Desafio Toro Desenvolvedor Full-Stack e Backend

Bem-vindo ao desafio de programação da Toro Investimentos.

Histórias de Usuário

Considere as seguintes User Stories:

`TORO-001` - Eu, como investidor, gostaria de acessar a plataforma Toro usando minhas credenciais de usuário e senha, para que eu possa aprender mais, investir mais e acompanhar meus investimentos.

`TORO-002` - Eu, como investidor, gostaria de visualizar meu saldo, meus investimentos e meu patrimônio total na Toro.

- Restrições:
  - Patrimônio do usuário deve conter as seguintes informações
    - Saldo atualmente em conta corrente
    - Lista de ativos (ações) pertencentes ao usuário, com quantidade de cada ativo e valor individual atual de cada um. (Ex: 10 ações PETR4, valor individual R$25,00)

        - Patrimônio sumarizado (Saldo + Valor totalizado dos ativos)

`TORO-003` - Eu, como investidor, gostaria de poder depositar um valor na minha conta Toro, através de PiX ou TED bancária, para que eu possa realizar investimentos.

- Restrições:
    - A Toro já participa do Sistema Brasileiro de Pagamentos (SPB) do Banco Central, e está integrado a ele. Isto significa que a Toro tem um número de banco (352), cada cliente tem um número único de conta na Toro, e que toda transferência entre bancos passa pelo SBP do Banco Central, e quando a transferência é identificada como tendo o destino o banco Toro (352), uma requisição HTTP é enviada pelo Banco Central notificando tal evento. O formato desta notificação segue o padrão REST + JSON a seguir (hipotético para efeito de simplificação do desafio):

```
POST <apiBaseUrl>/spb/events

{
"event": "TRANSFER",
"target": {
    "bank": "352", // Banco Toro
    "branch": "0001", // Única agenda, sempre 0001
    "account": "300123", // Conta do usuário na Toro (unica por usuário)
},
"origin": {
    "bank": "033", // Banco de origem
    "branch": "03312", // Agencia de origem
    "cpf": "45358996060" // CPF do remetente
},
"amount": 1000, // R$ 1000,00 reais
}
```

    - Outra restrição é que a origem da transferência deve sempre ser do mesmo CPF do usuário na Toro.

`TORO-004` - Eu, como investidor, gostaria de ter acesso a uma lista de 5 ações mais negociadas nos últimos 7 dias, com seus respectivos preços, para que eu possa escolher uma delas e comprar a quantidade que eu escolher, respeitando o limite de saldo disponível na minha conta Toro, para que assim eu consiga possa montar minha carteira de investimentos.

- Restrições:

    - Para efeito de simplificação do desafio, as 5 ações mais negociadas nos últimos 7 dias e seus respectivos preços não precisa ser "real", pode ser definida utilizando algum recurso pre-definido no backend (uma coleção predefinida no banco de dados ou arquivo JSON).

Etapa:

O Desafio Técnico da Toro Investimentos deve ser feito no tempo que você precisar.

Você está vivendo um dia a dia real nosso, as histórias acima são inspiradas em histórias reais elaboradas por nossos Product Managers (PM ou PO). Você, no papel de Time de Desenvolvimento, tem a liberdade para propor, discutir e implementar da melhor forma possível, buscando entregar o maior valor ao usuário no menor tempo, sem comprometer os requisitos, inclusive de qualidade.

O desafio consiste em você escolher uma das histórias acima e implementá-la. Prepare seu ambiente, crie um novo projeto, e implemente uma das USs. Este é o momento para considerar:

- Qual padrão arquitetural será adotado MVC? Clean Architecture? APIs Restful?
- Como irei automatizar os testes? Em quais níveis irei implementar testes? Unitário? Integração? E2E?
- Adotarei algum framework? Quais?

Então é isso. Escolha uma as 4 Histórias de usuário acima e Mãos à obra! Te esperamos na segunda etapa!

Iremos também fazer algumas perguntas sobre algumas das decisões tomadas por você em cima do seu projeto. Enfim, iremos conhecer melhor seu skill técnico.

É isso! O desafio está dado o que foi colocado até aqui já é suficiente para realizá-lo.

Daqui pra baixo é apenas material complementar de suporte ao seu desafio.

Abaixo, para efeito de direcionar o desafio, vamos te ajudar e propor um caminho para sua implementação. Mais uma vez, aqui é apenas um guia/sugestão, você tem total autonomia para fazer diferente, desde que atenda aos requisitos impostos pelas USs.

Se analisarmos o conjunto das USs tragas pelo nosso PM/PO acima, podemos já antecipar algumas Entidades necessárias na solução final:

- Uma coleção de Usuários, onde cada usuário tem:
    - um código da conta do usuário no Banco Toro (Para efeito de simplificaçao vc pode optar por usar o codigo da conta como codigo do usuário - chave primária, ou ter códigos separados);
    - um saldo de conta corrente (como não existe o requisito de extrato de conta, etc, vamos simplificar o saldo como num único atributo numérico do usuário que pode ser 0 ou maior 0);
    - Nome e CPF (nao está diretamente descrito no requisito, mas usaremos para identificar usuário na hora da transferencia bancária)
- Uma coleção de Ativos de Usuários
    - Ativos adquiridos por um único usuário. Você precisa ter saber quais ativos e quantidade do mesmo ativo que o usuário possui. Não está no requisito saber o "valor de compra" do ativo, apenas o "valor atual", então pode ser ignorado esta informação;
- Uma coleção contendo os "5 ativos mais negociados", e o seu valor atual (estes valores serao fiquitícios, mas deve ser possível alterá-los para efeito de demostração da solução final);

Considerando as Entidades e Coleções de Entidades acima, você precisa implementar o backend e o frontend da solução. Vale lembrar que este desafio é o mesmo para vagas "Backend" ou "FullStack", então você pode optar por trazer pronto para a segunda etapa uma das partes ou as duas a depender da vaga em questão:

Sugestão para a US `TORO-002`:

Backend:

> API de saldo e patrimonio do cliente

```
GET <apiBaseUrl>/userPosition

{
    "checkingAccountAmount": 234.00, // Saldo em conta corrente
    "positions": [
        {
            "symbol": "PETR4",
```

```
            "amount": 2,
            "currentPrice": 28.44,
        },
        {
            "symbol": "SANB11",
            "amount": 3,
            "currentPrice": 40.77
        },
    ],
    "consolidated": 413.19 // (234.00 + (28.44 * 2) + (40.77 * 3)
}
```

Esta api deve mostrar a posição do usuário conforme descrito na US e na respectiva restrição. Os valores retornados por esta API deve ser diretamente impactados e refletir a realidades após as operações realizadas nas APIs anteriores (deposito e compra de ativos).

Frontend:

Para o frontend vc deve garantir que o usuário possa visualizar as informações de patrimônio conforme descritas na US e na proposta da API de backend.

Sugestão de implementação para `TORO-003`:

Backend:

Trazer pronto as seguintes APIs.

> POST /spb/events

```
POST <apiBaseUrl>/spb/events

{
   "event": "TRANSFER",
   "target": {
       "bank": "352", // Banco Toro
       "branch": "0001", // Única agenda, sempre 0001
       "account": "300123", // Conta do usuário na Toro (unica por usuário)
   },
   "origin": {
       "bank": "033", // Banco de origem
       "branch": "03312", // Agencia de origem
       "cpf": "45358996060" // CPF do remetente
   },
   "amount": 1000, // R$ 1000,00 reais
}
```

Esta API deverá está funcionando e poderá ser chamada para simular um evento de transferência / deposito na conta do usuário. Após o envio deste evento, o saldo do usuário cujo a conta está indicado no "target" deverá ser impactado (somado ao saldo atual).

De acordo com a "restrição" descrita para esta US, é necessário verificar se o cpf da origem é o mesmo do usuário a receber tal transferência.

Frontend:

- Para esta US, basta que exista na UI um lugar onde o usuário veja os dados da conta dele para transferência.

Para a `TORO-004`:

Backend:

> API de 5 ativos mais negocidos

```
GET <apiBaseUrl>/trends

[
    {
        "symbol": "PETR4",
        "currentPrice": 28.44
    },
    {
        "symbol": "MGLU3",
        "currentPrice": 25.91
    },
    {
        "symbol": "VVAR3",
        "currentPrice": 25.91
    },
    {
        "symbol": "SANB11",
        "currentPrice": 40.77
    },
    {
        "symbol": "TORO4",
        "currentPrice": 115.98
    }

]
```

> API ordem de compra

```
POST <apiBaseUrl>/order

{
    "symbol": "SANB11",
    "amount": 3
}
```

No exemplo acima o usuário deseja comprar 3 ações `SANB11`. Neste caso, a API deve chegar o valor de `SANB11` naquele momento (no exemplo, R$40.77), verificar se o usuário tem pelo menos R$122.31 disponível em conta corrente e, em caso afirmativo, realizar a compra (debitar o saldo e registrar as novas quantidades de ativos `SANB11` ao cliente). Caso não tenha saldo suficiente, ou o ativo informado seja invalido, a API deve retornar uma codido e uma mensagem de erro indicando "saldo insuficiente" ou "ativo invalido". Esta operação deve impactar o saldo e a lista de ativos do usuário.

Frontend:

Para o Frontend, você deve oferecer um fluxo onde o usuário veja lista dos ativos mais negociados, o valor de cada um, e possa clicar em um deles para comprar. Ao clicar, usuário deve informar a quantidade desejada para compra e submeter a ordem. Em caso de sucesso, usuário deve visualizar mensagem de sucesso e seu novo saldo e novas posições de ativos. Em caso de erro (saldo insuficiente) deverá ver msg apropriada.

Requisitos

- O projeto deve ser publicado em um repositório público no github.com, bitbucket.org ou gitlab.com
- README com instruções de como instalar as dependências do projeto, de como rodar a aplicação e os testes automatizados e de como fazer o deploy
- Deve ser desenvolvido em NodeJS, .Net Core ou Dart
- Front-End deve ser em Flutter ou Angular
- Necessidades diferentes dos requisitos podem (devem) ser negociados previamente.

Bônus

- Sistema executável rodando hospedado numa conta AWS.
- Backend deployado com Framework Serverless ou AWS SAM, ou rodando em docker-compose;

- Usar o CI/CD da plataforma onde hospedar o código (bitbucket pipelines, gitlab-ci, github actions)

Critérios de Avaliação

Os seguintes critérios serão usados para avaliar o seu código:

- Legibilidade;
- Escopo;
- Organização do código;
- Padrões de projeto;
- Existência e quantidade de bugs e gambiarras;
- Qualidade e cobertura dos testes;
- Documentação;
- Contexto e cadência dos commits.
- Princípios SOLID e Clean Code
- Aderência aos 12 fatores: https://12factor.net/

Dúvidas

Caso surjam dúvidas, entre em contato conosco pelo nosso email:
desafiotoro@toroinvestimentos.com.br