# Roteiro de exercícios sobre Testes, disciplina Engenharia de Software e Sistemas

## Paulo Borba

## Centro de Informática

## Universidade Federal de Pernambuco

Adicione as respostas imediatamente após as perguntas, usando uma outra cor. Leia com calma e atenção as instruções, e reflita com cautela sobre as consequências de cada comando. **As instruções abaixo assumem que toda a parte de instalação do roteiro de SaaS foi concluída**. Em particular, se você estiver usando o Windows, tem que ajustar (no arquivo **package.json**) o caminho do **tsc**; ele tenta chamar **./node\_modules/.**, mas, no Windows, tem que descobrir o endereço onde o **tsc** está instalado e ajustar no **package.json** (invertendo as barras do PATH, já que essa é uma diferença importante do unix para o Windows). É também assumido que todas as referências (manuais, tutoriais, etc.) mencionadas nos roteiros anteriores já foram consultadas, e que os alunos devem estar prontos para rapidamente consultá-las novamente durante o resto do curso, inclusive quando seguindo o roteiro abaixo. Os *commits* mencionados abaixo não são referentes ao que está no *master* do repositório do projeto; eles são referentes ao que está no ***branch* criado pelo professor e associado à essa edição da disciplina (por exemplo, 2019-2, 2020-1, etc.)**

## Aula 1

1. No projeto [**teachingassistant**](https://github.com/pauloborba/teachingassistant/commits/master), analise o *commit* "criacao da feature e testes de alunos". Comece pelo arquivo **alunos.feature**, que contém apenas a descrição de um cenário, mas poderia ter vários. Coloque em execução a versão do sistema obtida ao final do roteiro anterior. Manualmente, execute via a GUI do sistema os passos desse cenário. Cole abaixo uma imagem do sistema em execução após a execução desses passos.
2. Para evitar essa execução manual, precisamos indicar que código deve ser executado para cada passo do cenário que queremos automatizar. Analisando o arquivo **alunos.ts**, como você acha que é feita a associação dos passos do arquivo de *feature* com o código desse arquivo de implementação dos passos? Consulte a documentação do [Cucumber.js](https://github.com/cucumber/cucumber-js) (software que faz a ligação entre os cenários e o código), especialmente a seção *Step Definitions*. Consulte a referência de [Javascript](https://www.w3schools.com/jsref/jsref_obj_regexp.asp) para expressões regulares. Qual o significado do **\d** e do **[^\"]** nas expressões regulares desse arquivo? Os **\*** usados nas expressões regulares desse arquivo são realmente necessárias? Como são instanciados os parâmetros da função que é passada como segundo parâmetro do **When**? O que determina a quantidade de parâmetros dessa função?
3. Para facilitar a implementação dos passos, usamos [Protractor](http://www.protractortest.org/#/api). Consulte a documentação da sua **API** (application programming interface), especialmente as funções associadas à variável **browser** exportada pelo Protractor. Que declaração do arquivo **alunos.ts** nos dá acesso a essa variável? Qual o comportamento da função **getTitle**? Na parte da API relacionadas a **element** (que representa elementos de páginas HTML), temos explicações sobre funções como **$** e **click**. Qual o comportamento da expressão **$("a[name='alunos']").click()**?
4. Para facilitar a verificação de restrições a serem satisfeitas pelos testes, usamos a API [chai](http://chaijs.com/plugins/chai-as-promised/). Qual o significado da expressão abaixo

**expect(browser.getTitle()).to.eventually.equal('TaGui')**?

Você acha que ela é realmente necessária? O teste seria de melhor qualidade sem essa linha na implementação do passo?

1. Atualize o seu repositório com o conteúdo dos *commits* até"criacao da feature e testes de alunos" (execute vários *cherry-picks*, um para cada *commit* até o *commit* mencionado na questão, para facilitar a resolução de possíveis conflitos). Instale as dependências necessárias digitando

**npm install**

no diretório **tests-acceptance**. Atualize o *driver* do *browser* digitando

**npm run webdriver-update**

Em um terminal, execute o *driver*digitando (é assumido que há uma versão de Java instalada na sua máquina)

**npm run webdriver-start**

Que arquivo do *commit* "criacao e configuracao do projeto de testes de aceitacao" define esses comandos associados ao **webdriver**? Em dois outros terminais ou abas do mesmo terminal, execute o servidor e o cliente (como nos roteiros anteriores). Em outro terminal ou aba, no diretório **tests-acceptance,** compile e execute os testes digitando

**npm test**

Cole logo abaixo a imagem do relatório da execução. Que arquivo do *commit* "criacao e configuracao do projeto de testes de aceitacao" define esse comando **test**? Caso o Firefox da sua máquina esteja desatualizado, atualize-o ou instale uma nova versão. Caso haja mais de uma versão do Firefox instalada na máquina (por exemplo, nos laboratórios, após instalar uma versão nova com o cinstala, que coloca o executável em ~/.local/bin) altere o arquivo **tests-acceptance/config/config.ts** para que ele contenha o seguinte:

**capabilities: {**

**browserName: 'firefox'**

**}**

Para o uso do chrome, você teria que colocar algo como a seguir, mas não recomendamos.

**capabilities: {**

**browserName: chrome,**

**version: ''versao\_do\_novo\_chrome\_instalado",**

**chromeOptions: {binary: 'pathdonovochromeinstalado'}**

**}**

1. Qual o propósito das alterações trazidas nos arquivos HTML pelo *commit* "adicao de nomes nos elementos HTML a serem referenciados pelos testes"? É correto fazer essas mudanças? Elas mudam o comportamento do sistema e o que é renderizado pelo *browser*? Sem essas mudanças o teste mencionado nas questões anteriores **passaria** (seja executado com sucesso)?
2. Consultar as opções de configuração do [selenium](https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities). Você acha que a implementação do segundo **Given** está correta? Por que? Que nova parte da API do Protractor você teve que ler para entender o funcionamento do código desse passo? Por que você acha que foi necessário usar **Promise.resolve** no último comando da implementação desse passo?
3. Para evitar a dependência com o texto do botão, que expressão você usaria, ao invés de **element(by.buttonText('Adicionar'))**, no último comando da implementação do **When**? Por que você acha que seria interessante evitar essa dependência?

## Aula 2

1. Analise o *commit* "criacao e configuracao do projeto de testes de aceitacao". Que declarações nos arquivos de configuração indicam a localização dos arquivos de *features* e *steps* de testes? Por que você acha que o projeto de testes foi colocado em um diretório diferente dos existentes no projeto?
2. Analise o *commit* "criacao da feature e testes de alunos". Cada comando utilizado para implementar um passo de um cenário tem um dos seguintes propósitos: (a) simular ações no *browser*, (b) consultar e processar o conteúdo do que é mostrado no browser, e (c) verificar se esse conteúdo satisfaz determinadas restrições. No código do primeiro **Given**, temos comandos com quais dos propósitos acima? Você acha isso correto? Por que? E no **When**, temos comandos com quais dos propósitos acima? E no **Then**, temos comandos com quais dos propósitos acima? Seria correto termos, em um **Then**, comandos que simulam ações no *browser*? Por que?
3. Uma forma alternativa de implementar o **Then** do último passo seria com um código similar ao do segundo passo, apenas trocando 0 por 1. Qual das alternativas (a que acabamos de propor e a que temos no código) você acha que impõe mais restrições no comportamento do sistema? Qual você acha que seria mais adequada? Por que?
4. Analise o *commit* "novo cenario de cadastro de aluno com CPF existente". Você considera os dois cenários **independentes**, no sentido que a execução de um não tem nenhum efeito na execução do outro? Caso o CPF utilizado no segundo cenário fosse "683", você consideraria os dois cenários independentes? Por que? Por que a independência é importante entre os cenários e passos de teste?
5. Analise o *commit* "implementacao do cenario de cadastro de aluno com CPF duplicado". Qual o propósito da modificação do arquivo HTML? Caso um desenvolvedor fosse responsável por criar o arquivo HTML, e outro fosse responsável por implementar o teste, qual seria a interface entre as duas tarefas? Por que, no **Given** adicionado, são invocados tanto comandos de criação de um aluno quanto de verificação se há um determinado aluno cadastrado?
6. Qual a principal melhoria no código de teste feita no *commit* "extracao de criarAluno nos testes de aceitacao"? Qual a principal melhoria no código de teste feita no *commit* "refatoracao dos testes de aceitacao"?
7. Implemente um teste de aceitação de GUI para a funcionalidade de remoção de alunos do roteiro anterior.
   1. Cole o teste logo abaixo.
   2. Verifique se o seu teste está seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
   3. Execute os testes e cole logo abaixo a imagem do relatório da execução.
8. **(Opcional)** Implemente um teste de aceitação de serviço para a funcionalidade de remoção de alunos do roteiro anterior.
   1. Cole o teste logo abaixo.
   2. Verifique se o seu teste está seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
   3. Execute os testes e cole logo abaixo a imagem do relatório da execução.
9. **(Opcional)** Implemente os testes correspondentes aos cenários de GUI para o cadastro de auto-avaliação especificados por você no roteiro de requisitos.
   1. Cole os testes logo abaixo.
   2. Verifique se os seus testes estão seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
   3. Execute os testes e cole logo abaixo a imagem do relatório da execução.
10. **(Opcional)** Implemente os testes correspondentes aos cenários de GUI para a visualização de discrepâncias de auto-avaliação especificados por você no roteiro de requisitos.
    1. Cole os testes logo abaixo.
    2. Verifique se os seus testes estão seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
    3. Execute os testes e cole logo abaixo a imagem do relatório da execução.

## Aula 3

1. Consulte a documentação do Jasmine ([com promises](https://github.com/matthewjh/jasmine-promises)) e do [request-promise](https://github.com/request/request-promise#cheat-sheet) e analise o código do *commit* "testes de servidor". Qual o comportamento de **beforeAll** e **beforeEach** no arquivo de teste? Que método HTTP é invocado no teste "só cadastra alunos"? Como isso é especificado (já que não é chamado nenhum método no objeto **request**)? No teste "não cadastra alunos com CPF duplicado", qual seria o comportamento caso as requisições não tivessem sido encadeadas com o **then**?
2. Atualize o seu repositório com o conteúdo dos *commits* relativos a testes de serviço (até o *commit* "testes de servidor"; note que a ordem dos commits não corresponde exatamente a ordem das questões, esse, por exemplo, aparece depois de commits que só vão ser usados na próxima aula). Instale as dependências necessárias digitando **npm install** no diretório **ta-server**. No mesmo diretório, compile e execute os testes digitando **npm test.** Cole logo abaixo a imagem do relatório da execução.
3. Qual o efeito de **expect(err).toEqual(null)** nos blocos *catch* dos testes do arquivo [ta-server.spec.ts](https://github.com/pauloborba/teachingassistant/commits/2019-2#diff-594cceae3b8a3b25e4dc455337c8627b)?
4. Com a versão atual do servidor, o teste "só cadastra alunos" passará ou falhará? Por que?
5. Implemente dois testes de serviço relacionados aos cenários de remoção de alunos.
   1. Cole os testes logo abaixo.
   2. Verifique se os seus testes estão seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
   3. Execute os testes e cole logo abaixo a imagem do relatório da execução.
6. **(Opcional)** Implemente dois testes de serviço relacionados aos cenários do cadastro de auto-avaliação.
   1. Cole os testes logo abaixo.
   2. Verifique se os seus testes estão seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).

Execute os testes e cole logo abaixo a imagem do relatório da execução.

## Aula 4

1. Analise o *commit* "ajustes na configuracao para testes de unidade" e explique logo a seguir como são especificadas as características (nomes, diretórios, etc.) dos arquivos de teste a serem executados.
2. Analise o *commit* "testes de unidade" (deveria ser testes de classe!) e tente entender os testes de unidade criados. Consulte a documentação do Jasmine ([básico](https://jasmine.github.io)). Qual o efeito do **beforeEach** e porque ele é necessário nesse caso? O que aconteceria caso **cadastro** tivesse sido inicializada na sua declaração, ao invés de no **beforeEach**? Os três casos de teste apresentados são dependentes? Por que?
3. Atualize o seu repositório com o conteúdo dos *commits* relativos a testes de classe (até o *commit* "testes de unidade"). Instale as dependências necessárias digitando **npm install** no diretório **ta-server**. No mesmo diretório, compile e execute os testes digitando **npm test.** Cole logo abaixo a imagem do relatório da execução.
4. Que mais testes de unidade deveríamos ter para obtermos uma maior **cobertura** da classe de cadastro de alunos? Implemente um desses testes, execute-o e cole o resultado da execução logo abaixo.
5. Qual a principal vantagem da modificação do *commit* "refatoracao dos testes, cadastro de alunos"?
6. Qual a principal vantagem da refatoração do *commit* "refinamento do teste de CPF duplicado, e refatoracao"? A modificação feita no último caso de teste, altera o seu comportamento? O teste resultante é melhor ou pior do que o anterior?
7. Implemente dois testes de classe indiretamente relacionados aos cenários de remoção de alunos.
   1. Cole os testes logo abaixo.
   2. Verifique se os seus testes estão seguindo rigorosamente as recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto).
8. Qual a principal diferença entre os testes "não aceita alunos com CPF duplicado" (no arquivo [cadastrodealunos.spec.ts](https://github.com/pauloborba/teachingassistant/commits/2019-2#diff-eeffe3b15e644bf1dfd4c5eb1b78e7fa) do *commit* do teste de classe) e "não cadastra alunos com CPF duplicado" (no arquivo [ta-server.spec.ts](https://github.com/pauloborba/teachingassistant/commits/2019-2#diff-594cceae3b8a3b25e4dc455337c8627b) do *commit* de teste de serviço)? Isso caracteriza bem a diferença entre testes de classes e testes de serviço? Por que? Por que você acha que o desenvolvedor optou por escrever os dois tipos de teste?
9. **(Opcional)** Solicite, a um colega de turma, três testes (um de aceitação, um de serviço, e um de classe) implementados por ele como resposta dos exercícios anteriores.
   1. Cole os testes logo abaixo.
   2. Identifique, e liste logo abaixo, quais recomendações para escrita de testes com qualidade (mencionadas nos *slides* e no livro texto) não foram seguidas nos testes especificados pelo seu colega.
10. Implementar parte dos testes do seu projeto: aceitação (um para cada cenário), de classe (pelo menos um, mas busque a quantidade suficiente para testar os aspectos mais relevantes da funcionalidade associada), e de serviço (pelo menos um, mas busque a quantidade suficiente para testar os aspectos mais relevantes da funcionalidade associada)