.

COPELAND, T; KOLLER, T; MURRIN, J. **Valuation, Measuring and Managing the Values of Companies.** John Wiley Sons, New York, 2000.

DATE-FNS, **Biblioteca moderna de utilitários de data em JavaScript**. 2023. Disponível em: https://date-fns.org/. Acesso em: 28 mar 2024.

FERREIRA, F; BORGES, H; Valente, M. **On the (un-)adoption of JavaScriptfront-end frameworks**. Softw Pract Exper. 2022;52(4):947-966. doi: 10.1002/spe.3044

GETTING Started. In: AXIOS. **Promise based HTTP client for the browser and node.js**. [S. l.], 2024. Disponível em: https://axios-http.com/. Acesso em: 30 mar. 2024.

GETTING Started. In: VITE. **Next Generation Frontend Tooling**. [S. l.], 2024. Disponível em: https://vitejs.dev/guide/. Acesso em: 30 mar. 2024.

INTRODUCTION to Apollo Client, In: APOLLO. **Documentation**, 2024. Disponível em: https://www.apollographql.com/docs/react/. Acesso em: 30 mar. 2024.

KOMPERLA, V.; DEENADHAYALAN, P.; GHULI, P.; PATTAR, R. **React: A detailed survey**. Indonesian Journal of Electrical Engineering and Computer Science, [*s. l.*], ano 2022, v. 26, ed. 3, p. 1710-1717, 3 jun. 2022. DOI 10.11591/ijeecs.v26.i3.pp1710-1717. Disponível em: https://ijeecs.iaescore.com/index.php/IJEECS/article/view/27582/16433. Acesso em: 2 abr. 2024.

LEVLIN, M. **DOM benchmark comparison of the front-end**. Tese (Mestrado em Sociologia) – Faculdade de Ciência da Computação Universidade Åbo Akademi. Turku, Finlândia, p. 45. 2020.

LIE, H. W; BOS, B. **Cascading Style Sheets**. 3. ed. Oslo: Håkon Wium Lie and Bert Bos, 2005. 1-4 p.

MAURYA, P. **Web Development Using ReactJS**. Journal of Current Research in Engineering and Science. Volume 6 –Issue 2, August 2023

MDN WEB DOCS. **O que é JavaScript?**. mdn web docs, 2023. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript. Acesso em: 02 abr. 2024.

PINT, M. Corrigindo a data do JavaScript. 2017. Disponível em: https://maggiepint.com/2017/04/09/fixing-javascript-date-getting-started/. Acesso em: 28 mar 2024.

RAGGETT, D. **Raggett on HTML 4**. Massachusetts: Addison-Wesley Longman, 1998. Disponível em: http://www.w3.org/People/Raggett/book4/ch02.html. Acesso em: 16 mar 2024.

RAUSCHMAYER, A. **Speaking JavaScript**. 1. ed. California: O'Reilly Media, Inc., 2014. p. 43.

RAWAT, P; MAHAJAN, A. **ReactJS: Um Framework de Desenvolvimento web Moderno**. Volume 5 - 2020, Edição 11 – Novembro.Series, 2020.

REACT. **Built-in React Hooks**. In: REACT. React: The library for web and native user interfaces. [S. l.], 2024. Disponível em: https://react.dev/reference/react/hooks. Acesso em: 12 mar. 2024a.

REACT. **Start a New React Project**. In: REACT. React: The library for web and native user interfaces. [S. l.], 2024. Disponível em: https://react.dev/learn/start-a-new-react-project. Acesso em: 12 mar. 2024b.

SIROIS, J; HALL, Z. **Why Lodash?**. In: LODASH. Lodash: A modern JavaScript utility library delivering modularity, performance & extras. [S. l.], [2024?]. Disponível em: https://lodash.com/. Acesso em: 30 mar. 2024.

STACK OVERFLOW. **Stack Overflow Developer Survey 2023**. Disponível em: https://survey.stackoverflow.co/2023/. Acesso em: 20 mar. 2024.

TC39. **Temporal**. [S. l.], [2024?]. Disponível em: https://tc39.es/proposal-temporal/docs/index.html. Acesso em: 28 mar. 2024.

THE SOFTWARE HOUSE. **State of frontend 2022**. [S. l.]: Aleksandra Dąbrowska, 2022. Disponível em: https://tsh.io/state-of-frontend/. Acesso em: 4 mar. 2024.

VARGAS, E. L; ANICHE, M; TREUDE, C; BRUNTIK, M; GOUSIOS, G. **Selecting Third-Party Libraries: The Practitioners' Perspective**. ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA.

W3SCHOOLS. **Introdução ao CSS**. 2024a. Disponível em:https://www.w3schools.com/html/default.asp. Acesso em: 12 mar. 2024.

W3SCHOOLS. **Referências HTML**. 2024b. Disponível em: https://www.w3schools.com/html/default.asp. Acesso em: 12 mar. 2024.

WHY Redux Toolkit is How To Use Redux Today. In: REDUX. **Homepage**. A JS library for predictable and maintainable global state management. [S. l.], 2024. Disponível em: https://redux.js.org/introduction/why-rtk-is-redux-today. Acesso em: 30 mar. 2024.

XIAOSHU, W. **Optimized Development of Web Front-end Development Technology**. Journal of Physics: Conference Series, v. 1693, n. 1, p. 012057, 1 dez. 2020.

XING, Y.; HUANG, J.; LAI, Y. **Research and Analysis of the Front-end Frameworks and Libraries in E-Business Development**. Proceedings of the 2019 11th International Conference on Computer and Automation Engineering - ICCAE 2019, 2019.

YOU-DONT-NEED. **You don‘t (may not) need Lodash/Underscore**. [S. l.], 2024. Disponível em: https://github.com/you-dont-need/You-Dont-Need-Lodash-Underscore/blob/master/README.md. Acesso em: 2 abr. 2024.