SEVEN SPRING DAYS

SUMÁRIO

0. INTRODUÇÃO

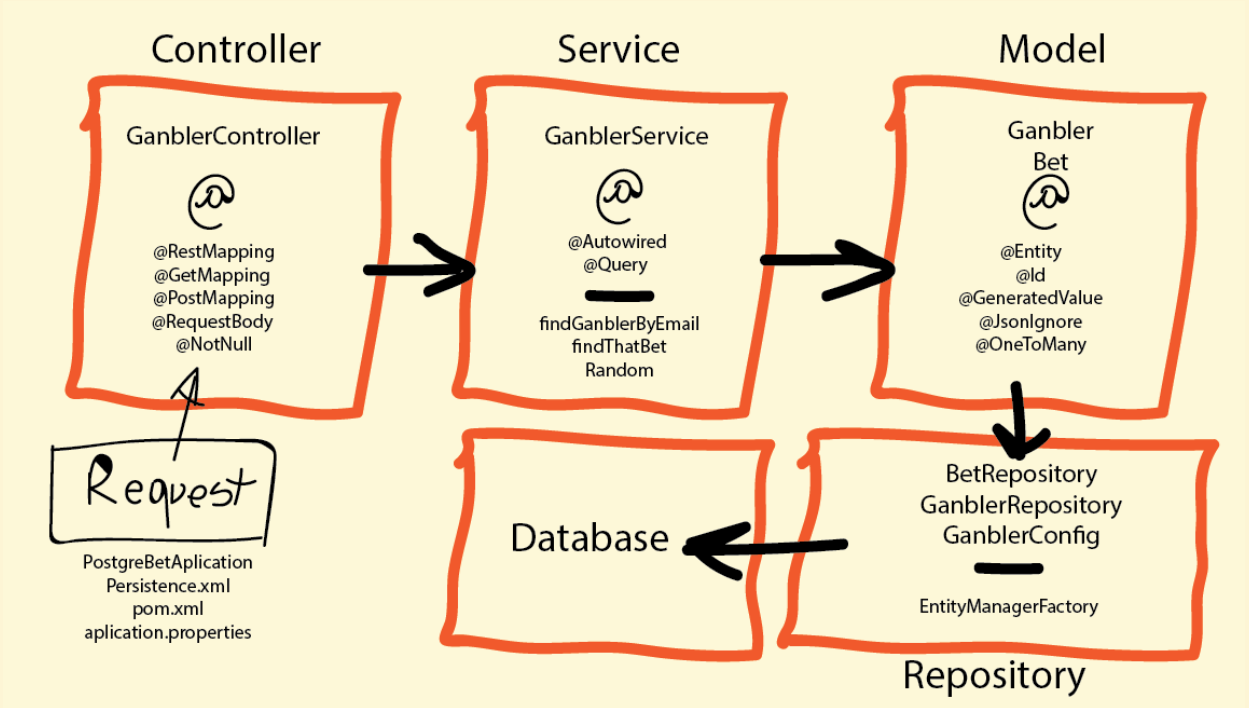
0.1 APRESENTAÇÃO

Olá! Meu nome é Danilo e eu estou aqui pra contar como foi realizador fazer esse projeto e compartilhar um pouco do que eu passei nesses últimos sete dias para conseguir concluir esse desafio. Eu já sabia que não seria fácil, mas eu não imaginava que eu iria me sentir tão bem em concluir isso tudo e ainda mais relatar cada detalhe que foi importante sem esquecer as recomendações do que foi proposto.

Enfim, sem muitas delongas vou priorizar a simplicidade e, como foi pedido no e-mail, vou fazer o possível para usar minhas próprias interpretações, mesmo que isso possa estar errado algumas vezes. Dessa maneira, eu vou me sentir mais livre para ser claro e direto ao ponto sem que algumas coisas, que eu ache importantes, sejam registradas aqui.

1. PROJETO ORANGE GANBLER

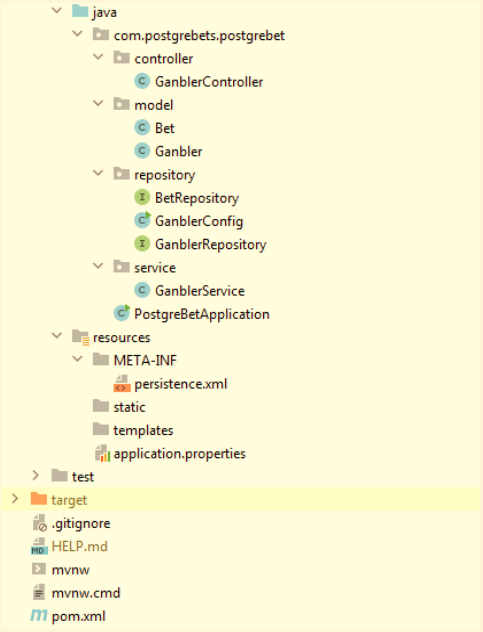
1.1 ARQUITETURA



O esquema acima representa os pontos mais importantes do projeto e os assuntos em que vou dar ênfase durante todo o post. Esses pontos foram os que eu achei mais importantes e trabalhosos durante esses dias e onde foquei maior esforço mental para entender.

O desafio se divide principalmente na primeira e segunda rotas e será sinalizado em negrito quando tratar de um requisito importante.

1.2. ROTAS



Agora vou dar um breve resumo sobre as rotas básicas do desafio proposto e falar sobre algumas anotações importantes.

Para iniciar, o Controller será responsável por criar definir a rota padrão para todas as outras através do comando abaixo.

```
@RestController
@RequestMapping("/api/v1")
```

Basicamente as rotas sempre irão iniciar com esse caminho para enfim serem divididas nas demais rotas do projeto.

A segunda e a terceira rota são exatamente as rotas propostas pelo desafio, mas antes eu vou exemplificar algumas anotações e linhas de código de algumas coisas importantes.

1.2.I ROTA FIND ALL

```
@GetMapping("/")
public List<Ganbler> getGanblers() {
    return ganblerService.findAll();
}
```

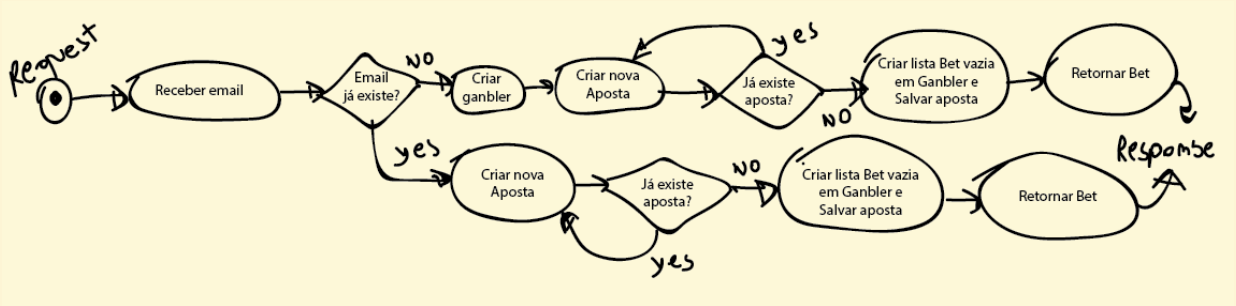
Este método utiliza-se da interface GanblerRepository através das anotações do @AutoWired que disponibilizam alguns serviços para os CRUD's básicos. Este @AutoWired dispensa a utilização do "new" nos objetos por fazer uma ligação entre eles automática, como o próprio nome já diz. Sendo assim, essa anotação facilita muito na ligação do construtor e dos getters and setters ao objeto principal sinalizado.

1.2.II. ROTA POST GANBLE

O PostMapping é a segunda rota. A partir daqui começo a primeira fase do desafio, onde o usuário faz uma requisição na rota "/ganble".

```
@PostMapping("/ganble/{email}")
public Bet newBet(@NotNull @RequestBody Ganbler ganbler) {
    return ganblerService.newBet(ganbler);
}
```

Esta rota chama uma ação POST para criar uma nova aposta "newBet" que por sua vez, antes de tudo**, verifica se existe alguma duplicidade de e-mail no banco de dados** utilizando o ".isPresent()"e logo depois executa a sequência lógica mostrada no esquema abaixo.



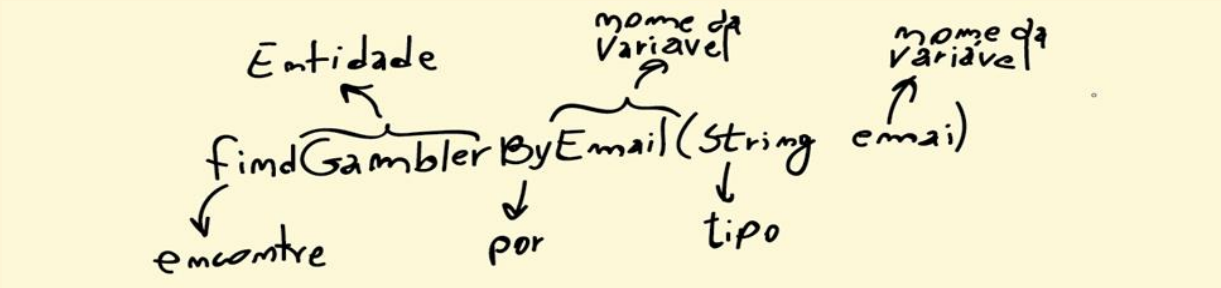
Em resumo existem dois caminhos lógicos a seguir e um deles só se diferencia do outro na necessidade de criar ou não um usuário. Mesmo assim é importante entender o caminho feito por essa rota.

O teste da variável é feito dentro de um "Do While". Neste exemplo um teste simples de um arranjo numérico onde apenas três números representam a aposta. O mais importante nesse contexto é a utilização de uma Custom Query, mas eu reservei um capítulo especial para isso.

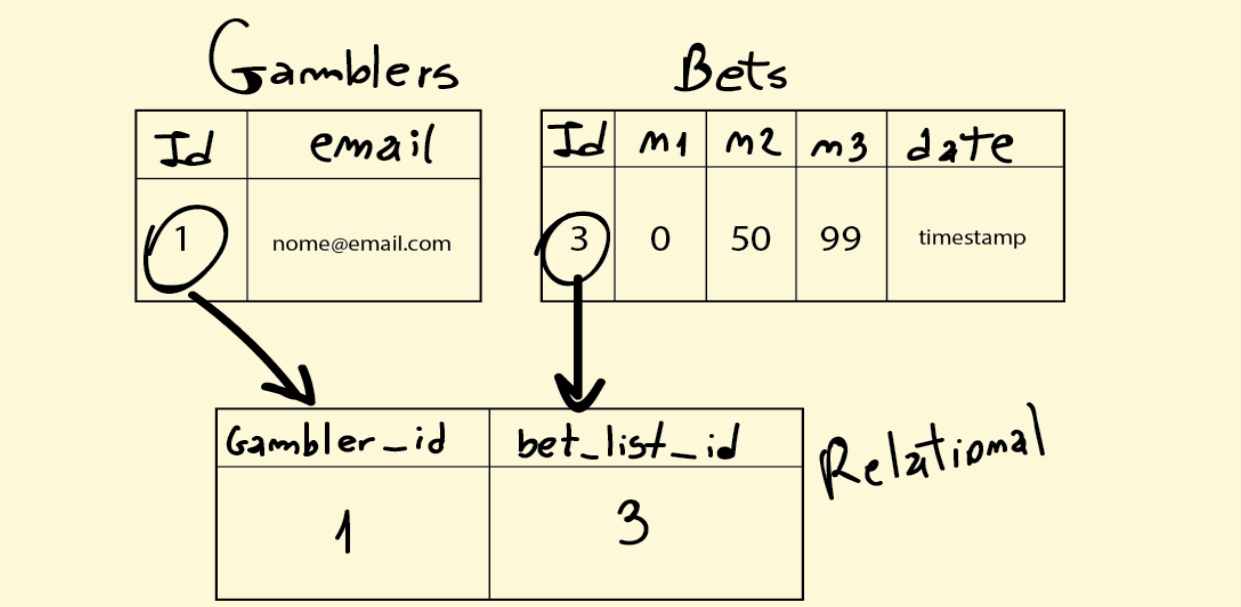
```
Random randomNumbers = new Random();
    int[] randomResult = new int[3];
    for (int i = 0; i <=2; i++){
        int num = randomNumbers.nextInt(99);
        randomResult[i] = num;
    }
    //TESTAR VALORES DA APOSTA
    newBet = new Bet(randomResult[0], randomResult[1], randomResult[2], new Date());
    testBet = betRepository.findThatBet(randomResult[0], randomResult[1],
randomResult[2]);

}while (testBet != null);
```

Uma abstração interessante para se comentar é a Named Query gerada pelo repositório que usa os nomes das variáveis e da Classe concatenados para uma ação simples usando o find. Esta anotação capta o objeto Ganbler e a partir do seu e-mail é gerada a aposta. Como vou mostrar abaixo, o Spring Data JPA faz tudo isso “por baixo dos panos”.



Além disso, para que tudo acontecesse foi necessário usar uma Chave Estrangeira e a anotação @OneToMany no Ganbler para referenciar uma ligação unidirecional entre os dois. De um “Pai Ganbler” para vários “Filhos Bet” gerando uma nova tabela relacional.



Observando o esquema podemos ver que os valores das duas chaves são referenciados para ligar uma tabela à outra. Desta forma, ao pesquisar pela Id de um usuário podemos ter as apostas e, além disso, cada aposta vai se ligar à “bet_list_id” como uma chave primária. Ou seja, cada usuário tem um ID que será um valor comum, repetitivo, para a tabela Relacional, porém sempre terá um valor único para cada aposta da tabela Bet, garaças “a bet_list_id”.

Por fim, “bet_list_id” será a chave primária da tabela relacional e fará referência a duas de forma unidimensional referenciando um Ganbler para muitos Bets.

1.2.II. ROTA GET BETS

A nossa segunda rota é a que **lista todas as apostas de um único usuário retornando um objeto com todas elas na ordem em que foram feitas**. O banco automaticamente já organiza as apostas pelo Id automaticamente a partir do comando abaixo.

```
@GeneratedValue(strategy = GenerationType.AUTO)
```

Os valores das chaves primárias são gerados em ordem e sem repetição de Ids para as tabelas. Mesmo assim, eu criei uma variável “date” com o “java.util” e marquei todas elas com um timestamp para verificar e garantir isso, principalmente por ser um recurso muito útil para a aplicação.

```
@GetMapping("/bets/{email}")
public List<Bet> findAllBets(@NotNull @RequestParam String email){
    return ganblerService.findAllBets(email);
}
```

Esta URL segue o caminho substituindo o “@” por “%40” para fazer a representação deste caractere que não pode ser usado diretamente em uma URL pelo padrão HTML. Seguindo para o ganblerService temos um simples método find novamente, retornamos uma List<Bet> através dos métodos padrões “.get().getBetList()”. Por fim, teremos a resposta da lista de apostas Bet para cada e-mail eviado e uma URL diferente para cada usuário.

```
public List<Bet> findAllBets(String email) {
    Optional<Ganbler> ganblerOptional = ganblerRepository
            .findGanblerByEmail(email);

    return ganblerOptional.get().getBetList();
}
```

1.3. CUSTOM QUERY

Uma Custom Query foi um comando que demorei a entender um pouco, principalmente pela sua relação com o uma EntityManager que necessita de um “persistence.xml” para referenciar duas classes. Entretanto, foi aí que eu realmente entendi o poder do Hibernate.

É incrível como ele consegue automatizar as coisas, embora pareça complicado no começo, ao menos é muito simples utiliza-lo. O suporte do JDBC para scripts SQL fez a execução direta de uma Query como String e isso facilita muito o serviço de quem já está acostumado.

O principal efeito nessa parte do projeto foi o de conseguir executar uma consulta nos três valores da aposta para **verificar se existem sequência de números iguais**.

```
//   select n1,n2,n3 from bets where n1=96 and n2=65 and n3=55;
@Query(value = "SELECT * FROM BETS WHERE n1 = ?1 AND n2 = ?2 AND n3 = ?3",
nativeQuery = true)
Bet findThatBet(int n1, int n2, int n3);
```

Acima pude entender bem a diferença entre as três formas e sua relação com uma Query Nativa para assim concluir o item bônus do desafio. Mais à frente vou falar também sobre um problema que, por causa do tempo esgotado, não consegui resolver.

1.4. VERIFICANDO REPETIÇÃO DE APOSTAS

Mesmo sem tempo para corrigir ainda fiquei refletindo sobre as melhores formas de fazer essa verificação para ficar bom de verdade. Na minha ideia atual a verificação só funciona para Arranjos de números e para que fique ainda melhor vamos pensar em um problema de Combinação, onde a ordem dos números não importa. Nesse ponto, eu precisaria de uma condição para assinalar que um número existe na tabela e para cada geração de número randômico usar um Break/Continue para gerenciar o fluxo lógico. Poderia ser usada também, alguma Custom Query para resolver esse problema diretamente com o PostgreSQL.

1.5. REST

Seguindo alguns exemplos do padrão REST eu separei as rotas em bets e ganbler. Já que neste projeto não seria utilizada uma autenticação com mais dados eu acabei usando o e-mail como um complemento para o Path URL onde cada requisição é dinamicamente separar pelo valor dos usuários serviço na própria URL.

2. TECNOLOGIAS DO MUNDO SPRING

Procurei usar o mínimo possível para que tudo pudesse ser explicado de forma simples, afinal eu devo usar os meus próprios conceitos como foi pedido para esse post. De fato, eu senti uma grande dificuldade em entender o papel de algumas ferramentas, mas isso é uma questão de tempo. Por hora vou comentar as principais dependências que usei neste projeto.

```
<artifactId>spring-boot-starter-data-jpa</artifactId>
```

Essa dependência fornece toda à automatização de Query's para facilitar a utilização das necessidades mais básicas de uma aplicação. Além disso, as anotações do Hibernate se comunicam pelo Spering Data para tornar o trabalho com SQL muito mais fácil.

```
<artifactId>spring-boot-starter-web</artifactId>
```

É a responsável pela mágica de responder requisições com uma série de comandos e elementos de response default. Até onde entendi, ela é a principal responsável pela organização de uma API REST em qualquer projeto Spring.

```
<artifactId>postgresql</artifactId>
```

Esse é o driver responsável por toda a comunicação entre o Java e a database SQL.

```
<artifactId>spring-boot-starter-test</artifactId>
```

Implementa a os módulos de teste da aplicação.

```
<artifactId>spring-boot-maven-plugin</artifactId>
```

Apesar de não ser uma dependência é um dos mais importantes. É ele o responsável pelo pom.xml e que implementa a relação entre as dependências e o projeto em si. E também, impõe muitas convenções que ajudam a manter o ambiente Java padronizado.

2.1. BENEFÍCIOS PARA A CONSTRUÇÃO DO CÓDIGO

Todas essas ferramentas tem um potencial muito maior, principalmente em um contexto enorme de uma empresa. Para uma empresa assim, com toda certeza, é a melhor escolha a se fazer.

Ter projetos que são automaticamente padronizados, com requests, exeptions, errors e outros componentes já indexados sem a necessidade de nenhum trabalho profissional para isso é sem dúvida a grande vantagem do Java, Spring e Hibernate.

Em relação ao JavaScript, existe uma menor liberdade que pode resumir em mais controle da aplicação mas para grandes contextos uma grande confusão de padrões. Muitas pessoas, empresas e organizações usando o mesmo código são resguardadas por essa arquitetura padronizada do Spring. Assim, boa parte dos problemas é resolvido no momento da iniciação do projeto no Spring Initializr.

2.2. EXPANSÃO DO PROJETO

Já é possível ter uma ideia básica da implementação de funcionalidades nesse código atual. De certo, precisaria de uma autenticação de usuário, uma criptografia para as informações mais importantes, tokens, gerador de templates, entre outros. Embora eu não conheça muitas ferramentas Spring consigo ter uma noção do que seria necessário para realizar isso pelas descrições das dependências e avaliando o que já utilizei em JavaScript.

- Thymeleaf para o redirecionamento, server-side rendering;
- Spring Security para autenticação;
- Spring Data R2DBC para relações reativas com o banco de dados;
- WebSocket para transações real time no banco de dados;
- Spring Validation para gerenciar validações;

2.3. PROBLEMAS DO PROJETO

Agora é um momento de autorreflexão.

Os maiores problemas dessa aplicação vêm da falta de domínio com Java e de conhecimento do Spring. Além disso, minha pouca experiência com databases relacionais me deu uma grande necessidade de estudo durante alguns dias, o que me tomou muito tempo.

Para ser sincero, os maiores problemas estão na simplicidade no modelo de usuários, na verificação das apostas repetidas (como já falei anteriormente), na utilização de ID's simples.

Em conclusão, eu acredito que o projeto tenha ficado de acordo com as exigências mínimas e que por causa de alguns problemas que eu tive durante a execução não consegui deixa-lo ainda melhor.

3. ETAPAS DO PROCESSO

Esse capítulo eu reservei para algumas coisas extras, mas que achei que poderiam ser importantes para contextualizar este Blog Post e responder os itens do desafio de forma mais agradável.

3.1 IMPLEMENTAÇÃO PARA WEB

Para implementar esse código eu utilizaria Angular ou Sapper/Svelte para lidar com as respostas das requisições e executar as ações para completar o fluxo lógico do projeto. Usar rotas dinâmicas, requisições por Fetch e variáveis reativas podem ser grandes atalhos para “linkar” as View’s em JavaScript a esta API REST na arquitetura MSC.

3.2. MAIORES DIFICULDADES

Minha ultima “conversa” em Java foi em 2016 onde fiquei quatro meses aprendendo do zero até meu primeiro programa envolvendo Visão Computacional e Arduino. Até então, eu entendia quase nada sobre todo esse contexto da programação: Webistes, API’s, IoT e etc.

Durante esses, quase, sete dias eu me esforcei para entender conceitos e planejar tudo. Eu também assisti a várias vídeo aulas diferentes, antes mesmo de começar a “codar”. Perguntei a vários amigos programadores e fui muito empolgado para encarar esse desafio.

Errei várias vezes, foram vários dias parados em problemas relacionados à linguagem que eu não sabia nem de perto como resolver e no fim tudo começava a fazer sentido com tantas tentativas e análises.

Depois de muitas derrotas eu finalmente consegui! Mais de 10h por dia sentado na cadeira pelo menos deram um valor positivo. Conseguir cumprir uma missão é muito satisfatório e recompensador. Eu também me sinto cada vez mais predestinado a fazer isso da minha vida. Faz muito tempo que eu não dormia tão pouco e, mesmo assim, me sentia bem motivado.

Eu tenho certeza de que dei o meu melhor pra alcançar esse objetivo e me sinto muito feliz por isso. Eu encerro aqui essa longa “jornada de primavera”.

Obrigado pelo grande empurrão ZUP!

https://github.com/hahayz77/SpringPostgreOrange.git