Cap 7 - Escrevendo os testes de acessibilidade com Cypress

No capítulo anterior, vimos como podemos configurar o Cypress e o Cypress-Axe em um projeto de teste.

Continuaremos com o que temos até agora e, em seguida, começaremos a adicionar alguns testes automatizados de acessibilidade em nosso projeto.

Em termos do que você pode esperar deste capítulo, veremos:

1. Como você pode testar uma página inteira com Cypress-Axe.
2. Análise dos logs e mensagens de erro que recebemos para que possamos entender as diferentes falhas de acessibilidade.
3. Teste em elementos específicos na página.
4. Como você pode desabilitar algumas regras do Axe - eu normalmente recomendo não desabilitar nenhuma regra de acessibilidade, dessa forma você pode validar mais regras tanto quanto possível. Mas, em alguns casos pode ser necessário.

7.1 Vamos começar

Todo o código que você verá hoje estará no repositório público do GitHub, que é adicionado como um link de recurso para este capítulo.

Não se preocupe se não entender imediatamente.

Repassarei o código um por um e tentarei explicar da melhor forma para que fique tudo claro para você.

Para este curso, vamos executar alguns testes de acessibilidade em um dos aplicativos de amostra mais populares que existem - o aplicativo **To Do**.

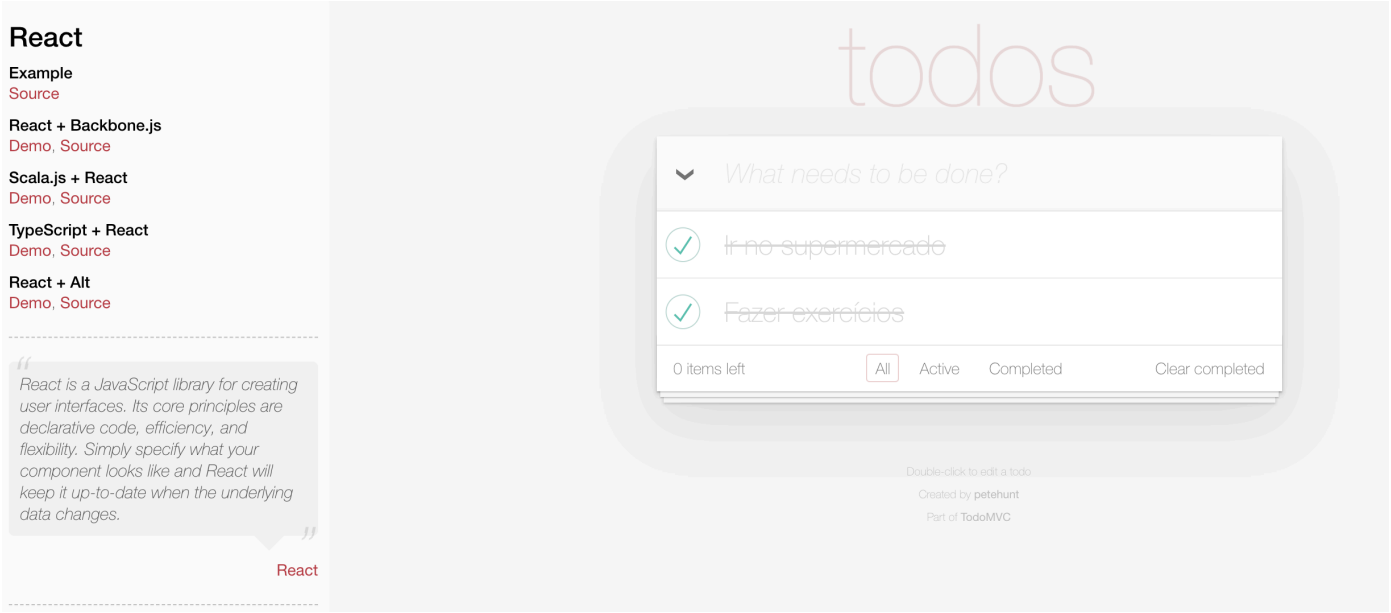
O link para acessar este aplicativo de tarefas pode ser encontrado aqui:

Basicamente, este é apenas um aplicativo simples baseado em React para adicionar itens de tarefas e marcá-los como concluídos.

Portanto, como você pode ver aqui, estou apenas adicionando alguns itens de tarefas pendentes como parte da minha lista.

Assim que terminar, posso marcá-lo como concluído e então posso verificar se tenho algum item ativo ou aquele que marquei como concluído.

Também posso excluí-los se quiser.



Funcionalmente, você pode ver que funciona, mas como este é um curso sobre teste de acessibilidade, vamos escrever alguns testes que testam o quão acessível é o aplicativo React To Do.

Portanto, voltamos ao VS e o que farei é excluir os outros arquivos de teste que não precisamos para este projeto.

Vou adicionar um novo arquivo de teste, que chamarei de *todo-app.spec.js* e, neste arquivo, adicionarei uma importação aos tipos de referência do Cypress.

Agora, o que isso significa é - para que possamos habilitar o IntelliSense para este projeto, temos que adicionar este tipo de referência.

Isso garantirá que possamos acessar todos os diferentes métodos Cypress automaticamente.

Vou adicionar meu *bloco de descrição* e, no primeiro parâmetro, vou dar a ele um nome que chamarei de "Todo Application".

Blocos de descrição no Mocha permite que você agrupe seus testes de uma forma muito mais estruturada.

Se você quiser saber mais sobre a estrutura de teste do JavaScript Mocha, recomendo enfaticamente o curso do Giridhar na Universidade de Automação de Teste.

Agora, vamos escrever nosso primeiro bloco *it*.

Um bloco *it* no Mocha é uma forma de descrever o que é nosso teste.

Ele aceita dois parâmetros - o primeiro é apenas o título do nosso teste e o próximo parâmetro é a função que contém os diferentes comandos do Cypress que precisamos executar.

Então, o primeiro comando que precisamos é **cy.visit**, que é um método do Cypress, para visitar nosso aplicativo.

O próximo comando vem, na verdade, da biblioteca Cypress-Axe e é chamado de **cy.injectAxe**.

Agora o que esse comando faz é realmente injetar o tempo de execução do axe-core em nosso aplicativo de teste, portanto, ele deve ser adicionado depois que você visitar sua página de teste.

O último comando de que precisamos é **cy.checkA11y** ou **cy.checkAccessibility**.

Este comando irá, na verdade, examinar sua página em busca de falhas de acessibilidade.

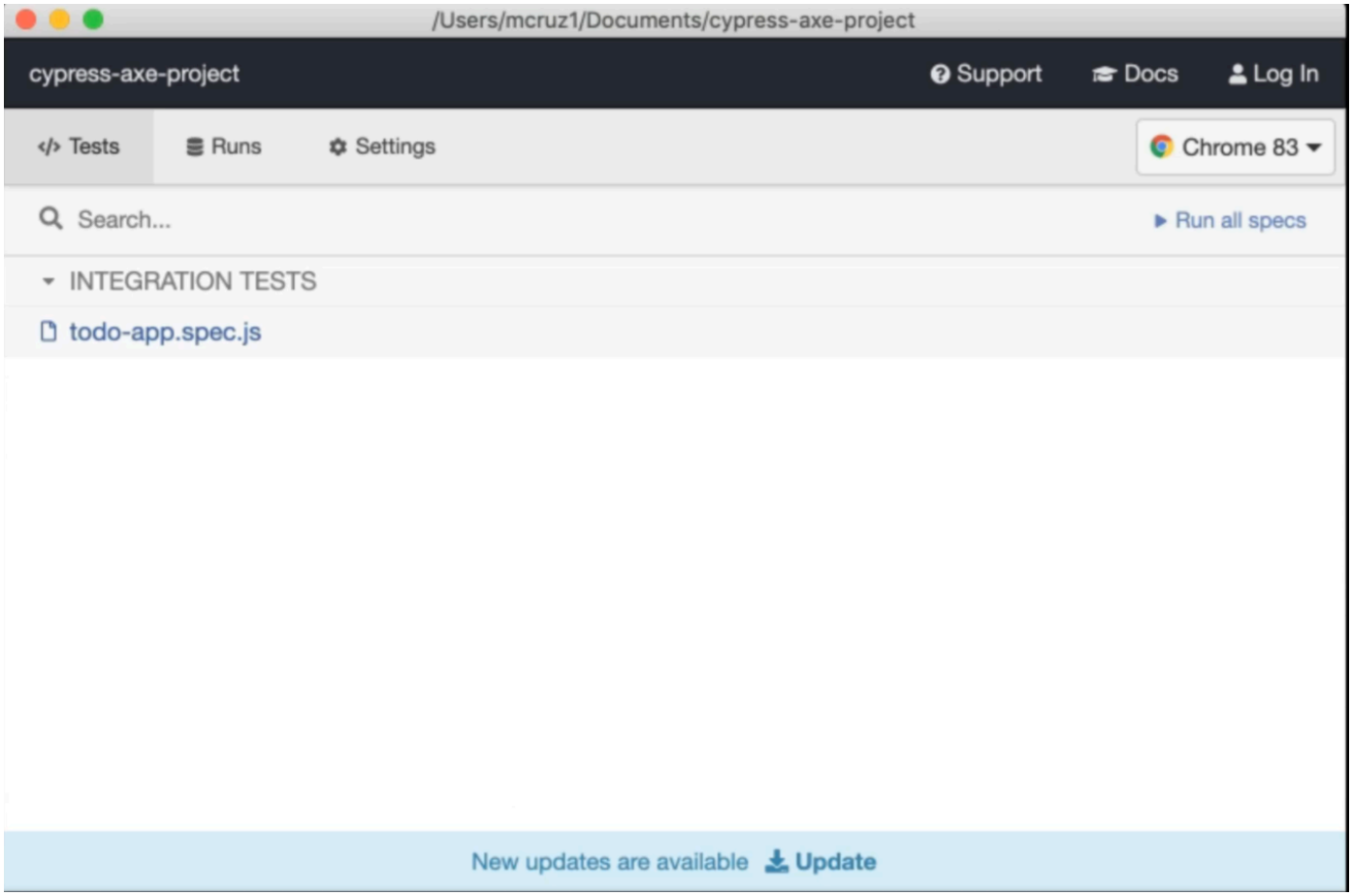
> Nota: onde estamos no projeto:
>
> cypress > integration > new> todo-app.spec.js

```
describe('Todo application', () => {
 it('should log any accessibility failures', () => {
   cy.visit('http://todomvc.com/examples/react');
   cy.injectAxe();
   cy.checkA11y();
 });
});
```

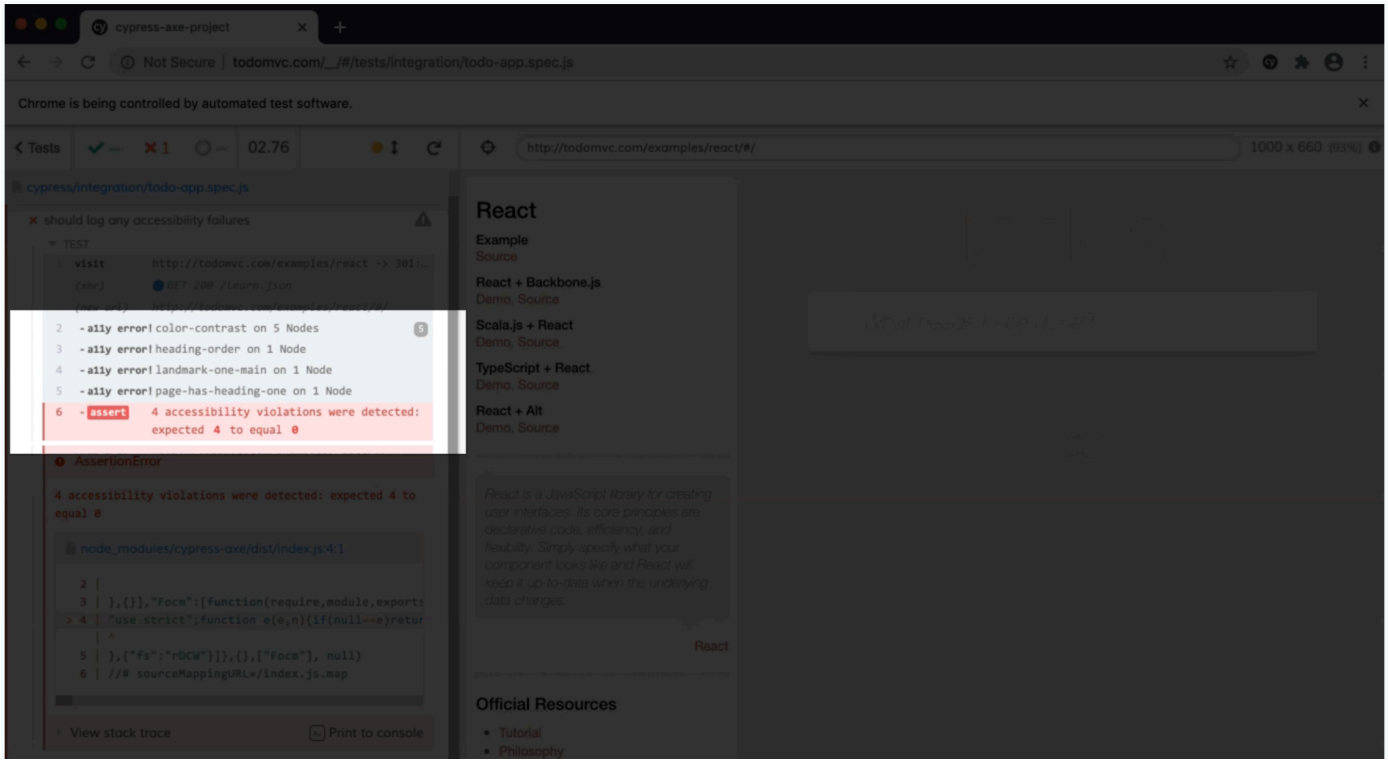
Agora vou apenas executar o comando:

```
npx cypress open
```

Se você se lembra do capítulo anterior, usamos este comando para iniciar o executor de teste Cypress.



Agora que está carregado, vamos prosseguir e clicar no arquivo de teste que escrevemos e esperar que os testes sejam executados.



Pelo que visto acima, o Cypress-Axe detectou 4 erros de acessibilidade.

Existem alguns erros no contraste de cor (color-contrast), alguns erros de acessibilidade na ordem do título (heading-order), ponto de referência principal (landmark-one-main) e também página com título um (page-nas-heading-one).

No próximo subcapítulo, veremos como analisar todas as diferentes regras de violação de acessibilidade que Cypress-Axe nos relatou.

7.2 Entendendo os bugs de acessibilidade

Neste subcapítulo, veremos como podemos analisar as diferentes violações de acessibilidade obtidas pelo uso do Cypress-Axe.

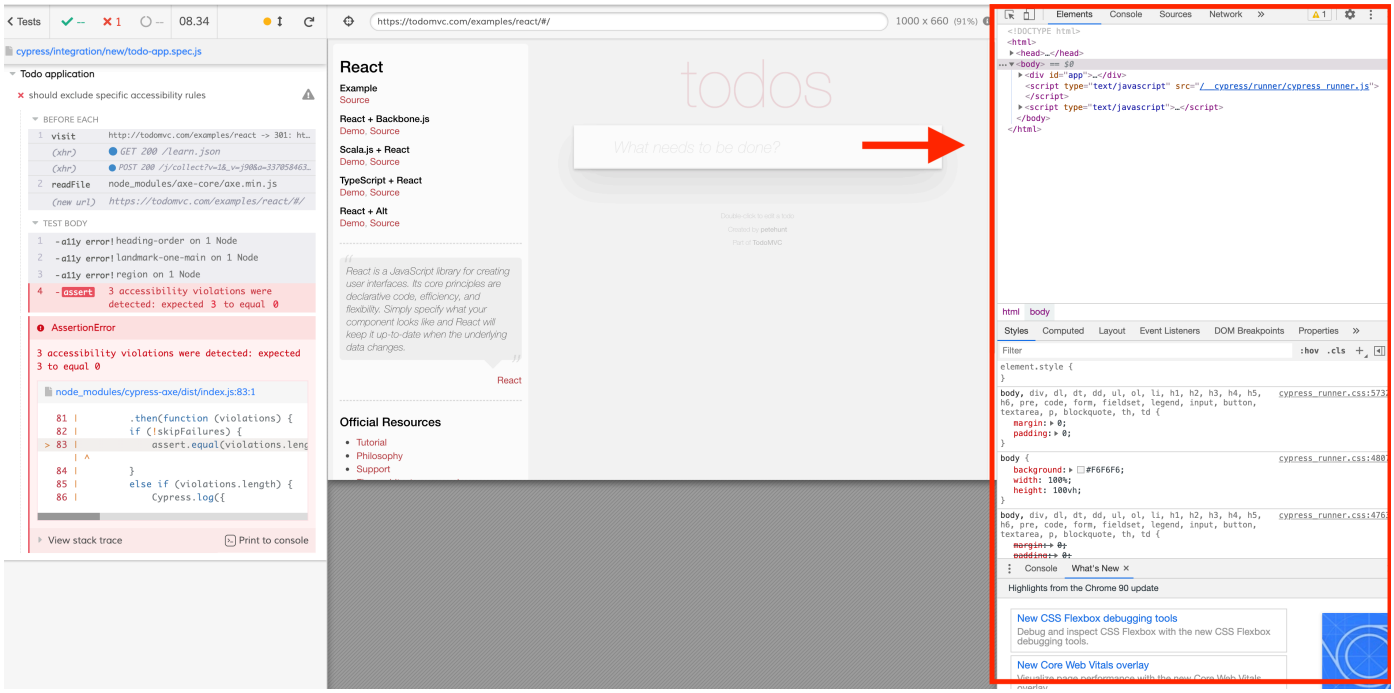
Se você se lembra do capítulo anterior, executamos nossos primeiros testes de acessibilidade no aplicativo React To Do.

Em seguida, fomos notificados de que quatro violações de acessibilidade foram detectadas.

Como testadores, precisamos entender o que essas diferentes violações significam para que possamos explicar para nossa equipe claramente.

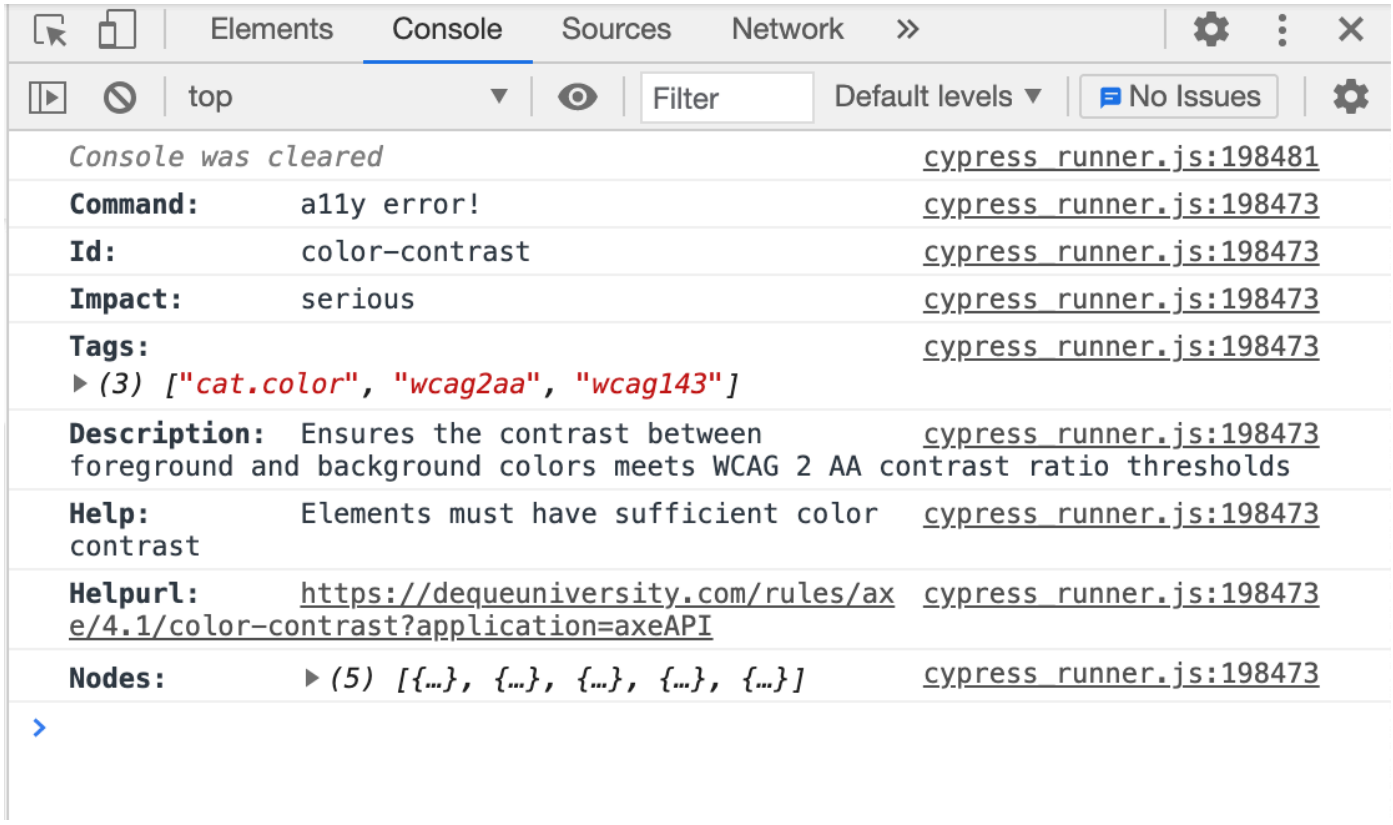
A vantagem do Cypress-Axe é que eles fornecem mais informações sobre essas diferentes violações, e podemos ver isso abrindo nosso console de ferramentas para desenvolvedores, clicando com o botão direito do mouse em qualquer parte do Test Runner e clicando em "Inspecionar".

Isso deve iniciar as ferramentas do desenvolvedor, que você pode ver aqui.



Vamos clicar na violação de acessibilidade relacionada ao **contraste de cor**.

Como você pode notar, após clicar na violação, registros adicionais no console são impressos:



Você pode obter uma descrição do que é a regra de acessibilidade. Nesse caso, trata-se de garantir que haja contraste suficiente entre as cores do primeiro plano e do plano de fundo.

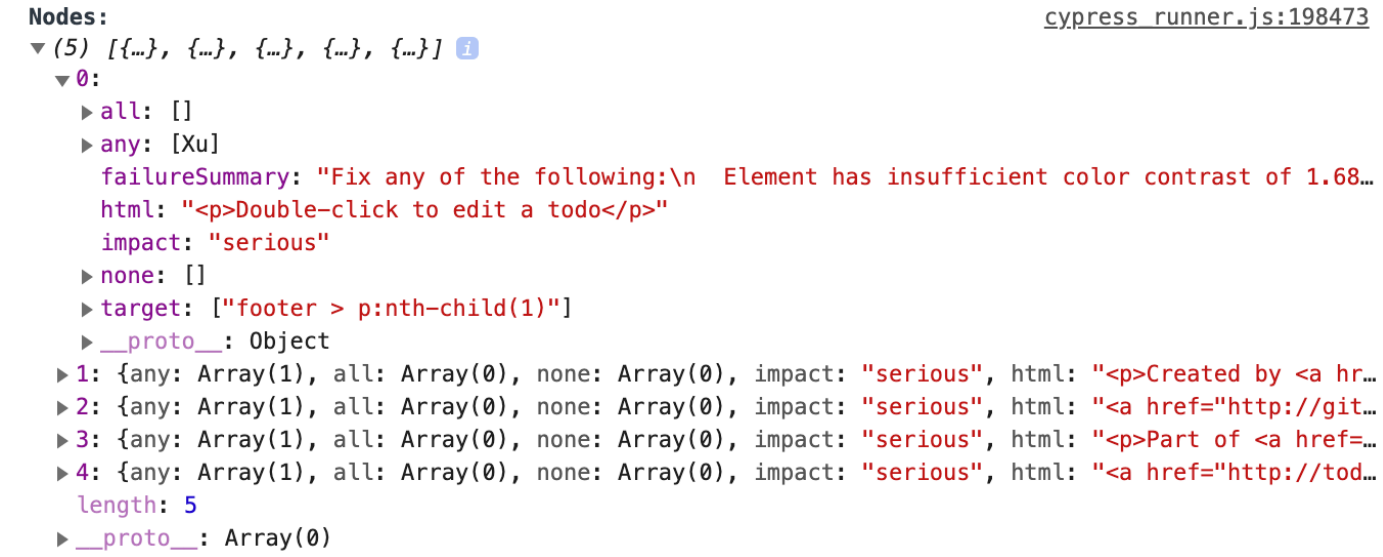
O Axe também fornece uma URL de ajuda, no qual você pode clicar diretamente (***Helpurl***), e isso deve redirecioná-lo para a página de ajuda, que contém informações relevantes, por exemplo, de como corrigir o problema.

Você pode ver qual é o critério de sucesso, e ainda usar um analisador de contraste de cores para ajudá-lo a decidir se suas cores estão acessíveis.

Você também pode encontrar outras informações, como o porquê dessa regra ser importante para pessoas com baixa visão.

Vamos voltar ao nosso teste e ver o que mais pode ser útil.

Este array contém os elementos específicos que violaram a regra de contraste de cores.



Se você expandir um por um, poderá descobrir qual seletor específico precisa ser alterado.

Se olharmos para a próxima regra de violação de acessibilidade, é sobre a **ordem do título**.

A ordem dos títulos é importante e deve ser organizada, pois as tecnologias assistivas usam isso para fornecer a navegação ao usuário.

A próxima regra de acessibilidade é sobre ter um **marco principal** em sua página.

Ter um ponto de referência principal garante que as pessoas com deficiência possam acessar o conteúdo principal de sua página com facilidade.

A última violação da regra de acessibilidade é para garantir que sua página tenha o **título h1**.

Geralmente, é uma prática recomendada que sua página tenha um único título h1, pois isso permitirá que os usuários pule diretamente para o conteúdo principal de sua página.

Em cada violação, você também poderá observar o impacto de cada uma das regras de acessibilidade: o contraste da cor tem um impacto mais sério em comparação com as outras três violações.

Isso deve ajudá-lo a aconselhar a equipe qual regra vocês precisam abordar primeiro e priorizar.

No próximo subcapítulo, veremos como você pode testar um elemento específico em vez da página inteira com Cypress Axe.

7.3 Testendo elemento específicos

Para este subcapítulo, veremos como você pode testar os elementos específicos em sua página de teste com Cypress-Axe.

Por padrão, o Cypress-Axe fará a varredura da estrutura DOM de sua página inteira.

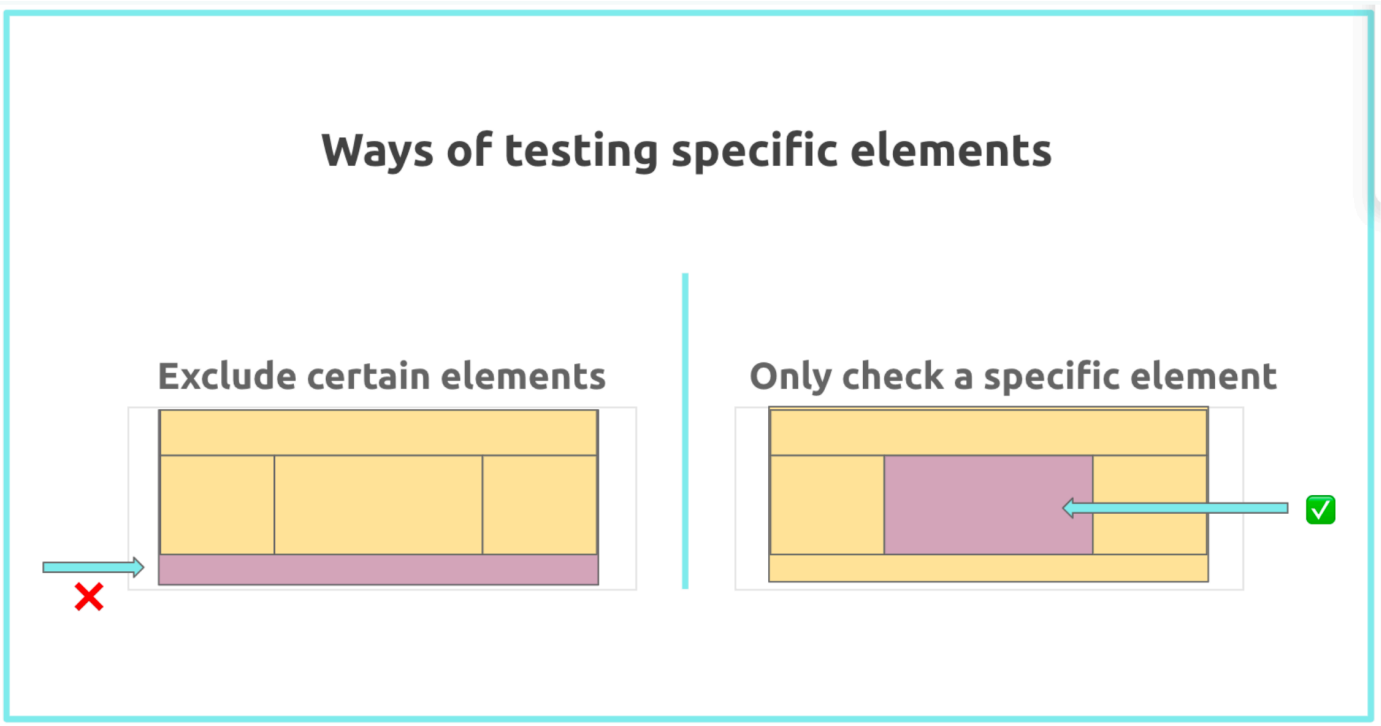
Na maioria das vezes, esse é o melhor a se fazer, já que você deseja que o Axe detecte erros de acessibilidade em toda a sua página.

No entanto, há alguns casos em que alguns elementos de sua página ainda estão sendo trabalhados e precisam da opinião da equipe de design e desenvolvimento sobre como torná-la acessível.

Nesse cenário, para não bloquear o pipeline de teste, você pode escolher ignorar os elementos e o Axe não fará a varredura desses elementos em relação a nenhuma das regras de acessibilidade.

Só use isso se for realmente necessário, e quando excluir elementos específicos, certifique-se de adicionar um comentário de tarefas ou vinculá-lo a um tíquete do JIRA em algum lugar, por exemplo, para ter rastreabilidade.

Existem duas maneiras de testar elementos específicos no Cypress-Axe:



A primeira opção é excluir certos elementos da página.

No primeiro diagrama, você pode ver que estamos excluindo o componente de rodapé, o que significa que quando o Cypress-Axe for executado, ele varrerá a página inteira, exceto o rodapé.

A segunda opção é verificar apenas um elemento específico da página.

Como você pode ver aqui, vamos apenas executar a verificação de acessibilidade no conteúdo principal.

Vamos agora ver como podemos traduzir isso em código.

Se você se lembra do subcapítulo anterior, o Cypress-Axe relatou quatro erros de violação de acessibilidade no aplicativo React To Do.

Vamos voltar ao Visual Studio Code e escrever nosso segundo teste.

Neste teste, vamos excluir os elementos específicos da página.

Apenas para acelerar um pouco as coisas, copiarei os dois comandos cy.visit e cy.injectAxe neste teste.

Em seguida, vou fazer uma chamada para cy.checkA11y, mas desta vez vou passar um objeto com o nome de propriedade **exclude**.

Exclude, aqui, é do tipo array e aceitará uma lista de strings que você deseja que o Cypress-Axe exclua.

```
/// <reference types="cypress"/>

describe('Todo application', () => {
 it('should log any accessibility failures', () => {
   cy.visit('http://todomvc.com/examples/react');
   cy.injectAxe();
   cy.checkA11y();
 });

 it('should exclude specific elements on the page', () => {
   cy.visit('http://todomvc.com/examples/react');
   cy.injectAxe();
   cy.checkA11y({ exclude: ['.learn'] });
 });
});
```

Para este teste, desejo excluir a parte esquerda do aplicativo Tarefas.

Usando o playground do seletor do Cypress, podemos passar o mouse na seção, copiar o nome do seletor fornecido pelo Cypress e colá-lo em nosso código.

Vamos salvar as alterações que fizemos e notar que o Cypress executa automaticamente o teste.

Você pode notar que apenas três erros de acessibilidade foram detectados, em oposição aos quatro de nossos testes anteriores.

O erro na ordem dos títulos não foi relatado porque excluímos a seção onde o erro foi encontrado.

Vamos voltar ao Visual Studio Code e escrever outro teste.

Para este caso, vamos testar um elemento específico em nossa página.

Semelhante ao nosso teste anterior, também copiarei os comandos **cy.visit** e **cy.injectAxe** novamente.

Então, em nossa chamada cy.checkA11y, vamos apenas passar uma string aqui.

Para simplificar, vou usar o mesmo seletor que usamos anteriormente.

```
/// <reference types="cypress"/>

describe('Todo application', () => {
 it('should log any accessibility failures', () => {
   cy.visit('http://todomvc.com/examples/react');
   cy.injectAxe();
   cy.checkA11y();
 });
```

```
  it('should exclude specific elements on the page', () => {
    cy.visit('http://todomvc.com/examples/react');
    cy.injectAxe();
    cy.checkA11y({ exclude: ['.learn'] });
  });

  it('should only test specific element on the page', () => {
    cy.visit('http://todomvc.com/examples/react');
    cy.injectAxe();
    cy.checkA11y('.learn);
  });
});
```

Vamos salvar e observar que temos apenas um erro de acessibilidade, pois estamos apenas testando um elemento específico.

Agora, a última parte que quero fazer é apenas refatorar um pouco esse código, já que temos alguns códigos duplicados.

Vou escrever um **beforeEach**, que será executado antes de cada um de nossos testes e, em seguida, adicionar os comandos **cy.visit** e **cy.injectAxe**.

Agora podemos remover as chamadas duplicadas no resto do nosso código:

```
/// <reference types="cypress"/>

describe('Todo application', () => {
  beforeEach(() => {
    cy.visit('http://todomvc.com/examples/react');
    cy.injectAxe();
  });

  it('should log any accessibility failures', () => {
    cy.checkA11y();
  });

  it('should exclude specific elements on the page', () => {
    cy.checkA11y({ exclude: ['.learn'] });
  });

  it('should only test specific element on the page', () => {
    cy.checkA11y('.learn');
  });
});
```

No próximo subcapítulo, vou mostrar como excluir as regras de acessibilidade específicas em seus testes.

7.4 Disabilitanto as regras de acessibilidade

Para este subcapítulo, veremos como você pode desabilitar algumas regras de acessibilidade com Cypress-Axe.

Semelhante ao subcapítulo anterior, pode haver alguns elementos em sua página que ainda estão sendo trabalhados e precisam da contribuição da equipe de design e desenvolvimento.

Nesse caso, uma ideia é desabilitar algumas das regras de acessibilidade e adicionar um comentário de tarefas ou vinculá-lo a um tíquete do JIRA.

Um outro cenário é, se você estiver fazendo testes de acessibilidade no nível do componente, e digamos que tenha um componente de texto que não possui uma tag de título h1.

Você pode desabilitar a regra **page-has-heading-one**, então esta regra será excluída, pois alguns componentes nem sempre terão uma tag de título h1.

Novamente, meu conselho é usar isso com cautela.

Sempre consulte e informe sua equipe antes de desabilitar as regras de acessibilidade.

Como apenas cerca de 20 a 40% dos erros de acessibilidade são detectados por tarefas automatizadas, precisamos testar nosso aplicativo em relação a todas as regras, se aplicável.

Agora veremos como você pode desabilitar algumas das regras.

No entanto, antes disso, vamos recapitular rapidamente o que fizemos:

1. Escrevemos dois testes adicionais que cobriram como excluir ou testar um elemento específico na página React To Do.
2. Para excluir elementos, tudo o que você precisa fazer é passar um objeto com a propriedade **exclude** e passar a lista de elementos que você deseja excluir como um array.
3. Para testar um elemento específico, tudo o que você precisa fazer é apenas passar a string do seletor que deseja testar.

Agora, vamos adicionar um novo teste para relatar apenas violações que tenham um impacto sério ou crítico.

Vamos imaginar que, neste momento a equipe trabalhará para encontrar violações de acessibilidade com gravidade mais alta e que as violações com gravidade moderada serão detectadas numa próxima etapa.

Para fazer isso, basta chamar **cy.checkA11y** novamente.

Para o primeiro parâmetro, vou passar **null** porque não vamos excluir nenhum elemento específico da página.

Então, para o segundo parâmetro, vou passar nosso objeto de opções aqui.

O Cypress-Axe tem uma opção **includedImpacts**, que aceita uma matriz de strings que mapeia para um nível de gravidade.

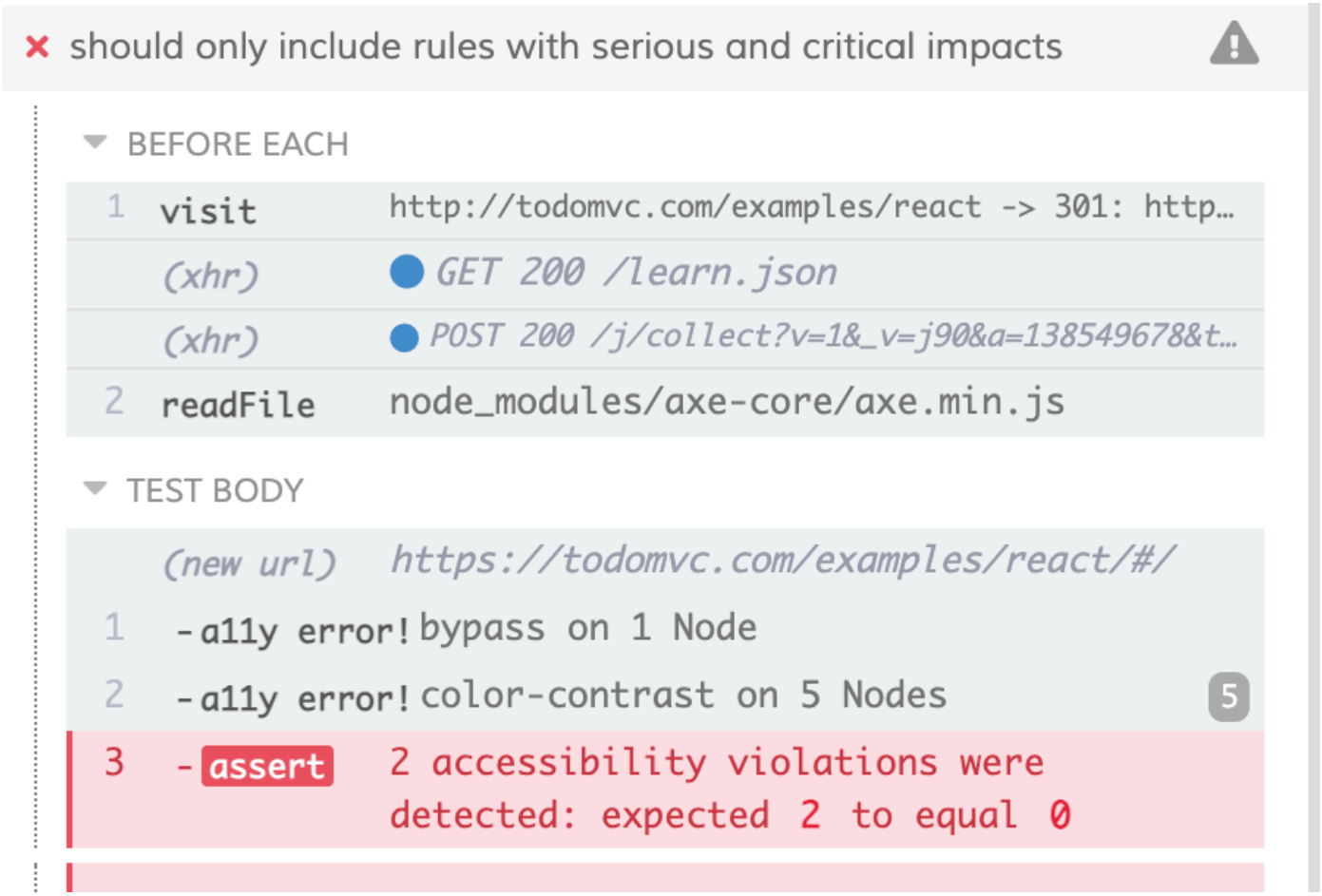
Para este teste, vamos incluir apenas o impacto sério e crítico.

```
it('should only include rules with serious and critical impacts', () => {

   cy.checkA11y(null, { includedImpacts: ['critical', 'serious'] });

 });
```

Se voltarmos ao executor de teste Cypress, observamos que há duas violações - aqui estão as que têm impacto sério ou crítico:



Quaisquer outras regras não foram relatadas porque foram excluídas pelo Cypress-Axe.

Isso é particularmente útil se você já está trabalhando em um projeto existente e deseja priorizar primeiro quais violações precisam ser corrigidas.

Em nosso próximo teste, mostraremos como excluir as regras específicas, independentemente de seu impacto.

Para fazer isso, é semelhante ao teste anterior.

Primeiro, precisamos fazer uma chamada para **cy.checkA11y**, e então também passaremos **null** porque não queremos excluir elementos específicos.

Então desta vez no parâmetro **options**, vamos passar a opção de configuração das regras.

Aqui, você só precisa passar o nome da regra que deseja desativar e definir a propriedade ativada como falsa.

Portanto, neste exemplo específico, vou apenas desativar o **contraste de cor** e, em seguida, continuar e adicionar uma única palavra-chave aqui, para que nosso executor de teste apenas pegue este teste.

```
it('should exclude specific accessibility rules', () => {
   cy.checkA11y(null, {
     rules: {
       'color-contrast': { enabled: false },
     },
   });
 });
```

Agora, se você voltar ao executor de teste, o contraste de cor foi desativado.

Esperamos que você tenha achado este vídeo relevante e tenha dado uma visão geral de como o Cypress-Axe é flexível quando se trata de seus requisitos de acessibilidade.

No próximo capítulo, veremos como usar o consultor de contraste do Applitools para detectar alguns de nossos problemas de acessibilidade de contraste de cor.

Links

- **Github links**
  - Git repo for source code
  - Branch for this chapter
- **Aplicação**
  - Todo MVC app
- **TAU - cursos relevantes**
  - Mocha Course