



ARQUITETURA SERVER-SIDE

Adriana Cássia Reis dos Santos – Aula 02

Professores

ADRIANA CÁSSIA REIS DOS SANTOS

Professora Convidada

Adriana Cássia iniciou sua carreira na área de TI aos 14 anos, em seu primeiro estágio do Ensino Médio como Suporte Técnico, onde permaneceu por seis anos. Após se graduar em Engenharia da Computação, passou a atuar como Analista de Sistemas. Realizou cursos e formações voltadas para programação, entre eles certificação Java (SCJP), migrando para a área de Programação, onde tem atuado nos últimos anos.

MIGUEL GOMES XAVIER

Professor PUCRS

Possui mestrado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul e está cursando doutorado em Ciência da Computação na mesma instituição, atuando principalmente nas áreas de alto desempenho, sistemas distribuídos, virtualização e cloud computing. Atualmente participa de projetos de pesquisa em cooperação com diferentes universidades envolvendo gerência de recursos em arquiteturas de alto desempenho. Tem participado de projetos de análise de dados (BigData), realizando contribuições científicas em prol do avanço da área na indústria e na academia.

Ementa da disciplina

Estudo sobre Arquitetura cliente-servidor para aplicações web. Introdução aos frameworks MVC server-side: Node.js, Express, Nestjs. Estudo de programação assíncrona e programação reativa. Desenvolvimento de aplicações web com o conceito de uso de serviços.

Arquitetura Server-Side

Por Professora Adriana Cássia

EMENTA

DA DISCIPLINA

- SEO (SEARCH ENGINE OPTIMIZATION)
- APLICAÇÕES WEB SPA'S (SINGLE PAGE APPLICATIONS)
- SSR (SERVER SIDE RENDERING)
- BIBLIOTECA DE DESENVOLVIMENTO
- REACT
- REDUX
- REACT HOOK FORM
- TEST JEST
- TDD

SEO (SEARCH ENGINE OPTIMIZATION)

Podemos definir SEO (Search Engine Optimization) como uma série de aprimoramentos no código e no conteúdo de um site visando que ele seja encontrado mais facilmente e melhor avaliado por algoritmos de mecanismos de busca.

Uma otimização eficiente posiciona o objeto de análise no topo das páginas de resultado (um bom ranqueamento), podendo chegar até mesmo na primeira colocação. Quanto melhor for o ranqueamento, maior a chance de conquistar o clique da pessoa usuária.

Por que otimizar um site para mecanismos de busca?

O marketing é um campo de trabalho que deve estar sempre atento às mudanças nas relações entre empresas e mercado consumidor. A mudança mais relevante dos últimos tempos teve a internet como responsável e criou todo o segmento digital, já em estágios muito desenvolvidos.

A disseminação intensa de informações mudou parte da dinâmica do marketing. Além de serem valorizadas abordagens mais sutis de aproximação à pessoa usuária, uma das grandes prioridades de uma estratégia passou a ser o destaque: seja o que se dá a um negócio ou a seu conteúdo, em meio a uma infinidade de informações em toda a web.

Atualmente, as pessoas que consomem priorizam a pesquisa online antes de tomar uma decisão de compra. É por isso que estar bem posicionado/ranqueado é um argumento positivo a ser levado em consideração.

Onde o SEO pode ser utilizado

O SEO pode ser adotado como recurso para atingir diferentes tipos de metas. Algumas situações comuns são:

- em casos de sites que contam com a audiência para obter lucros com cliques em anúncios, como portais de notícias que precisam gerar tráfego para tornar mais visível os conteúdos publicitários disponibilizados por seus parceiros;
- quando prestadores de serviços buscam conversões para seus negócios na rede, via formulários, email, entre outros;
- em situações em que e-commerces precisam chamar a atenção para seus produtos (fazendo um trabalho com palavras-chave mais específicas é possível obter maiores chances de conversão);
- no branding, pois o SEO gera um melhor posicionamento da marca e cria autoridade em um determinado assunto, fazendo com que a empresa se torne referência no que faz.

Onde o SEO pode ser utilizado

De uma maneira geral, sempre que você precisar tornar seu conteúdo mais visível para as pessoas, aposte no SEO. Como os algoritmos fazem uma pesquisa minuciosa em toda a rede em busca de informações relevantes para as pessoas que fazem pesquisas na internet, toda vez que eles chegarem até a sua página e identificarem sua relevância, darão a ela notas altas que serão importantes no ranqueamento dos mecanismos de busca.

APLICAÇÕES WEB SPA'S (SINGLE PAGE APPLICATIONS)

A sigla SPA vem de Single Page Applications, ou Aplicações de Página Única. Apesar de o nome permitir a dedução, isso não significa que a aplicação terá apenas uma única página.

O que muda na verdade é a forma com que a página irá ser carregada. Estamos acostumados com aplicações onde as páginas são renderizadas do lado do servidor, independente da tecnologia utilizada. Isso traz um efeito: cada nova página que precisa ser carregada se traduz em uma nova requisição para o servidor, requisição esta para que o browser consiga carregar o HTML, o CSS e o JavaScript da nova página requisitada.

APLICAÇÕES WEB SPA'S (SINGLE PAGE APPLICATIONS)

Por isso, em várias aplicações web, vemos o browser levando um “tempinho” para carregar a nova página, ou vemos a nova página ficar em branco inicialmente para ser carregada em seguida. Estes comportamentos ocorrem por causa do tempo entre o browser fazer a requisição para carregar a nova página e o servidor responder com as novas estruturas a serem renderizadas.

APLICAÇÕES WEB SPA'S (SINGLE PAGE APPLICATIONS)

Aplicações baseadas em frameworks SPA funcionam de maneira diferente, pois nelas não há a necessidade de se fazer requisições para carregamento de novas páginas. A aplicação seria “carregada” por inteiro na primeira requisição, onde todo o HTML, CSS e JavaScript necessários seriam carregados de uma vez. A partir deste momento, quando novas páginas precisassem ser carregadas, estas seriam carregadas através de rotinas JavaScript, retirando a necessidade de requisições para o servidor com a finalidade de obter o novo conteúdo a ser renderizado.

APLICAÇÕES WEB SPA'S (SINGLE PAGE APPLICATIONS)

Aplicações SPA de maneira geral nos permitem obter algumas vantagens. Uma delas é a possibilidade de otimização em geral da performance da aplicação ao deslocar todo o esforço de renderização para o cliente e permitir um tráfego de dados mais leve entre cliente e servidor. Outra é o reaproveitamento de código através de tecnologias como React/React Native e Angular/Ionic, o que possibilita o desenvolvimento com menor esforço e mais padronizado até mesmo de aplicações mobile. Porém, aplicações SPA têm a tendência de possuírem alguns pontos fracos, como justamente o deslocamento do esforço de renderização para o cliente e, em alguns casos, questões de SEO.

Como aplicações SPA funcionam?

De maneira geral, em uma aplicação SPA, o carregamento dos recursos (como CSS, JavaScript e HTML das páginas) ocorre apenas uma única vez: na primeira vez em que o usuário acessa a aplicação. Nesse primeiro acesso, todo o conteúdo HTML, CSS e JavaScript já é transferido para o cliente. A partir deste momento, quando o usuário transitar pelas páginas da aplicação, não será necessário mais fazer requisições para o servidor para a carga dessas novas páginas: o conteúdo relacionado a elas já foi baixado no primeiro acesso. O que acontece nesse momento é que o conteúdo da página é carregado via JavaScript, código este que é justamente gerado com base nos frameworks SPA, como Angular, React e Vue.js. Por isso, dizemos que o processamento do carregamento das páginas e seus respectivos recursos passa para o cliente, já que JavaScript é uma linguagem majoritariamente client side (existem algumas exceções, como quando trabalhamos com Node.js).

Como aplicações SPA funcionam?

Neste momento, o servidor fica responsável não mais por renderizar e processar a renderização do conteúdo, mas sim por lidar com os dados a serem manipulados pela aplicação. As operações de persistência em bancos de dados ou a devolução de conteúdo para ser renderizado (como uma lista de clientes, por exemplo) passam a ser exclusivamente da aplicação hospedada do lado do servidor. Atualmente, o que é comum é que o servidor exponha uma API RESTful para ser consumida por uma aplicação SPA. A aplicação SPA se comunica com essa API RESTful através de chamadas HTTP, trafegando dados em formatos como XML e JSON. A aplicação do lado do servidor fica responsável por responder a estas chamadas HTTP, realizando os processos de negócio e de persistência que sejam pertinentes na situação. Essa arquitetura traz uma vantagem muito clara: a separação e isolamento completos do back-end e do front-end. Isso possibilita que duas equipes trabalhem nas duas frentes em paralelo por exemplo, além de permitir esforços mais focados: uma pessoa que lida melhor com aspectos ligados ao front-end pode direcionar todo o seu esforço para tarefas relacionadas ao front-end. O mesmo vale para quem tem mais aptidão ao back-end.

Exemplos de site/aplicativos que utilizam SPA

SPOTIFY - <https://open.spotify.com/>

Google planilhas -

https://docs.google.com/spreadsheets/d/1czK-4sdbcuZqemzTkhXEPqd1p4HYRvpoeEj_ecXCXeQ/edit#gid=0

YouTube - <https://www.youtube.com/>

Como e onde utilizar?

Atualmente, aplicações SPA podem ser utilizadas em praticamente todas as situações, o que explica bastante a popularização de frameworks SPA como Angular, React, Vue.js e Ember na atualidade. Porém, existem algumas situações onde frameworks SPA podem não ser tão adequados.

Como e onde utilizar?

Aplicações que precisem de SEO extremo podem ter problemas se forem desenvolvidas com frameworks SPA. A maioria dos indexadores hoje conseguem entender aplicações SPA, além de dispormos de uma grande quantidade de meta-tags para melhorar este ponto. Porém, é um trabalho adicional frente à clássica renderização de páginas múltiplas no servidor (também chamado de multi-page application). Esse “probleminha” acontece justamente porque o conteúdo não é renderizado no servidor,, sendo renderizado completamente no cliente. Por isso, os mecanismos de busca podem encontrar dificuldade para indexar o conteúdo das páginas, tendo em vista que eles não podem indexar conteúdo gerado do lado do cliente. Isso explica porque a maioria dos frameworks SPA hoje contam com soluções SSR (Server-Side Rendering), onde pelo menos uma parte do conteúdo é carregado do lado do servidor para favorecer os mecanismos de indexação;

Como e onde utilizar?

Aplicações SPA podem ser um pouco mais lentas, principalmente na inicialização. Isso ocorre porque todo o conteúdo precisa ser baixado de uma vez para o cliente no primeiro acesso. Além disso, como todo o processo de renderização passa a ser de responsabilidade do cliente, a aplicação passa a sofrer um pouco com possíveis limitações de hardware de quem a acessa. Porém, obviamente, existem técnicas para mitigar este ponto, como o lazy loading;

Como e onde utilizar?

Há uma ligeira tendência a aplicações SPA serem menos seguras do que aplicações multi-page renderizadas no servidor, principalmente no que diz respeito a ataques XSS. Porém, de fato, isso na verdade está mais atrelado ao conhecimento do desenvolvedor sobre técnicas para evitar estes tipos de ataques; precauções estas que deveriam ser tomadas inclusive em aplicações renderizadas do lado do servidor;

Como e onde utilizar?

Atenção à fragilidade do ecossistema JavaScript. Esse é um ponto bem polêmico, mas é fato que a “instabilidade” de frameworks JavaScript é algo a se considerar. É relativamente comum que você desenvolva uma aplicação com uma determinada versão de um framework SPA, mas seja obrigado a reescrever uma quantidade considerável de código quando precisar atualizar a versão do framework utilizada inicialmente no projeto. Esse é, sem sombra de dúvidas, um fator importante a ser colocado na balança; não só no que diz respeito a utilizar um framework SPA ou um framework MPA, mas também no que diz respeito à decisão sobre qual framework SPA deverá ser utilizado, se for o caso.

Como e onde utilizar?

De maneira geral, aplicações SPA precisam que o JavaScript no browser esteja habilitado, embora não seja tão comum hoje em dia que o mesmo seja desabilitado. Também existem técnicas que podem mitigar este ponto, como o próprio SSR;

Vantagens

- **Melhoria do processo de indexação:** como parte do conteúdo é renderizado no servidor, é possível definir o conteúdo a ser carregado a partir do servidor da aplicação de maneira que este conteúdo colabore com os mecanismos de indexação dos motores de busca;
- **Menor exigência da máquina do cliente,** já que parte do esforço de renderização fica ainda no servidor;
- **Melhor performance da aplicação em geral na maioria dos casos,** justamente porque parte da aplicação já é pré-renderizada. Isso auxilia a reduzir inclusive incômodos na experiência de usuários de aplicações SSR, como a rápida página em branco (ou flicker) que acontece em uma parte das aplicações SPA (a página fica em branco enquanto o cliente não termina de processar o JavaScript para carregar a aplicação toda);

SSR (Server Side Rendering)

SSR é a sigla para Server Side Rendering, ou Renderização do Lado do Servidor. O SSR vem para solucionar um pouco dos problemas das aplicações SPAs, tentando manter suas principais vantagens. O SSR inverte o processo de renderização, trazendo uma parte do esforço de renderização de aplicações SPA para o servidor, de maneira similar ao carregamento tradicional.

SSR (Server Side Rendering)

O SSR pode fornecer aos usuários um carregamento mais eficiente da aplicação, já que parte da renderização é feita no servidor. Além da possibilidade de melhoria da performance, o SSR ajuda a lidar com alguns problemas de SEO (como indexação), já que parte da aplicação ainda é carregada pelo servidor.

Aplicações SSR também são comumente chamadas de universal apps.

Vantagens

- **Melhoria do processo de indexação: como parte do conteúdo é renderizado no servidor, é possível definir o conteúdo a ser carregado a partir do servidor da aplicação de maneira que este conteúdo colabore com os mecanismos de indexação dos motores de busca;**
- **Menor exigência da máquina do cliente, já que parte do esforço de renderização fica ainda no servidor;**

Vantagens

Melhor performance da aplicação em geral na maioria dos casos, justamente porque parte da aplicação já é pré-renderizada. Isso auxilia a reduzir inclusive incômodos na experiência de usuários de aplicações SSR, como a rápida página em branco (ou flicker) que acontece em uma parte das aplicações SPA (a página fica em branco enquanto o cliente não termina de processar o JavaScript para carregar a aplicação toda);

Desvantagens

- O TTFB (time to first byte) em aplicações SSR é maior, pois o servidor precisa justamente pré-carregar parte do conteúdo antes de enviar a resposta para o cliente. TTFB é o tempo entre o servidor receber a requisição e enviar o primeiro conteúdo a ser renderizado pelo cliente. Enquanto o TTFB não acontece, o usuário vê a página sendo “carregada” pelo cliente;
- Um pouco de incômodo dependendo da experiência de carga: como aplicações SSR são pré-renderizadas pelo servidor, assim que o TTFB é concluído, o cliente começa a iniciar imediatamente o processo de renderização, mostrando a aplicação para o usuário. Porém, o usuário só poderá interagir com a aplicação de fato quando o cliente terminar a sua parte da carga da aplicação. Se o processo de carga do lado do cliente for um pouco extenso, isso pode causar uma experiência de usuário estranha;

Quando utilizar um ou outro?

A decisão de utilizar SPA ou SSR vai depender muito dos objetivos da aplicação. Porém, de maneira geral, os seguintes pontos devem ser levados em consideração:

SPA é uma boa opção caso...

- **A página precise oferecer uma experiência de usuário mais rica e fluída;**
- **Existirão muitas interações na página com a renderização de conteúdos dinâmicos;**
- **A indexação no Google não seja prioridade.**

SSR é uma boa opção quando...

- **Ter boa indexação no Google é um requisito;**
- **Existir a necessidade da fluidez do SPA, porém, com um tempo de carga para o usuário mais eficiente;**
- **A aplicação possui um número mais extenso de páginas. Nesse cenário, a divisão do trabalho de renderização com o servidor pode vir a ser interessante.**

O que são as bibliotecas de desenvolvimento?

Uma biblioteca é um conjunto de funções, implementações ou subprogramas, organizados em classes e disponibilizados para outros programadores/desenvolvedores utilizarem. Uma boa analogia aqui seria a de um atalho ou kit composto por partes que ajudarão na execução de tarefas e na construção de um todo.

O propósito de se usar uma biblioteca é realmente resolver problemas, facilitar a utilização de uma linguagem e simplificar processos através da implementação de arquivos (códigos e dados) executáveis prontos para serem invocados; você usa aquilo que precisar.

Existem bibliotecas para manipular strings; acessar, converter, validar, manipular e exibir dados/arquivos; criar interfaces; e muito mais.

O que é React JS?

React é um framework JavaScript criado pelo Facebook (atual Meta) que é usado para criar interfaces de usuário (UI) em aplicativos web. Ele é popular por ser fácil de usar, altamente flexível e escalável, e é usado por muitas empresas de tecnologia, incluindo o Facebook, Instagram e Airbnb.

Principais casos de uso do React

Um dos principais casos de uso do React é criar aplicações web complexas que precisam ser atualizadas em tempo real.

Por exemplo, o Facebook usa o React para criar sua interface de usuário, que precisa ser atualizada constantemente com novas informações de amigos, mensagens, e notificações. O React é ideal para esse tipo de aplicação porque ele permite que você atualize a interface de usuário de forma rápida e eficiente.

Principais casos de uso do React

Outro caso de uso comum do React é criar aplicações web que precisam ser escaláveis e mantidas por equipes grandes.

O React é uma biblioteca modular, o que significa que os componentes podem ser facilmente reutilizados e compartilhados entre diferentes partes da aplicação. Isso torna mais fácil manter e expandir uma aplicação à medida que ela cresce, e permite que equipes de desenvolvimento trabalhem de forma mais eficiente juntas.

Principais casos de uso do React

Além disso, o React é frequentemente usado em conjunto com o React Native para criar aplicações móveis nativas para iOS e Android. Isso permite que você crie aplicações móveis de alta qualidade que se sentem e se comportam como aplicações nativas, mas são construídas usando tecnologias web, como JavaScript e CSS.

React e React Native são a mesma coisa?

A principal diferença entre o React e o React Native é o tipo de aplicação que eles criam. O React é usado para criar aplicações web, enquanto o React Native é usado para criar aplicações móveis. No entanto, ambos compartilham muitos conceitos e sintaxe, então se você é familiarizado com o React, será mais fácil aprender o React Native.

Em resumo, React e React Native são ferramentas diferentes, mas relacionadas. O React é usado para criar aplicações web, enquanto o React Native é usado para criar aplicações móveis. No entanto, ambos compartilham

Mas o que é aplicações móveis?

Aplicativos móveis, também abreviados de “apps” ou chamados de “app mobile”, são softwares desenvolvidos exclusivamente para dispositivos móveis como celulares e tablets, gratuitos ou pagos.

Por que utilizar React?

É popular e amplamente utilizado

O React.JS é um dos frameworks JavaScript mais populares e amplamente utilizados em todo o mundo. Ele é usado por muitas empresas de tecnologia, incluindo o Facebook, Instagram e Airbnb, e é adotado por desenvolvedores em todos os tipos de projetos. Isso significa que o React é uma opção confiável e bem suportada para o seu projeto.

Por que utilizar React?

Oferece muitos recursos e ferramentas úteis

O React.JS oferece muitos recursos e ferramentas úteis que tornam o desenvolvimento de aplicativos web mais fácil e eficiente. Isso inclui bibliotecas de componentes prontos para uso.

Como funciona o React?

O React usa a linguagem JavaScript para criar componentes, que são pequenos pedaços de código que representam uma parte específica da interface do usuário (UI) de um aplicativo. Cada componente tem um estado, que é uma variável que armazena as informações que mudam dentro do componente, como os dados de um formulário ou a cor de um botão.

Quando o usuário interage com o aplicativo, como clicar em um botão ou preencher um formulário, o estado dos componentes é atualizado e reflete as mudanças na UI. Isso é feito com o uso de funções de callback, que são funções que são chamadas quando uma ação é executada pelo usuário.

Como funciona o React?

O React também usa o que é chamado de Virtual DOM (Document Object Model Virtual), que é uma representação em memória da UI do aplicativo. Quando o estado dos componentes muda, o Virtual DOM é atualizado e comparado com o DOM real para determinar quais mudanças precisam ser feitas na UI. Isso é muito mais rápido do que atualizar o DOM diretamente, o que torna o React.JS muito rápido e eficiente.

DOM e Virtual DOM

O DOM, ou Document Object Model, é uma representação em árvore de um documento HTML ou XML. Ele é usado pelo navegador para entender o conteúdo de uma página web e permitir que o usuário interaja com ela.

Por exemplo, quando você clica em um botão em uma página web, o navegador usa o DOM para entender o que deve acontecer. Ele procura o elemento HTML correspondente ao botão no DOM, e depois executa a ação especificada pelo desenvolvedor.

O problema é que o DOM pode ser lento para atualizar e manipular. Ele é uma representação direta da página web, então qualquer alteração nos dados da página requer uma atualização completa do DOM. Isso pode ser muito lento para aplicações web mais complexas, que podem ter milhares ou até milhões de elementos HTML.

React Router

Por padrão, o React vem sem roteamento, pois foi pensado no modelo de aplicação single page (ou "de página única", em português), cuja funcionalidade está concentrada em uma única página. Porém, para tornar isso possível em nosso projeto, precisamos adicionar uma biblioteca chamada react-router (texto em inglês).

DOM e Virtual DOM

Para resolver esse problema, o React introduziu o Virtual DOM. O Virtual DOM é uma representação em memória do DOM, que é atualizada muito mais rapidamente do que o DOM real. Quando um componente do React é atualizado, o Virtual DOM é atualizado primeiro, e depois as alterações são sincronizadas com o DOM real. Isso torna a atualização da interface de usuário muito mais rápida e eficiente.

Em resumo, o DOM é a representação de um documento HTML ou XML no navegador, enquanto o Virtual DOM é uma representação em memória do DOM que é usada pelo React para atualizar a interface de usuário de forma mais rápida e eficiente.


Anatomia de um componente React

Um componente funcional é um tipo de componente no React que é definido como uma função JavaScript. Em vez de usar uma classe, como nos componentes de classe, um componente funcional é apenas uma função que recebe as propriedades como argumento e retorna o elemento React que deseja renderizar.

Isso torna os componentes funcionais mais leves e mais fáceis de entender e manter do que os componentes de classe.

Exemplo de componente funcional

Aqui está um exemplo simples de um componente funcional que exibe um cabeçalho simples:



```
function Titulo(props) {  
  return <h1>{props.texto}</h1>;  
}
```

Um componente funcional é um tipo simples e eficiente de componente no React.JS. Ele é definido como uma função que recebe as propriedades como argumento e retorna o elemento React que deseja renderizar. Isso torna os componentes funcionais fáceis de entender e manter, e é uma ótima opção para a maioria dos casos de uso.

O que são Props no React?

Em React, os componentes são como pequenas "peças de lego" que podem ser combinadas para criar a interface de usuário de uma aplicação. Cada componente é um pedaço de código que segue um determinado padrão, e que pode ser reutilizado várias vezes ao longo de sua aplicação.

As props, ou "propriedades", são um tipo de dados que podem ser passados para um componente React. Esses dados podem incluir coisas como texto, números, imagens, ou até mesmo outros componentes. Ao passar os props para um componente, você pode personalizá-lo de acordo com suas necessidades.

O que são Props no React?

Por exemplo, imagine que você tem um componente chamado "Post" que mostra um título, uma imagem e um texto.

Usando props, você poderia passar diferentes títulos, imagens e textos para cada instância desse componente, sem precisar criar uma nova versão do componente para cada postagem.

O que é JSX?

JSX é uma extensão da linguagem JavaScript que é usada pelo React para criar interfaces de usuário. Ele permite que você misture código JavaScript com sintaxe de HTML, o que torna mais fácil escrever componentes de interface de usuário em um único arquivo de código.

Redux

Redux é uma biblioteca para armazenamento de estados de aplicações JavaScript, criado por Dan Abramov. Ele nasceu através de uma implementação do Flux, uma arquitetura criada pelo Facebook para contribuir com as aplicações de User Interface, utilizando o conceito de fluxo de dados unidirecional. Quando desenvolvemos aplicações utilizando Javascript, sempre temos que lidar com o gerenciamento de estado. O Redux veio para suprir essa necessidade de simplificar o controle dos estados de uma aplicação. Compartilhar estados entre vários componentes diferentes se torna uma coisa muito fácil quando o utilizamos.

Redux

Ele basicamente tira a responsabilidade de cada um dos componentes de armazenar os estados, deixando tudo isso centralizado, sendo utilizado ao mesmo tempo por todos os componentes de forma compartilhada. Ele também roda em diferentes ambientes como servidor, cliente e nativo.

Redux

Fazendo o uso do Redux todos esses estados ficarão armazenados em uma árvore de objetos através do store. Para que isso aconteça, o Redux utiliza 3 recursos:

- **Store:** você pode pensar em store como um container ou um grande centro de informações, que tem disponibilidade para receber e entregar o que o seu componente requisita. A store armazena de forma centralizada todos os estados da aplicação. Vale ressaltar que a store é imutável.
- **Actions:** São ações disparadas da aplicação para o store. Elas são criadas através das action creators. As actions são a única forma de acionar uma mudança de estados no store.
- **Reducers:** Cada dado da store deve ter o seu próprio reducer. Ele é encarregado de lidar com todas as ações e especificam como o estado da aplicação irá mudar de acordo com a action que foi enviada para o store.

Redux

O fluxo geralmente funciona da seguinte forma: um componente gera uma interação através de um clique dado pelo usuário na interface - por exemplo, assim, um action creator é acionado e dispara uma ação para o store. Essa ação chega até um reducer que irá processar e fazer a alteração do estado no store. Assim um novo estado será disponibilizado para o componente.

Redux

Também é importante saber que o Redux tem 3 princípios, sendo:

- **Todos os estados estarão disponíveis exclusivamente através do store:** todo o estado da sua aplicação vai estar armazenado nesse store que é único, onde todos os componentes vão consultar nesse store.
- **Os estados são somente leitura:** os componentes não podem fazer uma manipulação direta nas informações que estão nele.
- **As alterações são feitas através de funções puras:** o Redux utiliza o conceito de programação funcional, por isso toda alteração no store é feita através de uma função pura, chamada de reducer. O reducer recebe o estado e a ação, onde com essa ação nós visualizamos o que precisa ser alterado no estado e o reduce entrega uma nova store do nosso estado da aplicação.

Redux

Por ser uma biblioteca o Redux pode até ser utilizado sozinho, mas ele é normalmente implementado em um conjunto de outras libs ou frameworks JavaScript (Ember, Vue, Angular...). Mas o comum mesmo, que você até já deve ter visto, é vê-lo em funcionamento com o React. É muito comum ver projetos e exemplos de React com Redux juntos, mas é bom deixar claro que o Redux não depende do React.

React Hook Form

React Hook Form é uma biblioteca que auxilia na criação e validação dos formulários, além de reduzir a quantidade de código desenvolvido, fazendo com que a captura de ações do formulário também seja mais objetiva. Outra facilidade que ele traz é na melhora significativa de desempenho, já que a ação de renderização também pode ser controlada. Dessa forma, apenas as alterações de entradas são rerrenderizadas, não o formulário inteiro.

React Hook Form

Criando validação de formulário simples, o React Hook Form, alinhado com os existentes dentro do próprio HTML, suporta as validações: required, min, max, maxlength, minlength, pattern, validate.

React Hook Form

Register: Responsável por registrar o valor do input e atribuir ao seu hook form.

formState: Salva o estado do input do formulário, muito usado para retornar os erros de validação, por exemplo.

watch: Quando declarado e atribuído para um input, retorna apenas o valor do input esperado.

handleSubmit: É o método responsável por manipular ou lidar com o submit do form.

reset: Responsável por apagar ou resetar o state do hook form.

useController: É um hook que funciona de forma similar a Controller, por meio do qual é possível passar as props diretamente para os campos do formulário, conforme o exemplo da documentação:

Yup

Yup é um construtor de esquema JavaScript para análise e validação de valor. Assim, com seu esquema de validação poderoso e simples de usar, o Yup possibilita uma maneira abstrata que não interfere no restante da lógica de negócio.

Dessa forma, é possível criar um objeto formatado que assemelha-se com o esquema pretendido para um objeto. E em seguida, utilizar as funções deste utilitário para verificar se nossos objetos de dados correspondem a esse esquema. Para, assim, validá-los.

Afinal, vale ressaltar que podemos trabalhar em validações de qualquer objeto na aplicação. Tal qual podem ser utilizados dados de formulários no Front-end ou até no Back-end, dentre outras utilidades.

Yup

Alguns tipos de validação yup ?

- **string.required:** Esperar algum valor. Assim, strings vazias são consideradas vazias.
 - **string.matches:** Permite fazer comparações principalmente utilizando regEx.
 - **string.email:** Valida se é um endereço de email.
- Outras duas muito utilizadas, mas que eu não coloquei no exemplo, são:
- **string.min:** define o tamanho mínimo da string.
 - **string.max:** define o tamanho máximo da string

React Hook Form

A validação de formulário facilita a aplicação de diversas funcionalidades do sistema. Bem como melhora a usabilidade e experiência do usuário. Então, neste artigo, aprendemos como fazemos a validação de formulário com React Hook Form e Yup.

Teste Unitarios

São testes que verificam se uma parte específica do código, costumeiramente a nível de função, está funcionando corretamente. Em um ambiente orientado a objetos é usualmente a nível de classes e a mínima unidade de testes inclui construtores e destrutores.

Qual é o Propósito dos Testes Unitários?

Muitos dizem que o objetivo dos testes unitários é validar que cada unidade de trabalho se comporta como projetada, esperada ou pretendida. E que você pode executar seus testes unitários toda vez que alguém faz uma mudança. Com apenas o apertar de um botão.

Mas o verdadeiro propósito do teste unitário é fornecer-lhe feedback quase instantâneo sobre o projeto e a implementação de seu código.

A criação de testes unitários, ou melhor, a facilidade ou dificuldade com que você pode criá-los lhe diz exatamente o quão testável é seu código. O quão bem você o projetou. E se você escutá-los, ao invés de fazer gambiarras para superar qualquer dificuldade, todos ganham.

Quais São os Benefícios dos Testes Unitários?

Os testes unitários proporcionam muitos benefícios.

- **Você economiza muito tempo nos testes de regressão.**
- **Você recebe um alerta sempre que, inadvertidamente, quebra um comportamento existente. Permitindo que você o enfrente imediatamente enquanto ainda está totalmente imerso no que você está trabalhando. Significa menos bugs escapando.**
- **Menos bugs escapando significa que você tem mais tempo para desenvolver valor.**
- **Trabalhando sem medo de quebrar o código existente sem saber, você possui mais recursos cognitivos preciosos. Significa que você será mais criativo e inovador, e capaz de criar soluções melhores.**

Quais São os Benefícios dos Testes Unitários?

- **Você recebe documentação viva, respirando, nunca desatualizada – pelo menos quando você dá a cada teste unitário um nome significativo (mais sobre isso depois).**
- **Uma documentação sempre atualizada permite que você acelere os novos contratados.**
- **Trabalhando sem medo de quebrar o código existente sem saber, novos membros da equipe se tornam produtivos mais rapidamente.**
- **Menos bugs também significa que você reduzirá a carga da equipe de suporte e os liberará para se concentrar no sucesso do cliente ao invés de controlar os danos.**

Como Escrever Testes Unitários (Usando as Melhores Práticas)

- **Dê nomes de seus métodos de teste que o ajudem a entender os requisitos do código que você está testando sem ter que procurar em outro lugar. Escolha um dos vários modelos experimentados e testados que existem para isso e mantenha-se fiel a ele.**
- **Certifique-se de que um teste só tenha sucesso porque o código que ele testa está correto. Da mesma forma, assegure-se de que um teste só falha porque o código que ele testa está incorreto. Qualquer outra razão para o sucesso ou fracasso é enganar a si mesmo ou perseguir um buraco de coelho.**

Como Escrever Testes Unitários (Usando as Melhores Práticas)

- **Faça um esforço para criar mensagens curtas e significativas de falha que incluam parâmetros de teste relevantes. Há poucas coisas mais frustrantes do que “Esperado 5, mas encontrado 7” e ter que caçar o valor dos parâmetros para o método em teste.**
- **Certifique-se de que cada teste possa produzir os resultados corretos (sucesso ou falha) mesmo quando for o único teste que você executar.**

Armadilhas Comuns em Testes Unitários

Erros simples podem te fazer tropeçar seriamente. Pior ainda, eles podem te acalmar com uma falsa sensação de segurança.

Estes são os erros que você quer evitar

- **Escrever testes sem afirmações.**

Tal teste nunca falhará! Se não houver problema porque você está meramente verificando se não há exceções, faça isso explicitamente com uma afirmação e uma mensagem apropriada.

Armadilhas Comuns em Testes Unitários

- **Escrever mais de um teste de cada vez.**

É uma indicação de que você está testando após o fato (em vez de uma abordagem de test-first), e está criando dores de cabeça para você mesmo ao acompanhar quais testes estão completos e devem ser bem sucedidos e quais testes falham por estarem incompletos.

Armadilhas Comuns em Testes Unitários

- **Não começar com um teste reprovado.**

Quando você não começa com um teste reprovado, você não saberá se ele é bem sucedido porque você tem um erro em seu teste ou porque o código funcional está correto.

Armadilhas Comuns em Testes Unitários

- **Não executar testes frequentemente.**

Idealmente, você deseja executar todos os testes unitários em cada etapa de um ciclo vermelho-verde-refatorar, quer você use isso com ou sem uma abordagem de test-first, como TDD e BDD.

Armadilhas Comuns em Testes Unitários

Escrever testes lentos.

Os testes lentos interrompem seu fluxo.

A propósito, não há problema em usar um framework de testes unitários (veja abaixo) para escrever testes (mais) lentos, mas eles não são testes unitários, e você quer mantê-los em um conjunto de testes separado.

Armadilhas Comuns em Testes Unitários

- **Tornar os testes dependentes de seu ambiente de testes.** Isso lhe dará dores de cabeça quando os testes falharem em um ambiente e forem bem-sucedidos em outro.

Jest

Jest é um framework de teste unitário de código aberto em JavaScript criado pelo Facebook a partir do framework Jasmine. Jest é uma das ferramentas de teste unitário mais difundidas dentro da comunidade de JavaScript.

Jest

O mais comum em teste unitário costuma ser o teste de igualdade. O Jest possui matchers de comparação que independem do tipo, estes são:

A função `.toBe(valor)` testa se o valor passado é idêntico ao esperado em valor e tipo.

```
1 | test("resultados devem ser idênticos",  
2 |   () => {  
3 |     let geladeira = produto.findGeladeiraById(12)  
4 |     expect(geladeira.modelo).toBe('Eletrolux')  
5 |   })
```

Jest

A função `.toEqual(valor)` testa recursivamente cada valor do objeto ou array.

```
1 test("resultados devem possuir os mesmo atributos",  
2     () => {  
3         let geladeira = produto.findGeladeiraById(12)  
4         expect(geladeira).toEqual({preco: 1249.99, ano: '2017', modelo: 'Eletrolux'})  
5     })
```

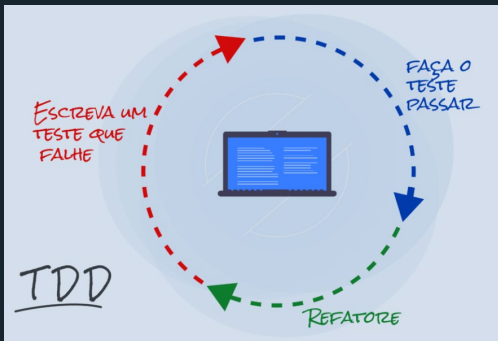
Jest

Para cada matcher de comparação é possível usar o `.not` para fazer uma comparação oposta.

```
1 test("resultados não devem possuir os mesmo atributos",
2     () => {
3         let geladeira = produto.findGeladeiraById(12)
4         expect(geladeira).not.toEqual({preco: 1249.00, ano: '2014', modelo: 'Brastemp'})
5     })
```

TDD

O TDD é considerado uma técnica ou metodologia muito adotada nos times de desenvolvimento.



Como o TDD foi criado?

O TDD e seu significado foram descobertos por Kent Beck em 2003. O engenheiro de software também é um dos pais do XP (Extreme Programming), que vem sendo usado desde 1996. Aliás, é difícil falar em XP sem lembrar do TDD, pois este último é um dos pilares da programação ao extremo.

Beck detalhou a técnica no livro “TDD – Desenvolvimento Guiado por Testes”. Enfim, ele ilustra com exemplos práticos como codificar e executar testes automatizados.

Qual é o ciclo de desenvolvimento do TDD?

Pode parecer que o TDD inverte a ordem das etapas. Mas, ao final, você verá que tudo faz sentido. Sendo assim, há 3 principais fases no ciclo de desenvolvimento desse método. Veja:

- 1.Red:** escrevemos um teste que vai falhar (natural, pois o código ainda não existe);
- 2.Green:** fazemos o ajuste necessário, conforme o resultado esperado, até fazer o teste passar;
- 3.Refactor:** refatoramos o código, retirando duplicidade, renomeando variáveis, usando padrões conhecidos, entre outras ações.

Qual é o ciclo de desenvolvimento do TDD?

Assim, para cada funcionalidade do programa, repetimos o ciclo acima. É por isso que a técnica se encaixa perfeitamente na metodologia XP. Afinal de contas, o TDD e seu significado estão muito presentes em pequenos ciclos de repetições. Ou seja, para cada funcionalidade do sistema, os Devs criam um teste que repete a sequência Red, Green e Refactor.

Quais as principais vantagens do TDD?

Quem não é habituado a codar usando a metodologia ágil do TDD pode se espantar. Mas, na prática, ele traz muitas vantagens. Confira algumas delas:

- código mais limpo, pois escreve-se códigos simples para que o teste tenha resultado positivo;**
- segurança na correção de bugs;**
- segurança no refactoring;**
- maior produtividade para o developer;**
- feedback mais rápido sobre a funcionalidade.**

Qual a diferença entre TDD, BDD e DDD?

Conhecemos esse conjunto de siglas no universo das metodologias ágeis. Elas são grandes aliadas no desenvolvimento de softwares. Existem algumas semelhanças e diferenças entre eles.

TDD

É o desenvolvimento orientado a testes. Primeiro, cria-se o teste conforme o resultado que se deseja atingir para determinada funcionalidade. Depois disso, se aprimora o código.

BDD

É Behavior Driven Development (Desenvolvimento Orientado ao Comportamento), cujos testes são baseados no comportamento do software ao longo da sua vida útil. É um complemento ou ainda uma evolução do TDD.

DDD

É o Domain-Driven Design (Design Orientado pelo Domínio). Ou seja, o problema que ele se propõe a resolver (a regra do negócio) é a parte principal do software.

Por que escolher o TDD?

Como você viu até aqui, o TDD aumenta a produtividade da equipe. Mas a sua implantação exige um novo esforço do time de desenvolvimento que está acostumado com outra abordagem.

Trata-se, portanto, de uma nova cultura. Mas é possível se adaptar. E, logo após a adaptação, não se perde mais tempo com vários códigos, pois se tem um código inicial já testado e em funcionamento, independentemente do número de Devs trabalhando no mesmo projeto.

TDD

Pelo fato de inverter a ordem dos trabalhos – do teste para o código – é um pouco impopular entre os Devs. No entanto, após pegar o jeito, o desenvolvimento ganha um up, e a técnica traz muitos resultados positivos ao projeto.

PUCRS online  **UOL** edtech.