UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALICE FERNANDES SILVA

GERADOR DE CÓDIGO PARA UMA API REST COM BASE NO FRAMEWORK SPRING BOOT

CAMPINA GRANDE - PB

2019

ALICE FERNANDES SILVA

GERADOR DE CÓDIGO PARA UMA API REST COM BASE NO FRAMEWORK SPRING BOOT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharela em Ciência da Computação.

Orientador: Professor Dr. Adalberto Cajueiro de Farias.

CAMPINA GRANDE - PB

2019

S586g Silva, Alice Fernandes.
Gerador de código para uma API REST com base no framework Spring Boot. / Alice Fernandes Silva. – 2019.

10 f.

Orientador: Prof. Dr. Adalberto Cajueiro de Farias.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Spring Boot. 2. Web service. 3. Programação repetitiva. 4. RESTful. 5. Geração de código automática. 6. Sistema CRUD. 7. Back-end em Sring Boot. I. Farias, Adalberto Cajueiro de. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

ALICE FERNANDES SILVA

GERADOR DE CÓDIGO PARA UMA API REST COM BASE NO FRAMEWORK SPRING BOOT

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharela em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Adalberto Cajueiro de Farias
Orientador – UASC/CEEI/UFCG

Professor Dr. Carlos Wilson Dantas Almeida
Examinadora – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC– UASC/CEEI/UFCG

Trabalho aprovado em: 25 de novembro 2019.

CAMPINA GRANDE - PB

Gerador de código para uma API REST com base no framework Spring Boot

Trabalho de Conclusão de Curso

Alice Fernandes Silva[1]
alice.silva@ccc.ufcg.edu.br
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil

Adalberto Cajueiro de Farias
adalberto@computacao.ufcg.edu.br
Unidade Acadêmica de Sistemas e Computação
Universidade Federal de Campina Grande Campina Grande, Paraíba, Brasil

RESUMO

Spring Boot é um framework que pode ser utilizado no desenvolvimento de um Web Service RESTful. Para isso, é necessário a codificação de métodos com requisições que permitam a conexão entre cliente e servidor. Em geral, uma aplicação de pequeno porte possui várias entidades e a programação de serviços RESTful se torna repetitiva. Este trabalho tem como objetivo o desenvolvimento de uma ferramenta que minimiza a programação repetitiva a partir da geração de código automática com a finalidade de servir como um sistema CRUD para o desenvolvimento de back-end em Spring Boot.

PALAVRAS-CHAVES: RESTful. programação repetitiva. automática.

Repositório com os artefatos da solução:
https://github.com/alicesilva/geradorcodigo-api-spring

1. INTRODUÇÃO

Um gerador de código é uma ferramenta que é capaz de gerar código a partir de um determinado modelo de desenvolvimento de software e isto sendo utilizado de maneira correta em um ambiente de produção de sistemas é possível obter alguns benefícios, como por exemplo, ganho de tempo na produtividade, construção de funcionalidades de maneira eficiente e segura, e diminuição dos custos de codificação, além disso, o uso do gerador acarretará em um único estilo de programação, aumentando assim a qualidade do software.

Como citado por [1], o uso de ferramentas geradoras de código oferece uma grande ajuda às técnicas de desenvolvimento ágil. Nos dias atuais em que se deseja cada vez mais otimizar os processos de desenvolvimento é necessário o uso de ferramentas que trazem esses benefícios.

Com a Internet sendo utilizada como o principal meio de comunicação, muitas empresas estão desenvolvendo suas aplicações nas plataformas web. Frameworks web estão sendo muito utilizados por serem práticos e ser fácil escrever código neles, além disso, muitas vezes é escrito o mesmo código para atender necessidades semelhantes [2].

Muitas das aplicações web utilizam de uma API REST para a troca de informações. Neste contexto, uma API REST é um conjunto de requisições e respostas HTTP, geralmente expressadas no formato XML ou JSON [3]. Diante disso, os desenvolvedores devem codificar métodos que são chamados de requisições para que se possa estabelecer uma comunicação entre o cliente e servidor.

Em uma aplicação onde se tem várias entidades com informações armazenadas em um banco de dados e que serão utilizadas pelo usuário, será necessário, por exemplo, a codificação de requisições que retornam as informações de cada entidade gravada no repositório de dados. Para o programador essa codificação se torna repetitiva pois a diferença do código entre um método e outro está apenas no nome da entidade, aumentando cada vez mais o uso do copiar (Ctrl+C) e colar (Ctrl+V) no momento do desenvolvimento.

O objetivo deste trabalho é desenvolver uma aplicação que gera código para a execução de uma API REST com base no framework Spring Boot - que é uma ferramenta

que visa facilitar o processo de configuração e publicação de aplicações [4], a partir de um arquivo de configuração com informações necessárias para o desenvolvimento de serviços RESTful.

A abordagem proposta em [5] implementa um gerador de artefatos para sistemas web a partir de um template com o objetivo de acelerar o processo de implementação no desenvolvimento de software e diminuir as atividades repetitivas de copiar, colar e alterar.

Dessa forma, é possível ver o quão é importante o gerador de código quando se tem este problema de repetição. Com a ferramenta de geração o programador irá apenas realizar pequenas modificações no código que foi gerado automaticamente, de acordo com as suas necessidades, sem precisar de copiar e colar código entre os arquivos que fazem parte da API REST de sua aplicação.

Existem outras soluções semelhantes a esta proposta, porém elas são focadas em criar o projeto completo no framework Spring Boot, além disso não utilizam das boas práticas de programação para a criação de uma API REST e não oferecem aos programadores uma boa e simples interface. Como exemplo tem-se o framework JHipster [10].

2. ARQUITETURA E PROJETO DE SOLUÇÃO

2.1 VISÃO GERAL DA SOLUÇÃO

Spring Boot é um projeto da Spring que veio para facilitar o processo de configuração e publicação de aplicações back-end, fazendo uso de padrões e recursos que reduzem drasticamente o tempo e complexidade do desenvolvimento [4].

Spring Data é um conjunto de projetos da Spring para a manipulação de dados de diversas formas, entre elas, em bancos de dados relacionais como o MySQL e o PostgreSQL e também em bancos de dados NoSQL como o MongoDB e o Redis. Um desses projetos é o Spring Data JPA que permite a implementação do mapeamento objeto-relacional de um modelo de dados. O framework permite também o desenvolvimento de métodos para o acesso aos dados com pouco ou nenhum código, o que facilita muito o desenvolvimento das aplicações e aumenta a produtividade dos programadores [6].

JPA (Java Persistence API) é uma API da linguagem Java que oferece uma interface para frameworks de persistência de dados. Ela define um meio de mapeamento objeto-relacional para objetos Java simples e comuns, denominados *beans* de entidades [7].

Portanto, é possível realizar o desenvolvimento de uma API REST com o Spring Boot combinada com Spring Data JPA para a disponibilização de um repositório de dados. Para isso, é preciso identificar quais recursos serão gerenciados.

Para que seja possível identificar as entidades que irão persistir no banco de dados, o gerador de código foi implementado na seguinte estrutura (Figura 1):

- Ler um arquivo de configuração de extensão .mydsl contendo as informações necessárias dos recursos da API REST;
- Validar as informações de cada entidade;
- Gerar arquivos na linguagem Java com o código suficiente para a execução da API REST.

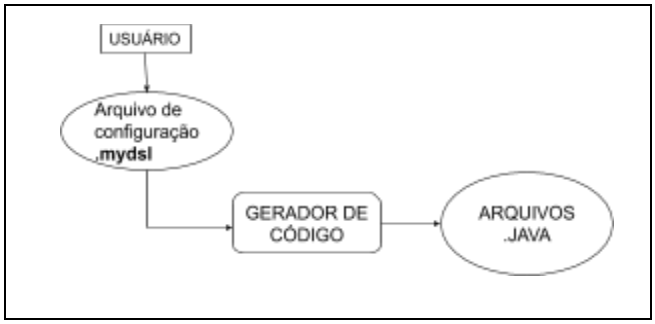

Figura 1. Diagrama do gerador de código.

2.2 TECNOLOGIAS UTILIZADAS

Por se tratar de uma aplicação para geração de código foi utilizado o framework Xtext no desenvolvimento.

Xtext é um framework para o desenvolvimento de linguagens de programação e linguagens específicas de domínio [8]. Nele é necessário definir o idioma a partir de uma gramática definida previamente. E como resultado, é obtido uma infraestrutura completa, incluindo analisador sintático, analisador semântico e geração de código. O Xtext é muito utilizado em projetos para compiladores de linguagens de programação.

Como observado na Figura 2 o código principal é organizado em três pacotes: org.xtex.tcc.mydsl, org.xtext.tcc.mydsl.validation e org.xtext.tcc.mydsl.generator.

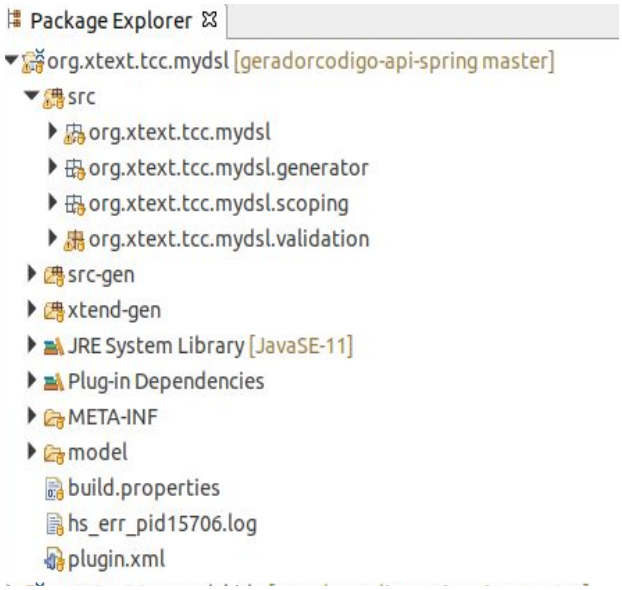

Figura 2. Estrutura de diretórios da aplicação.

O pacote org.xtex.tcc.mydsl contém o arquivo MyDsl.xtext que é onde está definida a gramática que define a estrutura da entrada recebido pelo gerador de código. Para esta aplicação foi criada uma gramática com regras e produções baseados no template do arquivo de configuração. Além disso, existe o arquivo GenerateMydsl.mwe2 que ao ser executado gera todos os artefatos necessários para o desenvolvimento da análise semântica e geração do código final.

Em org.xtext.tcc.mydsl.validation tem-se o arquivo MyDslValidator.xtend que é responsável por validar semanticamente as restrições específicas que foram definidas para a linguagem do domínio, juntamente com as classes auxiliares que podem ser criadas.

Já em org.xtext.tcc.mydsl.generator tem-se o arquivo MyDslGenerator.Xtend que é utilizado para a escrita da geração do código final. Para esta ferramenta, é gerado arquivos na linguagem Java com código para o desenvolvimento e produção de uma API REST baseada no framework Spring Boot.

2.3 PRINCIPAIS DECISÕES ARQUITETURAIS

Foi definido um template para o arquivo de configuração .mydsl, que é utilizado como entrada na ferramenta. Ele deve conter as seguintes informações:

- Nome da api;
- Uma lista de entidades;
- Para cada entidade deve ser informado:
    - Nome da entidade;
    - Lista de atributos.
- Para cada atributo deve ser fornecido:
    - Nome do atributo;
    - Tipo do atributo;
    - Nome da associação/relacionamento entre alguma entidade e este atributo;
    - Lista de operações que serão realizadas em cascata, no caso em que houver relacionamentos.

```
{
  "Nome da Api": "<inserir nome da api>",
  "Entidades": [
    {
      "Nome": "<inserir nome da entidade>",
      "Atributos": [
        { "nome-atributo": "<inserir nome do atributo>",
          "tipo-atributo": "<inserir tipo do atributo>",
          "associação/relacionamento": "<inserir nome do relacionamento
                                        entre este atributo e a entidade>",
          "operação em cascata": ["<inserir nome da operacao>",
                                  "<inserir nome da operacao>", ...]

        ]
      }
  ]
}
```

Figura 3. Template do arquivo de entrada do gerador de código.

O arquivo de configuração deve conter informações de pelo menos uma entidade. Cada entidade deve possuir no mínimo um atributo. Por obrigatoriedade cada atributo deve conter um nome e um tipo. O relacionamento é opcional, pois nem todo atributo representa uma associação com alguma entidade, e caso exista esse mapeamento objeto-relacional, é necessário informar quais operações serão realizadas em cascata.

Os nomes das operações em cascata e das associações são definidos como terminais na gramática. Portanto, estas informações devem ser especificadas no arquivo de configuração da mesma forma como mostra a Figura 4.

```
8  terminal NOME_ASSOCIACAO:
9      '"'("OneToOne" | "OneToMany" | "ManyToOne"
10         | "ManyToMany"
11     )? '"'
12 ;
13
14 terminal NOME_OPERACAO:
15     '"'("ALL"| "DETACH"| "MERGE"| "PERSIST"
16         | "REFRESH"| "REMOVE"
17     ) '"'
18 ;
```

Figura 4. Regras de produção da gramática para o nome da associação e da operação em cascata.

Os nomes das entidades e dos tipos dos atributos devem ser escritos com a inicial maiúscula seguindo o mesmo padrão de codificação da linguagem Java, mas os nomes dos atributos devem começar com a inicial minúscula. Além disso, são aceitos apenas as 26 letras do alfabeto e os nomes são simples, ou seja, não é aceito o caractere espaço.

Todas as restrições citadas anteriormente se referem a sintaxe do arquivo de configuração, então se a sua escrita não obedecer às essas regras o gerador irá detectar erros sintáticos no arquivo de entrada.

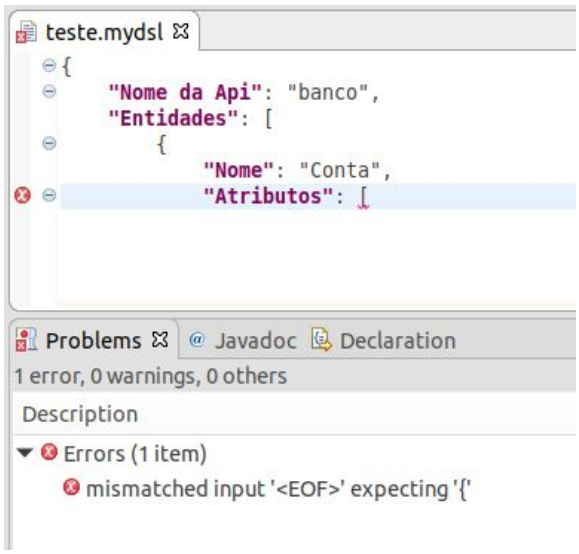

Figura 5. Exemplo de erro detectado quando o gerador não reconhece a estrutura do arquivo de entrada.

O tipo do atributo é dividido em três grupos: tipos primitivos, tipos coleções e tipo objeto. Todos os três grupos são definidos nas regras da gramática como terminais. O tipo primitivo pode possuir os seguintes valores: Boolean, Integer, Long, String, Float, Double, Time, Timestamp e Date. Já no grupo das coleções os valores reconhecidos pelo gerador são esses: List<T>, Set<T>, ArrayList<T> e HashSet<T>. Onde T representa o tipo do objeto que será armazenado na coleção.

```
terminal TIPO_PRIMITIVO:
    '"' ("Boolean" | "Integer" | "Long" | "String" | "Float" | "Double"|
     "Time" | "Timestamp" | "Date") '"'
;

terminal TIPO_COLECAO:
    '"' ("List<"(STRING_I) ">" | "Set<"(STRING_I) ">" | "ArrayList<"(STRING_I) ">"
        | "HashSet<"(STRING_I) ">") '"'
;

terminal STRING_I:
    "A" .."Z" ("a" .. "z" | "A".."Z")+
;
```

Figura 6. Regras da gramática utilizadas para definir o tipo dos atributos primitivos e das coleções.

Também é possível que o tipo do atributo seja do tipo objeto, neste caso, ele deve ser do tipo de alguma entidade que foi definida no arquivo de configuração.

Cada entidade deve ser unicamente identificada no banco de dados e para isso, por default, na geração de código cada uma delas possui um atributo id do tipo Long, que é marcado com a anotação @Id da JPA, identificando a chave primária da entidade.

Na etapa da validação foi realizada as seguintes análises:

- Se o tipo do atributo pertencer ao grupo do tipo objeto ele deve ser escrito igual ao nome de alguma entidade que foi definida no arquivo de configuração, identificando a classe do tipo do atributo;
- Se o tipo do atributo pertencer a alguma coleção, o tipo do objeto armazenado nele deve ser de algum tipo primitivo ou de alguma entidade definida no arquivo de configuração;
- Os nomes das entidades e dos atributos devem ser únicos.

Se o arquivo de entrada não obedecer à essas regras de validação, a ferramenta exibe uma mensagem de erro e não ocorre a geração do código.

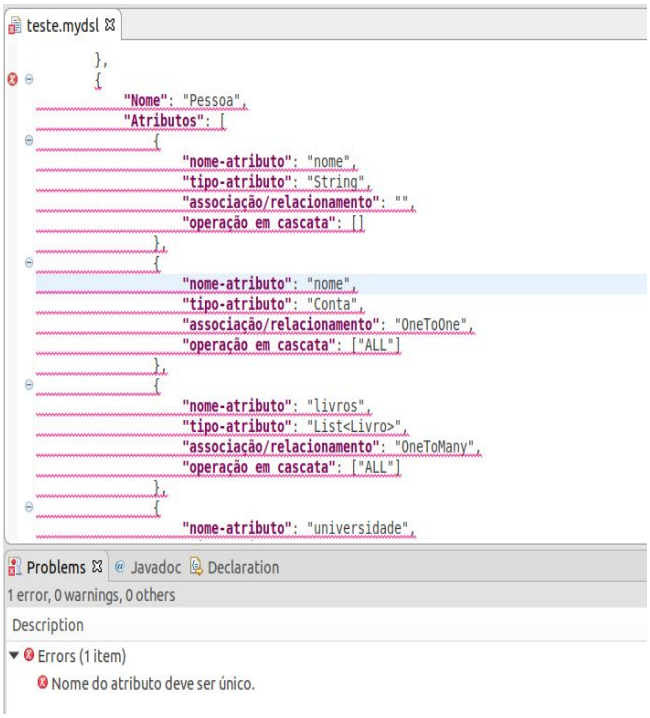

Figura 6. Exemplo de erro detectado na validação quando existem dois atributos com nomes iguais.

Na geração do código a ferramenta disponibiliza para cada entidade três arquivos:

- Uma classe Java que representa a entidade como um model que será persistido no banco de dados, com seus atributos e método getters e setters;
- Uma interface Java estendendo a classe JpaRepository que representa o repositório de dados, onde é possível ter acesso aos dados;
- E, por último uma classe Service onde está todas as regras de negócio e validação. Os serviços acessam os repositórios da entidades.

Os arquivos são nomeados da seguinte forma: <Nome-da-Entidade>.java, <Nome-da-Entidade>Repository.java e <Nome-da-Entidade>Service.java

As classes escritas na linguagem Java começam com o nome do pacote em que estão inseridas, *package <nome-do-pacote>;*. O mesmo ocorre para os arquivos gerados pela aplicação para que não ocorra erros de compilação ao serem inseridos em um projeto Spring Boot. As classes que representam os modelos das entidades iniciam com nome de pacote igual a *model*, os repositórios fazem parte do pacote *repository* e as classes services tem como nome de pacote *service*.

Além disso, é gerado um arquivo HomeController.java que representa o controlador da API REST, ele tem o nome do seu pacote como sendo *controller*.

Baseado no padrão de projeto Façade, que esconde as complexidades de um sistema em uma ou duas classe e provê uma interface simplificada ao cliente [9], esta classe representa a camada principal da API REST e é através dela que o cliente se conecta com o servidor por meio das requisições. As requisições geradas são expressas em formato JSON, devido ao fato de ser uma forma bem leve de representação e troca de informações mais rápida.

A Tabela 1 mostra os recursos da API que são gerados para cada entidade na classe do controller.

Método	URI	Utilização
POST	<nome-da-entidade>	Criar uma nova entidade e salvar no banco de dados.
GET	<nome-da-entidade>	Recuperar os dados de todas as entidades.
GET	<nome-da-entidade>/id	Recuperar dados de uma determinada entidade.
PUT	<nome-da-entidade>	Atualizar informações de uma entidade.
DELETE	<nome-da-entidade>	Excluir todas as entidades.
DELETE	<nome-da-entidade>/id	Excluir uma determinada entidade.

Tabela 1. Recursos gerados para cada entidade.

Diante disso, o controller gerado na ferramenta acessa as classes services de todas as entidades, facilitando a geração dos métodos quando existem relacionamentos entre elas para que seja possível associar uma entidade à outra. Isto é feito utilizando método com verbo PUT e capturando o ID das entidades na identificação do recurso.

A API trata apenas relacionamentos unidirecionais por ter uma implementação simples e as tecnologias utilizadas se responsabilizam de garantir a consistência dos relacionamentos e dos objetos.

Todas as requisições resultam em uma resposta, utilizando código HTTP para informar se ela processou com sucesso ou não. A Tabela 2 mostra o código de resposta para cada requisição, quando ela tem sucesso ou não.

Requsição	Resultado com sucesso	Resultado sem sucesso
POST	201 - Created	400 - Bad Request
GET	200 - OK	404 - Not Found ou 400 - Bad Request
DELETE	200 - OK	404 - Not Found ou 400 - Bad Request
PUT	200 - OK	404 - Not Found ou 400 - Bad Request

Tabela 2. Código de resposta para cada requisição quando ela processada com sucesso ou não.

3. SISTEMA EM USO

O gerador de código é disponibilizado para uso através de um plugin para a IDE Eclipse [11], em que a partir de um projeto Java e o arquivo de configuração .mydsl é possível obter os arquivos gerados de acordo com o que foi definido na estrutura da aplicação.

Com isso, foi realizado um estudo de caso para um pequeno sistema de reservas. A Figura 7 mostra o diagrama de classes desta aplicação.

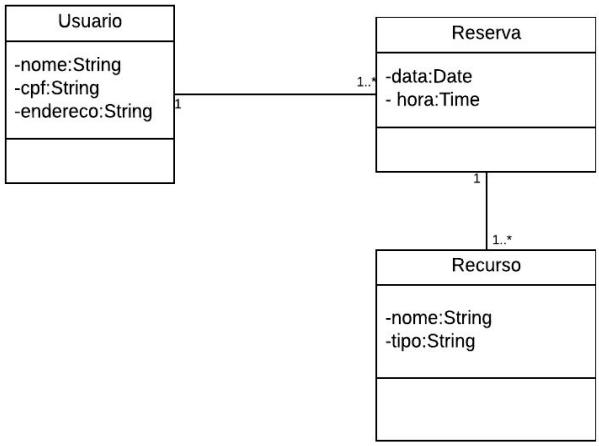

Figura 7. Diagrama de classes para um Sistema de Reservas.

A partir da modelagem UML o arquivo de configuração foi escrito e dado como entrada para o gerador de código. A partir disso, foi gerado automaticamente os arquivos com os serviços da API REST do sistema de reservas.

Logo após, iniciou-se o processo de criação de um projeto no framework Spring Boot configurado com o banco de dados PostgreSql.

Diante disso, foi anexado neste projeto os arquivos que foram gerados automaticamente. Foi detectado erros nos nomes do pacotes que iniciam as classes geradas pelo gerador. Além disso, a IDE também detectou erros na importação de classes de outro pacote. Isso acontece porque os nomes dos pacotes do projeto divergem dos nomes do pacotes de cada classe gerada pela aplicação.

Para resolver esses problemas foi necessário modificar os nomes do pacotes e das importações de cada classe que foi anexada no projeto para o sistema de reservas. O gerador de código não dá suporte para que o usuário insira o nome do pacote de cada entidade.

Após os ajustes, foi iniciado a execução da API REST, utilizando a ferramenta Postman [12] para testar todas as requisições que foram geradas na classe principal, HomeController.

Todas as requisições salvaram de forma correta as informações das entidades no banco de dados assim como retornaram os dados dos objetos gravados no repositório de dados. Não houve nenhuma diferença entre os resultados obtidos e os esperados.

4. EXPERIÊNCIA E LIÇÕES APRENDIDAS

4.1 PROCESSO DE DESENVOLVIMENTO

Inicialmente foi definido quais tecnologias iriam ser utilizadas no desenvolvimento da aplicação. A escolha foi baseada em trabalhos feitos anteriormente. Logo após foi feita uma análise da estrutura organizacional da ferramenta.

Foi realizado um estudo sobre o Spring Boot, Spring Data JPA e JPA, a fim de entender como ocorre o processo de desenvolvimento de uma API REST utilizando estas tecnologias. A partir disso foi definido de forma iterativa e incremental um template para o arquivo de configuração com o mínimo de informações possível e de fácil entendimento pelo o usuário da aplicação.

Diante disso, iniciou-se o processo de desenvolvimento nas seguintes etapas:

- Definição da gramática com regras de produção que fossem capazes de especificar o arquivo de configuração;

- Validação, na qual foram feitas análises, tais como, igualdade entre os nomes das entidades e verificação dos tipos dos atributos;
- Geração do código final que é constituído de arquivos escritos em linguagem Java.

4.2 DESAFIOS

Definir o template para o arquivo de configuração foi a parte mais desafiadora no desenvolvimento da aplicação. Foi necessário realizar um estudo sobre o desenvolvimento de uma API REST no framework Spring Boot para que fosse possível identificar quais as tecnologias utilizadas, as boas práticas de programação e quais os elementos principais para a execução.

O arquivo de configuração é baseado no formato JSON, então os objetos do arquivo são compostos por chaves e valores e para esta aplicação todos os valores são do tipo String. Portanto, foi preciso tratar cada valor retirando as aspas duplas para que fosse possível realizar a validação e a geração do código.

Escrever a gramática também não foi uma tarefa fácil. As regras gramaticais foram definidas de forma iterativa e incremental. Foi necessário vários testes e ajustes para que a ferramenta conseguisse ler e validar o arquivo de entrada de acordo com o template definido.

5. CONCLUSÕES E TRABALHOS FUTUROS

Usar o gerador de código para o desenvolvimento de back-end para prover serviços RESTful é eficaz pois otimiza o trabalho do desenvolvedor. Diante do código gerado apenas alguns ajuste precisam ser realizados de acordo com as necessidades do programador, uma vez que os arquivos gerados já são bem estruturados e com código suficiente para a execução de uma API REST.

O código gerado pela ferramenta é capaz de se adequar a configuração de qualquer sistema de gerenciamento de banco de dados relacional. Como exemplo foi utilizado o PostgreSQL para testar as requisições do caso de uso, porém outros sistemas como o MySQL também poderia ser utilizado para o teste das requisições com o código gerado pela ferramenta.

Para sistemas de pequeno porte que comporta os tipos básicos de objetos e relacionamentos unidirecionais o gerador de código desenvolvido neste trabalho tem forte impacto, com grande eficácia, pois quase nenhum código precisa ser escrito no desenvolvimento do back-end destas aplicações.

Trabalhos futuros podem estender o gerador de código para que seja possível abranger outros conceitos de desenvolvimento de API REST para aplicações web no framework Spring Boot, através da implementação de mais algumas funcionalidades:

- Tratar relacionamentos bidirecionais;
- Dar o usuário o poder de utilizar Herança entre as entidades;
- Adicionar outros tipos de atributos que possam ser persistidos em um banco de dados;
- Permitir que o usuário defina a chave primária de cada entidade;
- Permitir que o usuário insira o nome do pacote para cada entidade.

6. REFERÊNCIAS

[1] SPINELLI, Lucas Pompeo Pontes. Desenvolvimento de uma ferramenta para geração automática de código aberto em Java Server Faces. Assis, 2015.

[2] MERLIN, João Paulo. Desenvolvimento de uma ferramenta de scaffolding para criação de código fonte para front-end. 2018. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

[3]OLIVEIRA, Paulo Henrique Cardoso. Desenvolvimento de um gerador de API REST seguindo os principais padrões da arquitetura. 2015.

[4] WEBB, Phillip et al. Spring boot reference guide. Part IV. Spring Boot features, v. 24, 2013.

[5] ARANHA, Eduardo; BORBA, Paulo. Testes e geração de código de sistemas web. 16th SBES, p. 114-129, 2002.

[6] GIERKE, Oliver et al. Spring Data JPA-Reference Documentation. URL https://docs. spring.io/spring-data/jpa/docs/current/reference/html/.[utolj ára megtekintve: 2017. 04. 21.], 2012.

[7] BÖCK, Heiko. Java persistence api. In: The Definitive Guide to NetBeans™ Platform 7. Apress, 2012. p. 315-320.

[8] BEHRENS, Heiko et al. Xtext user guide. Dostupné z WWW: http://www. eclipse. org/Xtext/documentation/1_0_1/xtext. html, p. 7, 2008.

[9] MEDEIROS, Higor. 2012. Padrão de Projeto Facade em Java. (2012). https://www. devmedia.com.br/padrao-de-projeto-facade-em-java/26476

[10] RAIBLE, Matt. The JHipster mini-book. Lulu. com, 2016.

[11] Plugin do gerador de código. Disponível em: https://github.com/alicesilva/geradorcodigo-api-spring/tree/master/plugin

[12] Postman. Disponível em https://www.getpostman.com/