



Nome: Phelipe Almeida de Souza.

Função: Desenvolvedor Full Stack Java Jr.

Email: phelipe_1000@hotmail.com

Desafio

Criar uma API REST que sorteie números aleatórios para loteria. E como forma de facilitar a identificação do apostador, a aposta deverá estar associada a um email. O objetivo será criar dois endpoints, o primeiro receberá o email da pessoa e irá retornar um jogo como objeto de resposta, já o segundo é para listar todas as apostas do usuário solicitado todas em ordem de criação.

O QUE É UMA API REST?

Primeiramente gostaria de explicar o que é uma API REST, é uma interface que fornece dados em um formato padronizado baseado em requisições HTTP, ou seja,

que segue as regras especificadas pelas constraints. Ela será essencial para podermos realizar um sistema de loteria proposto no desafio.

FERRAMENTAS QUE SERÃO UTILIZADAS NO PROCESSO DE CRIAÇÃO

- **IDE** – Será utilizado o **Eclipse**
- **MySQL Workbench 8.0 CE** – Gerenciador de Banco de Dados
- **XAMPP** – Como software de servidor web
- **Postman** – Ferramenta que será utilizada para testar nossa API.
- Como linguagem de programação iremos utilizar **Java e Spring + Hibernate**.

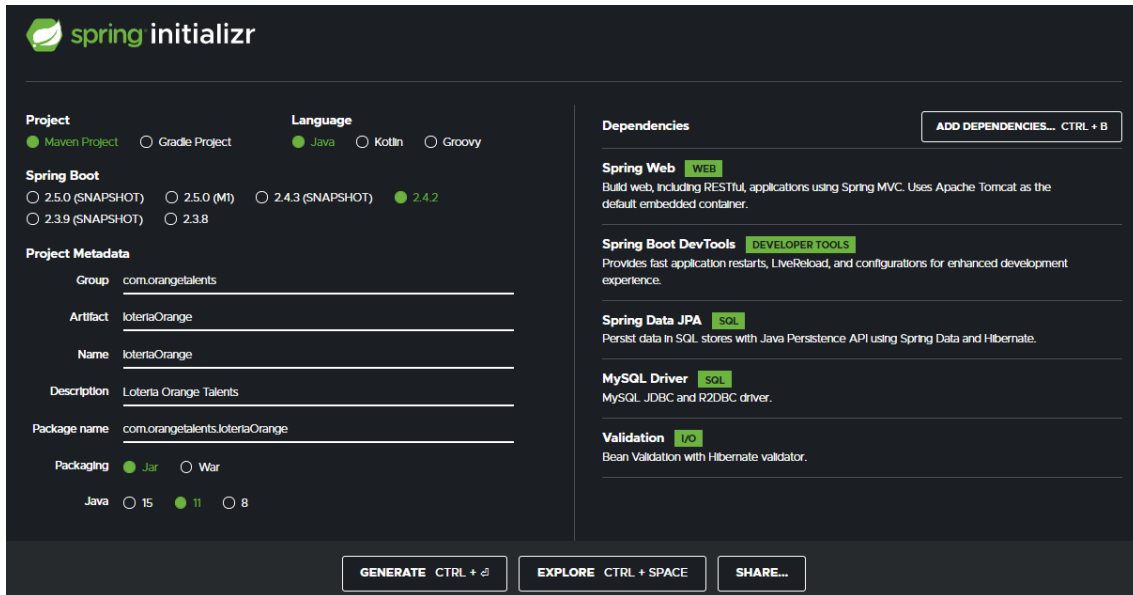
ABAIXO LISTAREI AS TECNOLOGIAS DO MUNDO SPRING QUE IREMOS UTILIZAR

- **Spring Web** – Para criar um web service.
- **Spring Boot DevTools** – Ajudar nas questões de restart e live reloading (Atualização do Projeto).
- **Spring Data JPA** – Para utilizar o Spring Data e outros recursos da JPA e Hibernate.
- **MySQL Driver** – Driver de conexão ao banco de dados MySQL.
- **Validation** – Incorporar validação aos dados da aplicação através de uma API fácil de usar e customizar.

AGORA VAMOS DAR INICIO AO NOSSO PROJETO!

Para dar inicio ao nosso projeto Spring iremos utilizar o Spring Initializr através do site <https://start.spring.io/> por ser uma das maneiras mais rápidas e praticas, com ele já é possível gerar o projeto com as dependências que iremos utilizar.

As dependências que iremos utilizar são: **Spring Web, Spring Boot DevTools, Spring Data JPA, MySQL Driver e Validation**:



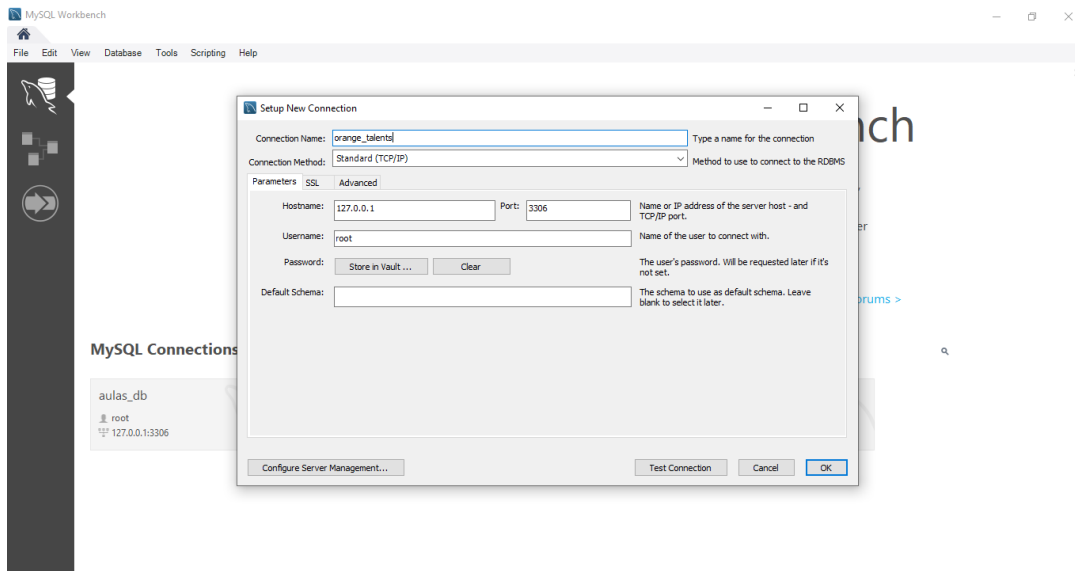
The Spring Initializr web interface is shown with the following configuration:

- Project:** ☒ Maven Project, ☐ Gradle Project
- Language:** ☒ Java, ☐ Kotlin, ☐ Groovy
- Spring Boot:** ☐ 2.5.0 (SNAPSHOT), ☐ 2.5.0 (M1), ☐ 2.4.3 (SNAPSHOT), ☒ 2.4.2, ☐ 2.3.9 (SNAPSHOT), ☐ 2.3.8
- Project Metadata:**
 - Group: com.orangetalents
 - Artifact: loteriaOrange
 - Name: loteriaOrange
 - Description: Loteria Orange Talents
 - Package name: com.orangetalents.loteriaOrange
- Packaging:** ☒ Jar, ☐ War
- Java:** ☐ 15, ☒ 11, ☐ 8
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - Spring Data JPA** (SQL): Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
 - MySQL Driver** (SQL): MySQL JDBC and R2DBC driver.
 - Validation** (JAO): Bean Validation with Hibernate validator.

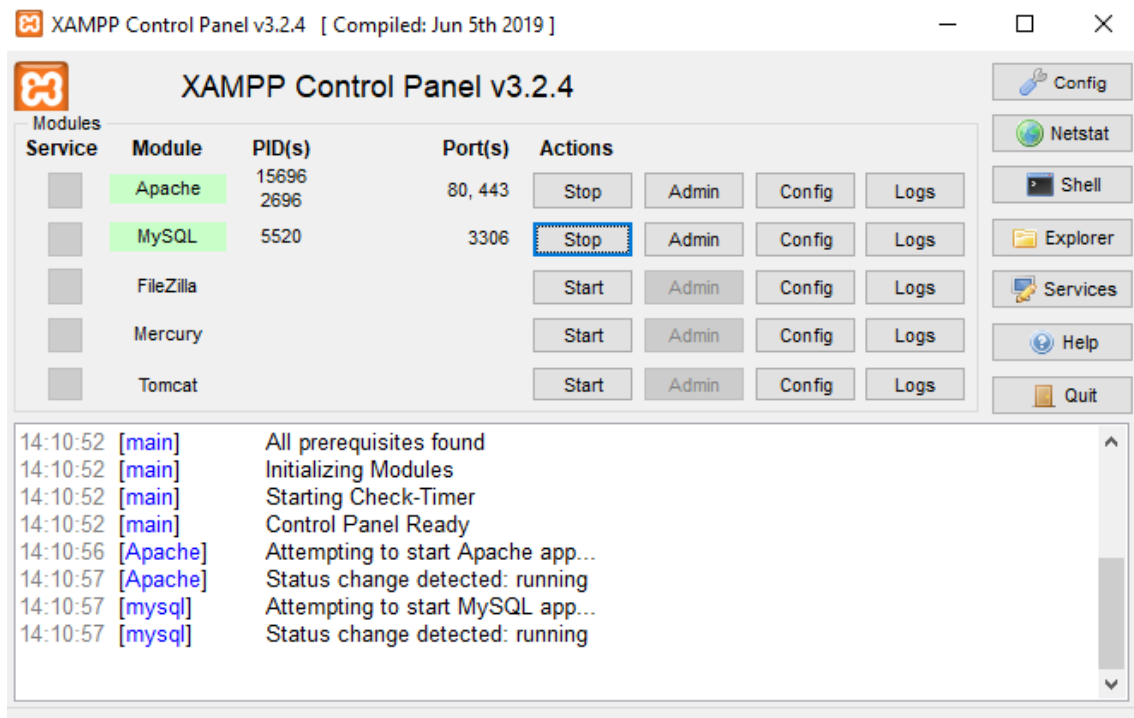
Buttons at the bottom: GENERATE (CTRL + G), EXPLORE (CTRL + SPACE), SHARE...

Ao clicar no botão GENERATE gera o projeto Spring em formato ZIP, após extrair o arquivo podemos abrir em qualquer IDE, no nosso caso será o Eclipse.

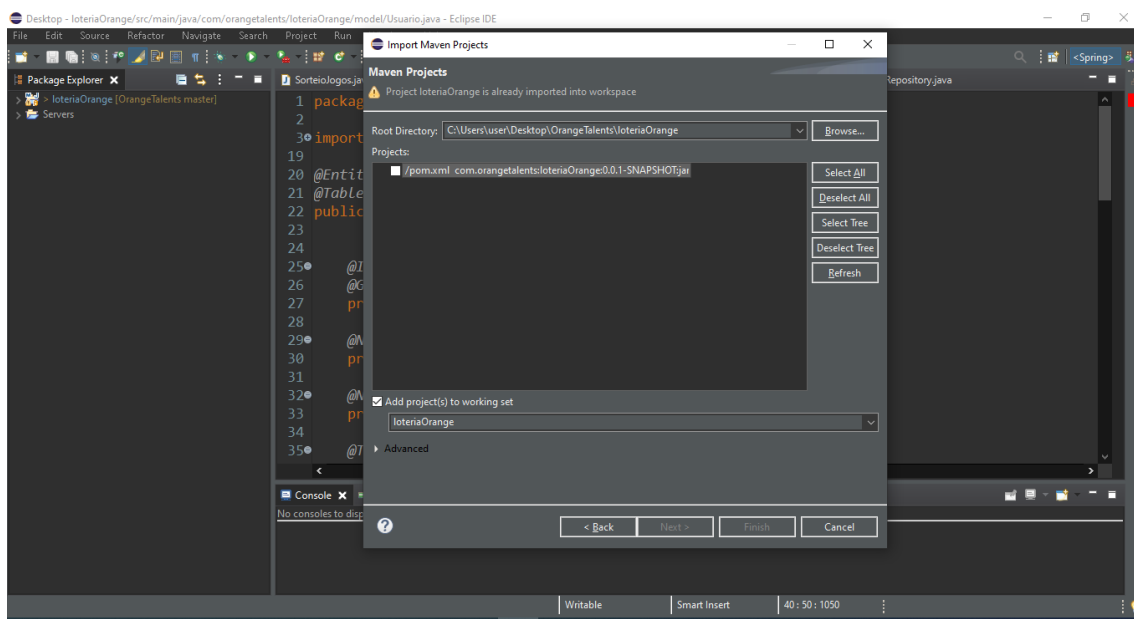
Vamos abrir o gerenciador de banco de dados MySQL Workbench onde criaremos uma nova conexão padrão:



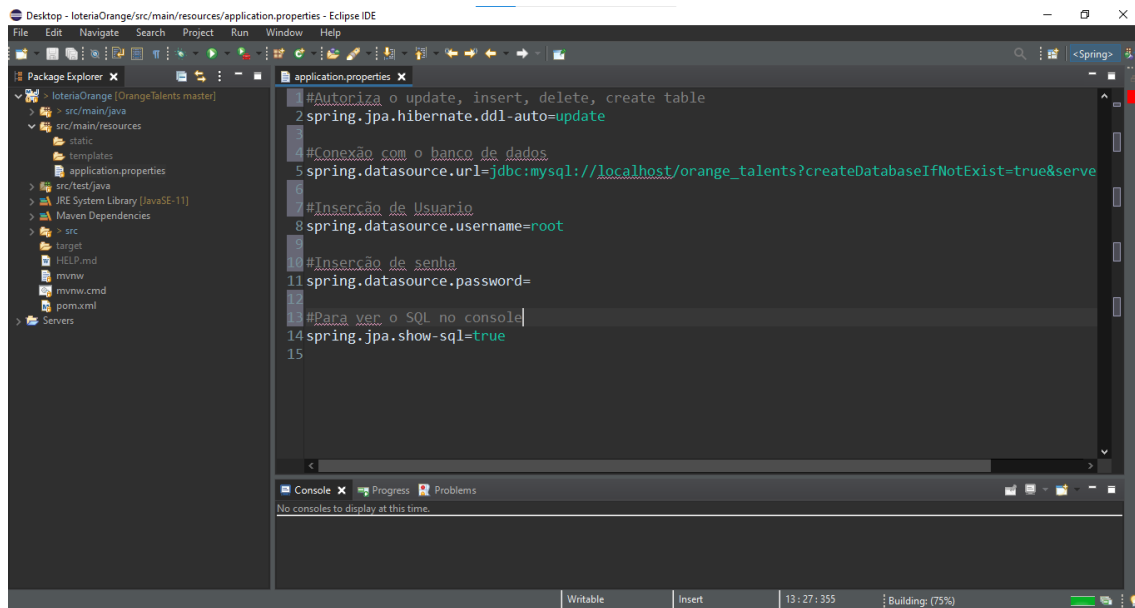
Agora Iremos abrir nosso software de servidor web, o XAMPP, e vamos ativar o Apache e o MySQL no botão Start:



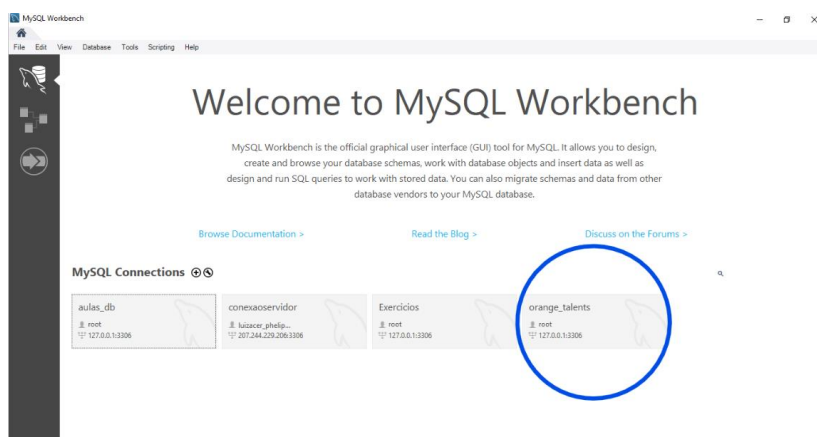
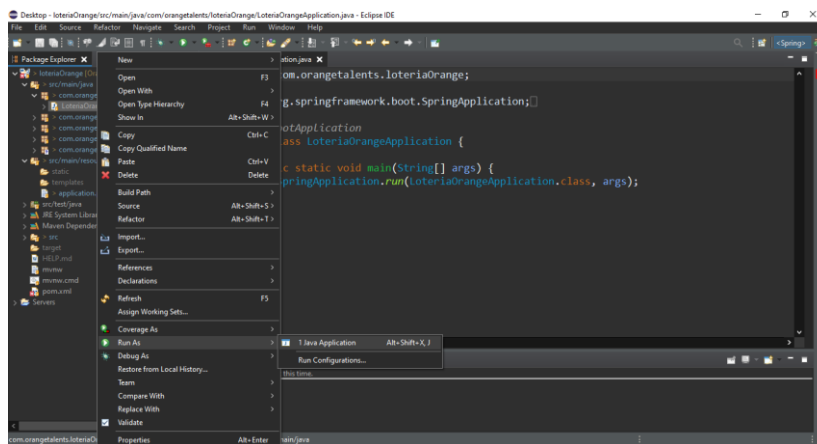
Feito isso iremos abrir nosso projeto em nossa IDE Eclipse, Vamos fazer juntos esse passo, primeiro vamos em: File > Import > Existing Maven Projects e selecionamos o arquivo que foi extraído. Bom lembrar que temos que marcar a opção de “pom.xml” que contem todas as nossas dependências e também a opção “Add project(s) to working set” para que nosso projeto seja visto na área de trabalho da IDE. Feito isso clicamos em “Finish” e aguardamos serem carregados todas as dependências.



Projeto iniciado iremos através do arquivo application.properties (src/main/resources) configurar a conexão com o banco de dados, inserção de usuário e senha, autorização de update, delete, insert, create table, etc.



Feito isso entraremos no diretório src/main/java > Pacote “**LoteriaOrange**” e rodar a aplicação LoteriaOrangeApplication.java clicando com o botão direito no mesmo Run As > Java Application. Com isso irá criar o banco de dados “**orange_talents**”.



Agora criaremos nossas classes e as suas respectivas camadas, iremos utilizar quatro camadas, sendo elas Model, Repository, Controller e Service.

Model: é onde fica os modelos da aplicação em que serão criadas as classes **Jogo** e **Usuario**.

Repository: será criada as classes(interface) **UsuarioRepository** e **JogoRepository** que fará as transações com o banco de dados.

Controller: essa camada é utilizada para validar as regras de negocio e o banco de dados, dentro delas criaremos a classe **UsuarioController**.

Service: é onde valida os dados, executa as regras de negocio, usa a Repository e a Model para ter acesso ao banco de dados e persistir os dados, nela será criada a classe **UsuarioService**.

Segue Imagens das classe já configuradas:

Jogo

```
Jogo.java X Usuario.java
1 package com.orangetalents.loteriaOrange.model;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 @Table(name = "tb_jogos")
7 public class Jogo {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    private long id;
12
13    @NotNull
14    private String numeroDoJogo;
15 }
```

Usuario

```
Jogo.java X Usuario.java X
1 package com.orangetalents.loteriaOrange.model;
2
3 import java.util.ArrayList;
4
5 @Entity
6 @Table(name = "tb_usuario")
7 public class Usuario {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    private long id;
12
13    @NotNull
14    private String nome;
15 }
```

JogoRepository

```
JogoRepository.java X UsuarioRepository.java
1 package com.orangetalents.loteriaOrange.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface JogoRepository extends JpaRepository<Jogo, Long> {
7
8 }
9 }
```

UsuarioRepository

```
JogoRepository.java X UsuarioRepository.java X
1 package com.orangetalents.loteriaOrange.repository;
2
3 import java.util.Optional;
4
5 @Repository
6 public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
7
8     public Optional<Usuario> findByEmail (String email);
9
10 }
11 }
```

UsuarioController

```
1 package com.orangetalents.loteriaOrange.controller;
2
3 import java.util.List;
4
5 @RestController
6 @RequestMapping("/usuario")
7 @CrossOrigin(origins = "*", allowedHeaders = "*")
8 public class UsuarioController {
9
10
11
12
13
14
15
16
17
18 @Autowired
19 private UsuarioService service;
```

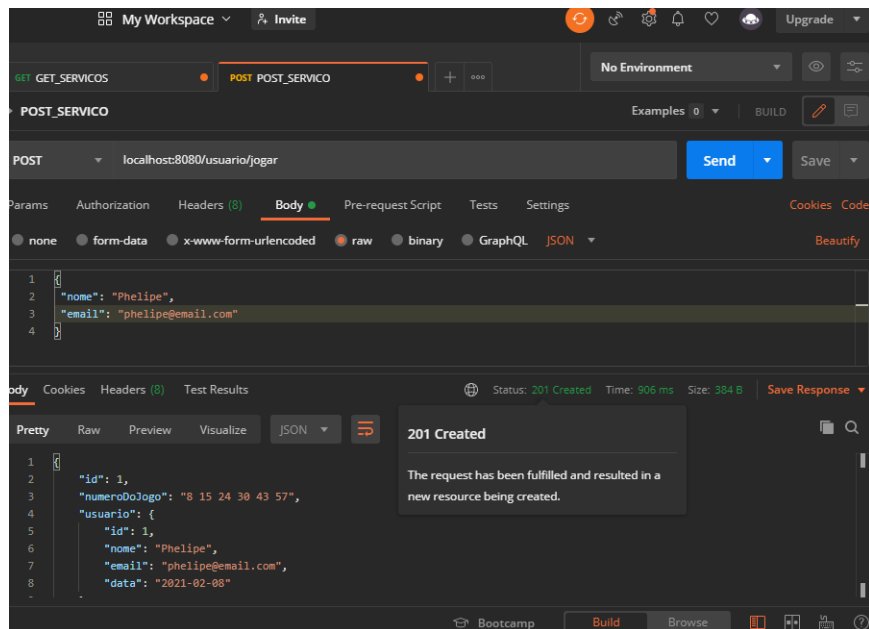
UsuarioService

```
1 package com.orangetalents.loteriaOrange.service;
2
3 import java.util.ArrayList;
4
5 @Service
6 public class UsuarioService {
7
8
9
10
11
12
13 @Autowired
14 private UsuarioRepository usuarioRepository;
15
16
17 @Autowired
18 private JogoRepository jogoRepository;
```

Executando a Aplicação:

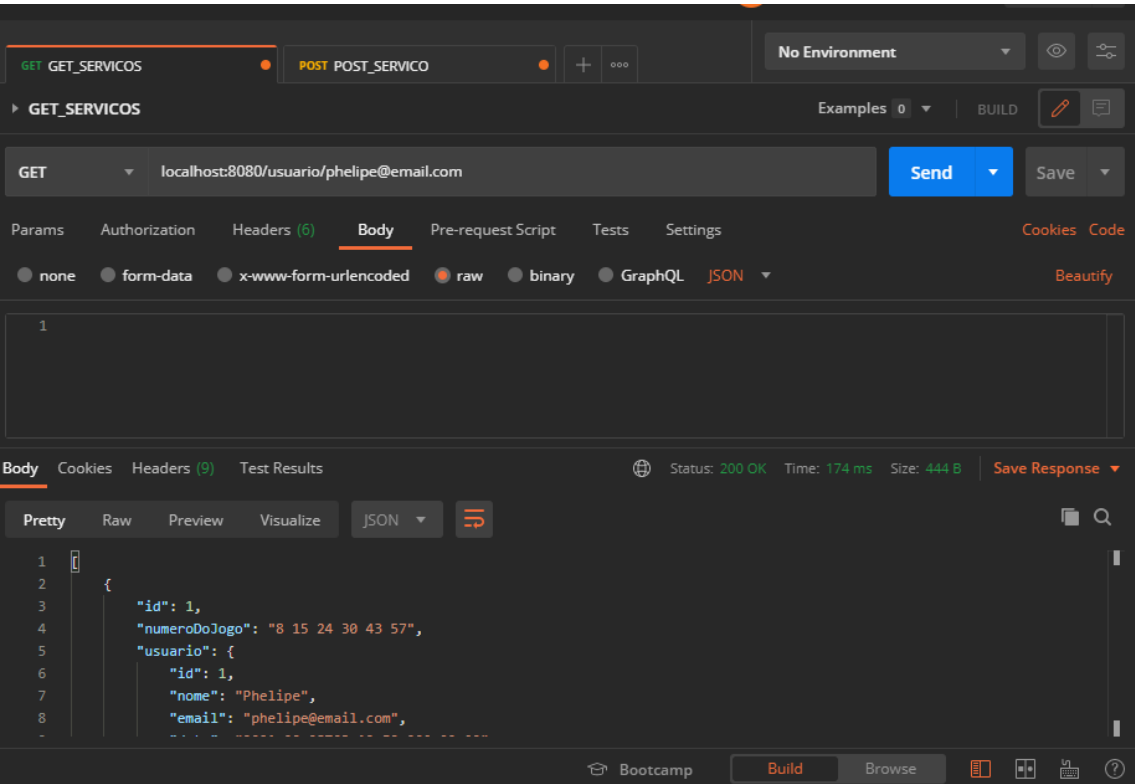
Ao Executar a classe LoteriaOrangeApplication, estamos “rodando” a nossa API REST no <http://localhost:8080>

Após rodar faremos o teste do nosso primeiro endpoint através do Postman como mostra a imagem abaixo:



Após colocar o nome e o email do usuário, já temos como objeto de retorno um numeroDoJogo com 6 dezenas.

Agora vamos dar um GET para buscar o jogo desse usuário, a busca será feita através do email (localhost:8080/usuario/email).



Feito a busca teremos o numero do jogo que foi feito e salvo no banco de dados:

tb_usuario

The screenshot shows a database query tool with the following SQL query:

```
1 use orange_talents;
2 select * from tb_usuario;
3 select * from tb_jogos;
```

The result grid shows the following data:

| id | data | email | nome |
|----|---------------------|-------------------|---------|
| 1 | 2021-02-08 03:19:59 | phelipe@email.com | Phelipe |

tb_jogos

The screenshot shows a database query tool with the following SQL query:

```
1 use orange_talents;
2 select * from tb_usuario;
3 select * from tb_jogos;
```

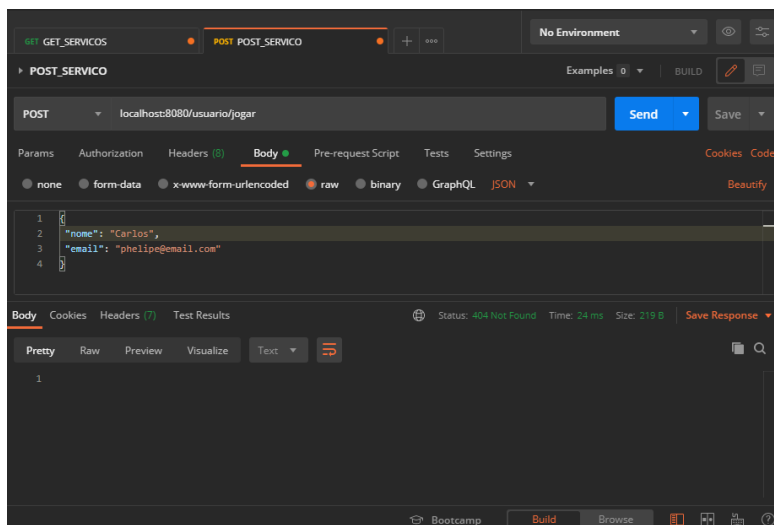
The result grid shows the following data:

| id | numero_do_jogo | usuario_id |
|----|------------------|------------|
| 1 | 8 15 24 30 43 57 | 1 |

E para proteger nossa aplicação de e-mails duplicados foi criado um método no nosso UsuarioService, segue abaixo um trecho do código:

```
public ResponseEntity<Jogo> gerarJogos(Usuario usuarioDados) {  
    Usuario jogador;  
    Optional<Usuario> usuario = usuarioRepository.findAllByEmail(usuarioDados.getEmail());  
  
    if (emailCadastrado(usuarioDados.getEmail()))  
        return ResponseEntity.notFound().build();  
}
```

Caso o email já esteja em nosso banco de dados é gerado um notFound como resposta, segue abaixo imagem de teste feito no Postman, nesse caso mudamos o nome e deixamos o mesmo email e temos o Status 404 Not Found como objeto de retorno:



Deixarei abaixo um link contendo um vídeo da execução da API no Postman e posteriormente consultando o banco de dados:

<https://drive.google.com/file/d/1bCTiSRntX4hE9bVJ8UpTDZLqreBk7wU9/view?usp=sharing>

Chegamos ao fim, desde já agradeço por estar participando do processo da Orange Talents, deixarei minhas redes sociais para qualquer dúvida e Github onde o código está hospedado.

Muito Obrigado!

<https://www.linkedin.com/in/phelipe-almeida-1009b968/>

<https://github.com/Phelipe-Souza>

Atenciosamente,
Phelipe Almeida de Souza.