Boas práticas de Mercado

Introdução

Bem-vindos à aula sobre “Boas Práticas de Mercado” na disciplina Programação II. Nesta jornada, exploraremos aspectos cruciais que transcendem a codificação, enfatizando práticas que potencializam não apenas nosso projeto colaborativo, mas também aprimoram seu portfólio e perfil profissional. Abordaremos a preparação para testes unitários, destacando sua importância na detecção proativa de falhas e na garantia da qualidade do software. Seguiremos com o versionamento de código, discutindo como organizar e documentar suas contribuições de maneira eficaz. Ademais, orientaremos sobre a construção de um portfólio impactante, essencial para evidenciar suas habilidades e trajetória como desenvolvedor. Ao fim, delinearemos os passos subsequentes na sua carreira de programação, encorajando a contínua aprendizagem e evolução.

Preparação para testes unitários

Neste módulo final da nossa disciplina, vamos nos dedicar às boas práticas de mercado. Ao longo das aulas, discutimos várias dessas práticas, mas agora consolidaremos o essencial, aplicando esses conhecimentos ao projeto que desenvolvemos juntos. Focaremos em comentar sobre práticas que não só aprimoram seu portfólio, mas também reforçam a qualidade e a eficiência do seu trabalho como desenvolvedor.

Nossa jornada começa com a preparação para testes unitários. A importância de realizar testes de qualidade foi um tema recorrente em nossas discussões, e agora vamos aprofundar como preparar sua aplicação para essa fase crítica. Você aprenderá a implementar testes unitários no nosso projeto colaborativo, entendendo como esses testes contribuem para a agilidade, segurança e detecção

proativa de falhas e exceções no seu código. O teste unitário pode ser como o exemplo a seguir:

```
// Teste unitário para adicionar uma nova tarefa

describe('POST /tasks', () => {

     test('adiciona uma nova tarefa com sucesso', async () => {

              const newTask = { title: 'Aprender Jest', completed: false };

                              const response = await request(app).post('/tasks').send(newTask);

        expect(response.statusCode).toBe(201);

        expect(response.body.title).toBe(newTask.title);

         expect(response.body.completed).toBe(newTask.completed);

    });

});
```

Além disso, abordaremos o versionamento de código e suas melhores práticas, oferecendo dicas valiosas para construir um portfólio impactante que destaque suas habilidades e projetos. E, claro, discutiremos os próximos passos após a conclusão desta disciplina, orientando seu caminho no constante aprendizado e evolução profissional.

Focando nos testes unitários, fundamentais para testar as menores partes do seu programa, vamos além da simples implementação. É necessária uma preparação

meticulosa, pensando em cada detalhe, como a convenção de nomenclatura dos testes, que deve ser clara e refletir exatamente o que está sendo testado. Essa clareza é essencial para facilitar a manutenção e garantir a qualidade do software.

Para a preparação efetiva, considere todos os aspectos do teste: os cenários positivos e negativos, as exceções e as variações dentro de cada caso. Essa abordagem holística assegura que você não apenas cubra todos os possíveis caminhos de execução, mas também compreenda profundamente o funcionamento da sua aplicação. Lembre-se, a cobertura de teste abrangente é mais do que um número; é uma garantia de confiabilidade.

No mercado de hoje, onde o tempo é um recurso valioso, frequentemente observamos que a implementação de testes unitários pode consumir mais tempo do que o desenvolvimento em si. Isso ressalta a importância de métodos que otimizem essa prática, assegurando uma integração eficaz com o ciclo de desenvolvimento do software.

Adição de testes unitários na aplicação

Avançaremos na adição de testes unitários na nossa aplicação. Vou orientá-los na preparação do ambiente para implementar esses testes, enriquecendo o projeto e seu portfólio. Começaremos instalando o Jest e configurando o package JSON, essenciais para estruturar nossos testes.

A primeira etapa envolve o planejamento de testes, identificando as dependências necessárias, como o MongoDB para simular nosso servidor de banco de dados. É crucial consultar a documentação do Jest e de quaisquer outras ferramentas envolvidas para entender as configurações necessárias, evitando armadilhas comuns de implementação.

Em nossa aplicação, utilizaremos valores mockados para simular cenários de teste, requerendo um entendimento sobre mocks e sua importância em replicar o comportamento de componentes externos, como o banco de dados, assegurando a veracidade dos testes.

Para praticar na sua aplicação, comece no VSCode, ajustando o script de testes no PackageJSON. Introduzao MongoDB Memory Server para simular o banco de dados, seguido pela instalação do Jest, o qual é um framework de teste abrangente.

Com as dependências configuradas, crie uma estrutura de diretórios de testes no projeto, seguindo as convenções do Jest. A organização dos testes em diretórios como `database` e `repository` reflete nossa abordagem de testar desde a camada de acesso aos dados, subindo gradualmente pelas camadas da aplicação.

Na configuração, ajuste o PackageJSON para incluir os scripts de teste e ignorar diretórios específicos durante a execução dos testes, otimizando o processo de teste e focando nos componentes relevantes. Veja o teste a seguir:

```
// Teste unitário para atualizar uma tarefa

describe('PUT /tasks/:id', () => {

    test('atualiza uma tarefa existente', async () => {

            const task = { title: 'Aprender Node.js',
completed: true };

        const updatedTitle = 'Aprender Node.js avançado';

        const response = await request(app)

            .put(/tasks/${task.id})

            .send({ ...task, title: updatedTitle });

        expect(response.statusCode).toBe(200);

        expect(response.body.title).toBe(updatedTitle);
```

```
    });

});
```

Praticando com um exemplo de teste unitário como o acima, foque na camada de repositório, simbolizando a interação direta com o banco de dados. A criação dos testes segue uma lógica de preparação e limpeza do ambiente de teste, utilizando métodos do Jest para configurar e desmontar o cenário de teste antes e após cada teste.

Esse processo destaca a importância de testar ambos os cenários positivos e negativos, abrangendo a gama completa de comportamentos esperados da aplicação. Utilize assertions para validar se os resultados dos testes estão alinhados com os valores esperados, garantindo que a aplicação se comporte como previsto em diferentes condições.

Ao concluir os testes e executar o comando de teste, observe os seus resultados no terminal, identificando se todos os testes passaram ou se ajustes são necessários, refletindo um ciclo iterativo de testes e melhorias.

Encerrando esta etapa, é essencial ressaltar a importância de replicar esses testes para outras partes da sua aplicação, solidificando o entendimento e a prática dos testes unitários em diferentes contextos do projeto. Essa prática não só melhora a qualidade do software mas também aprofunda seu conhecimento e habilidades como desenvolvedor.

Versionamento de código e boas práticas

Já discutimos a importância do versionamento anteriormente, mas agora focaremos em como implementá-lo eficientemente, especialmente em contextos de trabalho em equipe, onde um versionamento organizado é crucial para colaboração e entrega eficaz.

Versionamento de código não se limita a atualizar números de versão após cada mudança no software; ele é uma parte integral do desenvolvimento, envolvendo a gestão de diferentes versões do código-fonte. Utilizando sistemas de controle de versão distribuído, como o Git, você pode rastrear alterações, colaborar com outros desenvolvedores e manter um histórico organizado de seu trabalho.

Mesmo trabalhando individualmente, é vital versionar seu código. Um incidente com seu dispositivo pode levar à perda de todo o trabalho não versionado. Utilizando plataformas como o GitHub, você pode armazenar seu código de forma segura e acessível, permitindo também a retrospectiva do seu progresso e evolução como desenvolvedor.

Exploraremos o Git como ferramenta para o versionamento. É fundamental que seu sistema tenha o Git instalado. Verificamos isso através do comando `git --version` no terminal. Se o Git não estiver instalado, deve-se baixar e configurar a versão apropriada para seu sistema. Após a instalação, verifique a configuração do Git, assegurando que seu nome e e-mail estejam corretos, pois eles serão associados aos seus commits.

Foque, também, nas boas práticas de commit. Uma mensagem de commit deve ser informativa e refletir claramente as alterações realizadas, facilitando o entendimento do histórico de mudanças e a localização de alterações específicas.

Algumas práticas recomendadas no desenvolvimento e manutenção de APIs: primeiramente, a utilização do JSON para troca de dados é preferencial, devido à sua facilidade de uso e formato padronizado; ao nomear endpoints, prefira substantivos a verbos, refletindo claramente os recursos que estão sendo manipulados.

A segurança é primordial, portanto, utilize SSL (Secure Socket Layers) para proteger sua API contra ataques. Os códigos de status HTTP devem ser usados adequadamente para indicar o sucesso ou falha das operações da API. Para APIs com grande volume de dados, implemente filtros, classificação e paginação, melhorando a experiência do usuário e a eficiência da aplicação.

Portfólio para uma pessoa desenvolvedora

Discutiremos como estruturar um portfólio eficaz para desenvolvedores. Um portfólio é essencialmente uma coleção de trabalhos e experiências que demonstram suas habilidades e competências em programação. Pode incluir qualquer projeto que você tenha desenvolvido, seja em aulas ou individualmente, como dashboards, aplicações web, entre outros.

Para criar um portfólio impactante, é importante escolher uma plataforma de hospedagem confiável; o GitHub é uma opção popular devido à sua versatilidade e integração com diversas ferramentas de desenvolvimento. No GitHub, você pode não apenas armazenar seu código, mas também aproveitar para criar um portfólio interativo, destacando seus projetos e habilidades.

Começar a documentar seu progresso e projetos desde cedo é crucial. Mesmo que você esteja iniciando agora, criar um histórico de suas atividades de desenvolvimento pode fornecer uma visão clara de sua evolução ao longo do tempo. Se você ainda não iniciou, agora é o momento perfeito, pois ter um portfólio é fundamental para evidenciar seu crescimento e aprendizado na área.

Um portfólio deve ser mais do que uma simples coleção de códigos; ele deve refletir suas habilidades específicas, linguagens de programação com as quais você trabalhou, e áreas de interesse. Incluir idiomas que você domina e hobbies pode adicionar um toque pessoal, mostrando um pouco mais sobre quem você é além de suas habilidades técnicas.

Quanto ao design do portfólio, ele deve ser atrativo e refletir seu estilo pessoal. O layout deve ser pensado de forma que represente sua identidade como desenvolvedor, tornando-se um reflexo de sua personalidade e abordagem profissional.

Além do GitHub, outras plataformas como Linktree e WordPress podem ser utilizadas para organizar e compartilhar seus projetos de forma estruturada e

acessível, oferecendo uma visão compreensiva do seu trabalho e facilitando o acesso por parte de recrutadores e colegas de profissão.

Em resumo, o portfólio é uma ferramenta vital para qualquer desenvolvedor, servindo como um diário de sua jornada profissional e um meio de demonstrar suas competências e realizações. Na próxima aula, exploraremos os passos subsequentes e como continuar se desenvolvendo na carreira de programação. Até lá!

Próximos passos

Agora que você adquiriu conhecimento substancial, é crucial pensar em como aplicar e expandir esses aprendizados.

Primeiramente, é recomendável aprofundar-se em temas como versionamento de APIs e versionamento semântico, essenciais para o lançamento de aplicações no mercado. Além disso, expanda sua experiência com testes, aplicando-os em diferentes partes do projeto desenvolvido, para identificar e corrigir possíveis falhas, reforçando sua capacidade de garantir a qualidade do software.

A organização é fundamental, então revise e estruture seus repositórios para refletir claramente os projetos e as experiências adquiridas. Isso não apenas ajuda na construção de um portfólio coerente e atraente, mas também serve como um registro de sua jornada de aprendizado e desenvolvimento.

Ao detalhar seus projetos, forneça descrições claras e informativas, que evidenciem os objetivos, as tecnologias utilizadas e os resultados alcançados. Essas descrições serão valiosas tanto para o seu autoconhecimento quanto para apresentar suas capacidades a potenciais empregadores ou colaboradores.

Manter seus projetos atualizados é outra prática recomendada. Isso não apenas reflete sua dedicação e capacidade de evoluir com as demandas tecnológicas, mas também garante que seu trabalho permaneça relevante e aplicável às necessidades atuais do mercado.

Se ainda não iniciou seu portfólio, agora é o momento. Utilize plataformas como GitHub para documentar e exibir seus projetos, habilidades e progresso na área de desenvolvimento. Isso fortalecerá sua presença no mercado e facilitará o acesso a novas oportunidades.

Por fim, encorajo você a continuar explorando e se desafiando em áreas complementares ao que aprendemos. Isso pode incluir aprofundar-se em bases de dados relacionais, expandir seu conhecimento em testes para além da camada de repositório, ou até mesmo iniciar projetos em linguagens e frameworks distintos, como Java com Spring Boot, comparando-os com suas experiências anteriores em Node.js.

Este é o encerramento de nossa disciplina. Espero que esta jornada tenha sido enriquecedora e que continue se desenvolvendo e se conectando na vasta e dinâmica área de desenvolvimento de software. Até nosso próximo encontro, desejo-lhe sucesso e evolução contínua em sua carreira. Vejo você em breve!

GitHub da Disciplina

Você pode acessar o repositório da disciplina no GitHub a partir do seguinte link: https://github.com/FaculdadeDescomplica/ProgramacaoII. Esse repositório tem como principal objetivo guardar os códigos das aulas práticas da disciplina para aprimorar suas habilidades em vários tópicos, incluindo a criação e consumo de APIs com controle de autenticação utilizando Node.js e utilizando boas práticas de programação e mercado.

Conteúdo Bônus

Para complementar os estudos sobre "Boas Práticas de Mercado" e aprofundar seus conhecimentos práticos em testes de software, sugiro o vídeo "Aprenda a testar

rotas de API com Jest no NodeJS" do canal CodarMe disponível no YouTube. Esse vídeo é uma excelente fonte para entender como aplicar testes unitários em rotas de API utilizando Jest no ambiente do Node.js.

Referências Bibliográficas

Bibliografia Básica:

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados.** 7.ed. Pearson: 2018.

MEDEIROS, L. F. de. **Banco de dados:** princípios e prática. Intersaberes: 2013.

VICCI, C. (Org.). **Banco de dados**. Pearson: 2014.

Bibliografia Complementar:

CARDOSO, L. da C. **Design de aplicativos**. Intersaberes: 2022.

JOÃO, B. do N. **Usabilidade e interface homem-máquina.** Pearson: 2017.

LEAL, G. C. L. **Linguagem, programação e banco de dados:** guia prático de aprendizagem. Intersaberes: 2015.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados:** implementação em SQL, PL/SQL e Oracle 11g. Pearson: 2013.

SETZER, V. W.; SILVA, F. S. C. **Bancos de dados**. Blucher: 2005.

Ir para exercício