Trilha de conhecimento: preparo para processo Seletivo do Mercado Livre

Olá!

Fizemos esse documento com muito carinho, compilando conteúdos importantes para serem revisados para você mandar muito bem na primeira etapa do Processo Seletivo do Mercado Livre, que é o teste técnico na plataforma Hackerrank, a ser disponibilizado no dia 19/11/2021.

O Mercado Livre (conhecido também como MELI) é o **maior marketplace da América Latina** e a **20ª melhor empresa do MUNDO para se trabalhar**. Possui um programa de aceleração, feito com a Digital House, com duração de 08 semanas para todas as pessoas desenvolvedoras júnior que ingressam no MELI. Do ponto de vista de carreira, o MELI se preocupa genuinamente em formar um bom time de tecnologia, uma vez que o desenvolvimento de toda a tecnologia da companhia acontece *in house*. O time de tecnologia dobrou de tamanho no último ano. Hoje são mais de 2.000 pessoas e o time continuará crescendo em um plano agressivo de expansão para os próximos anos.

A trilha de conhecimento está dividida em momentos síncronos, que vão acontecer entre os dias 09/11 e 18/11 e conteúdos assíncronos, para você consumir no melhor dia e horário para você! **Não deixe de levar possíveis dúvidas para os plantões**. O time de pessoas instrutoras ficará super feliz em poder colaborar com seu desenvolvimento e aprendizado!

Composição da Trilha de conhecimento de Preparo para o Processo Seletivo do Mercado Livre:

1. [Assíncrono] Desafios de Lógica de Programação
2. [Assíncrono] Referências de conteúdos sobre HTTP + Rest;
3. [Síncrono] Momento síncrono: retomando HTTP + Rest, no dia 12/11, no fechamento.
4. [Assíncrono] Referências de conteúdos sobre Banco de Dados Relacional;
5. [Assíncrono] Referências de conteúdos-base sobre POO (Programação orientada a Objetos);
6. [Síncrono] Master Class sobre POO, no dia 12/11, às 14 horas. A acontecer neste zoom;
7. [Assíncrono] Conteúdo sobre Inversion Of Controler (IoC);

Te desejamos uma excelente preparação para esse processo seletivo!
Conte conosco e vamo que vamo!

🚀 Desafios de Lógica de programação

Para você se preparar para questões de lógica de programação que possam surgir no processo seletivo, indicamos que você faça os desafios nível 1 da plataforma beecrowd. São 217 desafios e você não precisa fazer todos, mas se conseguir separar um tempo diário para fazer, com certeza se preparar para questões de lógica de processos seletivos.

Nesse vídeo, o Gabriel Oliva explica o funcionamento da plataforma beecrowd.

🚀 Conteúdos sobre HTTP + Rest

Média de tempo necessário: 2h00 a 4h00

Os links abaixo abrangem os conceitos básicos sobre o protocolo HTTP (conceitos e códigos de status) e arquitetura REST (métodos e suas aplicações) além de dar um overview sobre cookies, uma forma de armazenar informações entre o cliente e o servidor.

HTTP

- Conteúdo sobre HTTP | Bloco 26 - Introdução ao desenvolvimento Web com NodeJS - Express: HTTP com Node.js
- Documentação sobre HTTP
- Documentação sobre os StatusCodes do método HTTP

Cookies

- Documentação sobre cookies
- Como usar cookies no Express

REST e RESTful

- Conteúdo sobre REST | Bloco 27 - NodeJS: Camada de Serviço e Arquitetura Rest e Restful - Dia 27.4
- O que é REST?
- Métodos HTTP

🚀 Conteúdos de banco de dados relacional

Média de tempo necessário: 5h00

Abaixo indicamos conteúdos sobre banco de dados relacional, todos eles do Course.

Os links sugeridos, dão base para conhecimento em SQL, reforçamos como funciona uma Chave primária, e também como é construído e quais são os principais conceitos de um banco relacional.

O link abaixo, nos reforça, como são criadas as chaves primárias, e qual a principal função da mesma, em uma base relacional de dados, iremos aprender, o que são tabelas e colunas.

- 19.1 | "Constraints (restrições), chaves primárias e chaves estrangeiras" *(Tempo médio de leitura 1h)*

Um banco de dados relacional é um banco de dados que modela os dados de uma forma que eles sejam percebidos pelo usuário como tabelas, nesse link iremos aprender sobre como funciona, tabelas, e como fazemos para realizar buscas.

- 19.2 - "Encontrando dados em um banco de dados" - Aqui entendemos como funciona um banco relacional *(Tempo médio de leitura 1h)*

Normalizar um banco de dados, nos mostra como as regras são feitas para fazer a modelagem do mesmo, assim reduzimos redundância, e aumentamos a integridade dos dados, melhorando também a indexação de consultas, aumentando a performance.

- 21.1 | Normalização e Modelagem de Banco de Dados (Foco aqui 💡) *(Tempo médio de leitura 3h)*

🚀 Links didáticos sobre a base de POO

Média de tempo necessário: ~1h00

Esses links vão lhe ajudar a compreender os princípios básicos da orientação à objetos, entender os termos, fundamentos e propriedades que diferenciam-na de outros conceitos de programação.

- Programação Orientada a Objetos (POO), este vídeo traz os conceitos fundamentais sobre orientação a objetos. *(Tempo médio de leitura 10 min)*

- Os 4 pilares da Programação Orientada a Objetos, este artigo aponta dentro dos 4 pilares de POO uma comparação com a programação estruturada e o que se espera como características de elementos desenvolvidos dentro de paradigma orientado a objetos. *(Tempo médio de leitura 20 min)*

- JavaScript orientado a objetos para iniciantes, este artigo demonstra como o paradigma de orientação a objetos pode ser atendido em Javascript. *(Tempo médio de leitura 30 min)*

🚀Entendendo o princípio Inversion of Control (IoC)

Média de tempo necessário: 30 min

Imagine o seguinte cenário. Você trabalha no controle de qualidade de produtos. Especificamente, recebendo produtos, conferindo a padronização, embalando, endereçando e disponibilizando para a expedição entregar. No início, o processo era bastante simples, você tinha um par de regras bem definidas pela própria fábrica e só precisava segui-las. Porém, com o tempo, essas regras foram mudando e tornando o processo que outrora era simples, em mais e mais complexo. Agora existia a necessidade de cobrir novas situações, atender regulamentações, otimizar os ingredientes, etc. Com o tempo, você e o time de qualidade começaram a se confundir, errar os processos e além de ficarem exaustos por tantas mudanças nas regras.

Por causa de tantos erros, o time técnico resolveu criar um mecanismo para aferir a qualidade. Esse mecanismo funcionava da seguinte forma, você não precisava mais executar aquela infinidade de processos, bastava apenas usar o mecanismo de aferição de qualidade se ele retornasse um resultado positivo você continuava com o fluxo de trabalho normal. Dessa forma, você e seu time perceberam que as regras poderiam continuar mudando à vontade, pois era só você aferir com seu mecanismo de qualidade para decidir se o fluxo seguiria ou não. De certa forma, essa decisão saiu das suas mãos agora você possui um mecanismo de aferição de qualidade que lhe diz se o produto pode seguir ou não no fluxo. Desse dia em diante o trabalho ficou mais produtivo, as mudanças acontecem sem causar estresses e o próprio mecanismo de conferência passa a ser um ativo, ou seja, algo que pode virar um produto, que a sua fábrica oferece para terceiros.

Esse exemplo, acontece muito em JavaScript e talvez você não tenha percebido. Praticamente todo Array.prototype.map ou Array.prototype.filter é uma IoC. Vamos pensar pela ótica do exemplo anterior. O método do array garante a iteração (operar em cada item) e o resultado (criar outro array, selecionar itens). No entanto, a lógica dessa operação vem do código do usuário: contents.filter(content => content.pubDate.year >= 2020).

Tá complicado de entender ainda, vamos transformar esse cenário em código Javascript.

```
function (items){
```

```
  for (const item of items){
      if(item.weight < 100){ ShippingService.submit(item);}
  }
}
```

Com um código assim, convenhamos! Fica muito tranquilo. Mas e se começarmos a complicar as validações para submeter ao serviço de expedição dessa forma:

```
function (items){
 for (const item of items){
    if((item.weight > 80) && (item.weight < 100) && (!item.oxidated)){
        ShippingService.submit(item);
     }
 }
}
```

Bem, as coisas começaram a ficar mais complicadas agora. Você está responsável por um monte de validações. Mas que tal se enviarmos isso para uma função que ficará encarregada de tudo e dessa forma você ficar com a obrigação apenas de submeter o que precisa ser validado. Como algo do tipo:

```
isValid(item){
  return ((item.weight > 80) && (item.weight < 100) && (!item.oxidated))
}


function (items){
 for (const item of items){
    if(isValid(item)){
        ShippingService.submit(item);
     }
```

```
  }
}
```

Perceba que com a função isValid recebendo todo trabalho de validação a função responsável por enviar os produtos para expedição, precisa se preocupar somente com isso agora.

Mas… e se nós nem precisássemos implementar a lógica de validação. E se essa lógica viesse na forma de um parâmetro. Não ficaria melhor?

```
function (items, isValid){
  for (const item of items){
     if(isValid(item)){
         ShippingService.submit(item);
      }
  }
}
```

Dessa maneira, perceba que toda a responsabilidade de trazer a lógica de validação, é de quem invoca a função para submeter os produtos para expedição. Dessa forma estamos aplicando o princípio de IoC.

Aprofundando um pouco mais em IoC

IoC não é propriamente um padrão, é um princípio que engloba padrões com a finalidade de prover algum tipo de Callback ao invés de atuar diretamente sobre algo. Dessa forma um padrão que se encaixa dentro da categoria IoC é um padrão que inverte, ou redireciona o controle para um agente externo. A finalidade dos padrões que usam o princípio IoC é remover as dependências do seu código.

Podemos usar um exemplo do uso de um padrão para representar o princípio IoC. Esse padrão é conhecido como Dependecy Injection (DI) que é o mais específico do princípio IoC. Vamos considerar duas classes, Carro e Motor e no primeiro exemplo temos o exemplo típico de dependência entre um e outro.

```
class Carro {
  constructor(modelo) {
     this._modelo = modelo;
```

```
    this.motor = new Motor();
  }
}

class Motor {
  constructor() {
    this._potenciaAtual = 0;
    this._potenciaMaxima = 100;
  }

  set potencia(valor) {
    if (valor >= 0 && valor < this._maxPower) {
       this._potenciaAtual = valor;
    } else {
          throw new Error(`Valor de pontência incorreto: ${valor}. Deve estar
entre 0 e ${this._maxPower}` );
    }
  }
}

const carro = new Carro('Onix');
```

Esse código demonstra uma forte dependência entre Carro e Motor, dessa forma todas as vezes que um carro for criado, existirá a dependência de criação de um motor dentro do construtor da classe Carro. Como pode ser visto no código, a propriedade this.motor de Carro deve ser uma instância de Motor.

Dessa forma, vamos considerar uma dentre as inúmeras formas de resolver esse problema através do uso da Dependency Injection (DI), que é um padrão que utiliza o princípio IoC.

```
class Car {
  constructor(modelo, motor) {
    this._modelo = modelo;
    this.motor = motor;
  }
}
const carro = new Carro('Onix',  new Motor());
```

Em nosso segundo exemplo, perceba que foi criada uma abstração para diminuir a dependência entre Carro e Motor. Agora não estamos mais instanciando um objeto Motor dentro de Carro. Motor, se tornou um parâmetro do construtor da classe Carro. Isso significa o seguinte, quando formos criar um carro, todas as dependências são definidas no cliente.

Entretanto é válido ressaltar que esse exemplo não esgota nem de longe todas as possibilidades do princípio IoC. Existem outras oportunidades de uso do DI e até Design Patterns que fazem uso do mesmo princípio.

Os padrões mais conhecidos que fazem uso desse princípio são:

- Service Locator
- Factory
- Abstract Factory
- Template Method
- Strategy

Se você fizer uma pesquisa sobre cada um desses designs de projeto, você vai conseguir enxergar o princípio IoC em cada um como algo comum.

🚀 Links didáticos sobre a base de IoC

Média de tempo necessário: ~2h10min

Esses links vão lhe ajudar a entender com mais profundidade o que é o princípio IoC e suas respectivas variações em padrões de projeto.

- Tutorial com aplicação prática de alguns dos principais design patterns que adotam o princípio IoC. *(Tempo médio de leitura 40 min)*

- Maravilhoso artigo escrito por Martin Fowler, em que ele define o tamanho e a forma ideal de desacoplamento de responsabilidades propostas pelo IoC. *(Tempo médio de leitura 20 min)*

- Artigo, também escrito por Martin Fowler, que mostra com nitidez a linha tênue que separa as responsabilidades que um código que adota o princípio IoC precisa manter. *(Tempo médio de leitura 20 min)*

- Excelente artigo com vídeos e parte escrita demonstrando o princípio IoC presente em soluções simples e que muitas vezes utilizamos em nosso dia a dia sem perceber. *(Tempo médio de leitura 50 min)*