UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS I - CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM COMPUTAÇÃO

LUIS LANCELLOTE MARQUES DOS SANTOS COSTA

AUTOMAÇÃO DE TESTES: UMA PROPOSTA PARA PARAMETRIZAÇÃO DE TESTES UNITÁRIOS PARA APIS SPRING

CAMPINA GRANDE
2022

LUIS LANCELLOTE MARQUES DOS SANTOS COSTA

AUTOMAÇÃO DE TESTES: UMA PROPOSTA PARA PARAMETRIZAÇÃO DE TESTES UNITÁRIOS PARA APIS SPRING

Trabalho de Conclusão de Curso (Artigo) apresentado ao Curso de bacharelado em Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Orientador: Prof. Dr. Frederico Moreira Bublitz.

CAMPINA GRANDE
2022

C837a Costa, Luis Lancellote Marques dos Santos.
Automação de testes [manuscrito] : uma proposta para parametrização de testes unitários para APIs Spring / Luis Lancellote Marques dos Santos Costa. - 2022.
22 p.

Digitado.
Trabalho de Conclusão de Curso (Graduação em Computação) - Universidade Estadual da Paraíba, Centro de Ciências e Tecnologia , 2022.
"Orientação : Prof. Dr. Frederico Moreira Bublitz , Coordenação do Curso de Computação - CCT."

1. Testes unitários. 2. API Spring. 3. Desenvolvimento de software. I. Título

21. ed. CDD 005.1

Elaborada por Camile de A. Gomes - CRB - 15/559 BC/UEPB

LUIS LANCELLOTE MARQUES DOS SANTOS COSTA

AUTOMAÇÃO DE TESTES: UMA PROPOSTA PARA PARAMETRIZAÇÃO DE TESTES UNITÁRIOS PARA APIS SPRING

Trabalho de Conclusão de Curso (Artigo) apresentado ao Curso de bacharelado em Computação da Universidade Estadual da Paraíba, como requisito parcial à obtenção do título de bacharel em Computação.

Aprovada em: 21/ 07/ 2022.

BANCA EXAMINADORA

Prof. Dr. Frederico Moreira Bublitz (DC - UEPB)
Orientador(a)

Prof. Dr. Wellington Candeia de Araújo (DC - UEPB)
Examinador(a)

Dra. Karolina Celi Tavares Bezerra (Universidade do Minho - Portugal)
Examinador(a)

Aos meus pais Leonilsa e Magno, pela dedicação, companheirismo e amizade, DEDICO.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

AUTOMAÇÃO DE TESTES: UMA PROPOSTA PARA PARAMETRIZAÇÃO DE TESTES UNITÁRIOS PARA APIS SPRING

Luis Lancellote M. dos Santos Costa[1]

RESUMO

Considerando a grande importância da implementação de testes unitários, dentro do processo de desenvolvimento de software. A adoção de ferramentas ou processos que tornem a escrita de testes mais ágil se torna fundamental para que os desenvolvedores possam garantir a qualidade do software de forma mais eficiente. O objetivo deste trabalho é apresentar uma ferramenta capaz de automatizar a parametrização de testes unitários, reduzindo a repetição de código. A obtenção de dados necessários para a realização deste trabalho foi possível através da pesquisa e análise de livros e artigos contendo informações quanto à problemática abordada. Ao final, para a comprovação da eficácia da ferramenta apresentada, foi implementada uma API Spring e em seguida, apresentados os resultados obtidos com a escrita de cenários de testes com o auxílio da ferramenta proposta.

Palavras-chave: Testes unitários. API Spring. Desenvolvimento de software.

ABSTRACT

Considering the great importance of implementing unit tests, within the software development process. The adoption of tools or processes that make the writing of tests more agile becomes essential for developers to be able to guarantee the quality of the software more efficiently. The objective of this work is to present a tool capable of automating the parameterization of unit tests, reducing code repetition. Obtaining the necessary data to carry out this work was possible through research and analysis of books and articles containing information about the problem addressed. At the end, to prove the effectiveness of the tool presented, a Spring API was implemented and then the results obtained with the writing of test scenarios with the help of the proposed tool were presented.

Keywords: Unit tests. Spring API. Software development

1 INTRODUÇÃO

Those who can imagine anything, can create the impossible.
- Alan Turing

Com o desenvolvimento de software, a execução de testes se torna necessária para que sejam identificados eventuais erros provenientes da entrega de novos componentes da aplicação e permite que os problemas sejam resolvidos de forma menos custosa sem que ocorram prejuízos ao usuário final da aplicação.

[1] Luis Lancellote Marques dos Santos Costa, luislancelote@gmail.com.

Segundo Oliveira (2016) o desenvolvimento de software é um processo composto por etapas, e o teste de software é uma etapa que compõe esse processo. Para que possamos garantir a qualidade do software, é necessário que realizemos a etapa de testes.

A implementação de ferramentas ou processos que tornem a escrita de testes unitários mais ágil se torna fundamental para que os desenvolvedores possam garantir a qualidade do software de forma mais eficiente. Barbosa (2012) cita (CRISPIN; GREGORY, 2009):

> "A necessidade de maior qualidade e menor tempo para entrega do software impacta não somente as equipes de desenvolvimento e teste de software, mas também toda a disciplina de engenharia de software, neste aspecto, pode-se constatar as mudanças de paradigmas e de metodologias de desenvolvimento que vem acontecendo nos últimos anos. "

Com isso, a automação de testes pode ser vista como um dos principais recursos para melhorar a eficiência da execução de testes em uma aplicação.

Atualmente são utilizadas ferramentas como JUnit e Mockito, para criação de suítes de testes unitários, facilitando a rastreabilidade de eventuais problemas nos componentes da aplicação. Porém a etapa de parametrização de testes unitários, por vezes se torna repetitiva em uma aplicação de grande escala ao utilizar um *framework* como o Spring.

Visando simplificar a escrita de testes unitários para APIs desenvolvida com o framework Spring, o artigo traz como solução uma ferramenta capaz de simplificar a implementação de cenários de testes unitários por meio da automatização da parametrização de atributos em entidades da aplicação.

Como resultado, esperamos obter cobertura de 100% nos cenários de testes parametrizáveis em uma suíte de testes implementada com auxílio da ferramenta apresentada neste artigo.

Este capítulo apresenta o contexto no qual este trabalho se insere, as motivações para a sua realização e seus objetivos.

No Capítulo 2 são apresentadas as tecnologias adotadas: Spring Boot, JUnit 5 e o Mockito.

No Capítulo 3 descreve os trabalhos relacionados a este e como os mesmos contribuíram para o desenvolvimento deste artigo.

No Capítulo 4 é descrita a metodologia utilizada.

No Capítulo 5 será apresentada a proposta deste trabalho, bem como os requisitos funcionais e não-funcionais.

No Capítulo 6 serão apresentados os resultados da aplicação da ferramenta implementada.

Finalmente no Capítulo 7 são apresentadas a conclusão e perspectivas de trabalhos futuros.

2 TECNOLOGIAS UTILIZADAS

Neste capítulo serão apresentadas as ferramentas utilizadas no desenvolvimento deste trabalho. Serão abordados os conceitos sobre o Spring Boot, JUnit 5 e o Mockito.

O Spring Boot foi utilizado para implementação de uma API para demonstração da eficácia da ferramenta proposta. Já a Suíte de testes unitários foi

implementada com auxílio do JUnit 5 e por fim foi utilizado o Mockito com o objetivo de isolar os componentes da aplicação na suíte de testes implementada.

2.1 Spring Boot

O Spring é um framework Java que foi criado com o objetivo de disponibilizar para o programador diversas ferramentas para simplificar os processos de implementação de APIs e comunicação com banco de dados. O Spring é constituído por alguns módulos e um deles é o Spring boot, que visa facilitar o processo de configuração e publicação de aplicações.

Santos (2020) cita WEISSMANN (2015) que diz que o Spring se baseia em quatro princípios:

- Prover uma experiência de início de projeto (*getting started experience*) extremamente rápida e direta;
- Apresentar uma visão bastante opinativa sobre o modo como devemos configurar nossos projetos Spring e, ao mesmo tempo é flexível o suficiente para que possa ser facilmente substituída de acordo com os requisitos do projeto;
- Fornece uma série de requisitos não funcionais já pré-configurados para o desenvolvedor como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado, etc;
- Não provê nenhuma geração de código e minimiza a zero a necessidade de arquivos XML.

2.2 JUnit 5

Segundo Garcia (2017) o JUnit é um *framework open source* que permite a criação e execução de testes automatizados. Ele é um dos mais influentes na engenharia de software em geral. Ele foi lançado em 10 de setembro de 2017. É possível executar testes implementados com JUnit utilizando ferramentas de *builds*[2] populares como Gradle e Maven, e também com uso de IDEs como Intellij e Eclipse.

Ainda segundo Garcia (2017), a arquitetura do JUnit 5 é completamente modular, permitindo que os desenvolvedores usem as partes específicas do quadro de que necessitam. O JUnit 5 deve fornecer também a integração com estruturas de terceiros, como Spring ou Mockito. Dentre as principais funcionalidades do JUnit, podemos citar:

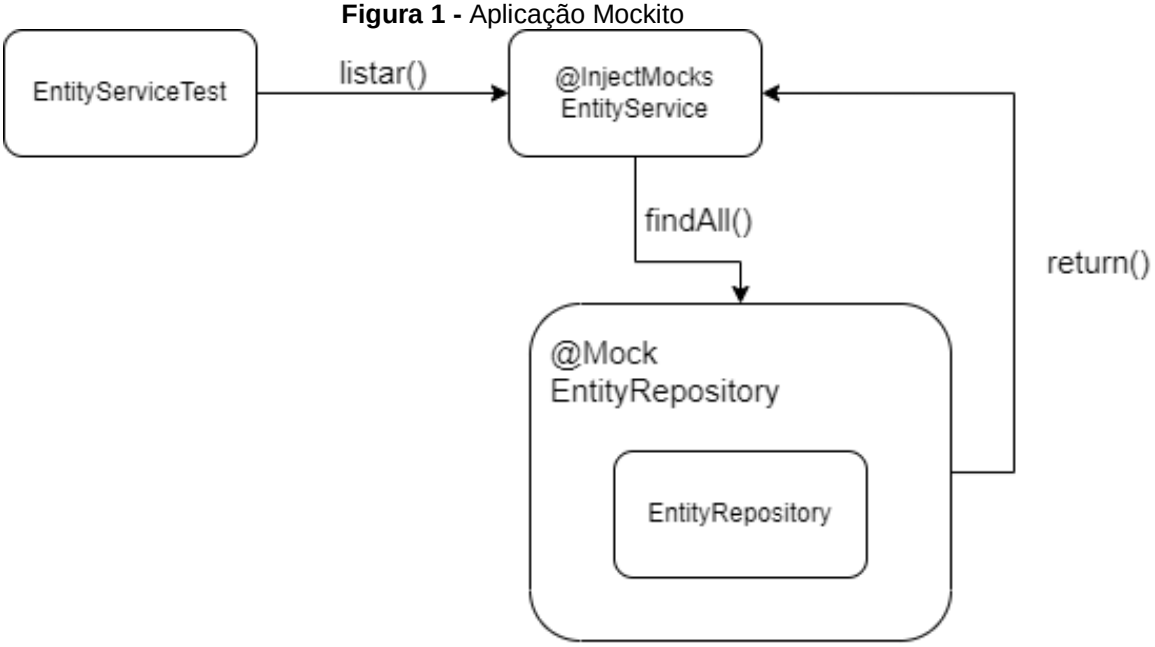
- Segregação de API: dissociar a descoberta e a execução da definição de teste;
- Compatibilidade com versões mais antigas: suporta a execução com Java 3 e Java 4 na nova plataforma JUnit 5;
- Modelo de programação moderno para escrever testes (Java 8): Hoje em dia, os desenvolvedores escrevem código com os novos recursos do Java 8, como as expressões lambda.

[2] É o processo de transformar arquivos de texto de código-fonte em um ou mais arquivos que são usados quando o aplicativo é executado.

2.3 Mockito

O Mockito é um *framework open source* que permite isolar componentes da aplicação utilizados em testes unitários, simulando seu comportamento diante de determinadas parametrizações. Ao realizar o *mock*[3] de uma dependência de uma classe, é possível simular o comportamento da dependência de acordo com o cenário de teste aplicado. Na figura 1, segue a aplicação do mockito nos testes realizados:

Figura 1 - Aplicação Mockito



Fonte: Elaborada pelo autor, 2022

Garcia (2017) descreve ainda que o Mockito fornece uma API para isolar o sistema sob testes (SUT) e seus componentes necessários para execução dos testes (DOCS). De um modo geral, usar Mockito envolve três etapas diferentes:

1. Objetos simulados: Para isolar o SUT, usamos a API Mockito para criar simulações de seus DOCs associados. Desta forma, garantimos que o SUT não depende de seus DOCs, e nosso teste de unidade é na verdade focado no SUT.
2. Definição de expectativas: O diferencial do objeto mock é que os objetos simulados podem ser programados com expectativas personalizadas de acordo com as necessidades da unidade teste.
3. Verificação: Mockito fornece uma API poderosa para realizar diferentes tipos de verificações. Com esta API, avaliamos as interações com o SUT e DOCs, verificando a ordem de invocação com um mock, ou capturando e verificando o argumento passado para um método.

[3] Simular o comportamento de objetos reais de forma controlada.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados trabalhos relacionados que também propõem softwares para auxiliar na automação de testes.

3.1 Google CodePro AnalytiX

CodePro Analytix é um conjunto abrangente de ferramentas de análise de software composto por uma coleção de plugins nativos do Eclipse. O CodePro se integra perfeitamente a qualquer ambiente de desenvolvimento de desktop Java baseado em Eclipse, adicionando auditoria de código, métricas, geração de teste, edição de teste JUnit, cobertura de código e recursos e funcionalidades de colaboração em equipe. (Rubel, Clayberg, 2008)

Ainda de acordo com os desenvolvedores da ferramenta (Rubel, Clayberg, 2008), os principais recursos do CodePro AnalytiX são:

- Detecção, reparo e relatório de defeitos
- Definir, distribuir e impor qualidade
- Padrões entre as equipes de desenvolvimento
- Análise de código estático (mais de 960 regras de auditoria)
- Análise de código duplicado
- Filtragem de auditoria avançada
- Relatórios de gerenciamento poderosos
- Métricas de código com detalhamento e gatilhos
- Auditoria para Java, JSP, JSF, Struts, Hibernate e XML
- Geração de teste JUnit automatizada
- Editor de teste JUnit
- Análise de cobertura de código
- Análise e relatórios de dependência
- Colaboração de equipe integrada
- Integração perfeita com Eclipse, Rational, ® WebSphere® e MyEclipse; suporta o Rational Application Developer v7.

3.2 Randoop

Randoop é uma ferramenta desenvolvida com o propósito de gerar testes unitários significativos de forma automática para classes, implementadas na linguagem Java, assim como também na plataforma .Net. A Randoop gera suas classes de teste utilizando uma técnica baseada em aleatoriedade e direcionamento por feedback (*feedback-directed random test generation*) (PACHECO; ERNST. 2007).

Ainda segundo (PACHECO. ERNST. 2007) o Randoop recebe como entrada um conjunto de classes em teste, um limite de tempo (em segundos) e, opcionalmente, um conjunto de “verificadores de contrato” que podem complementar aqueles já usados pelo Randoop de forma padrão. O Randoop gera dois conjuntos de testes: um contém testes de violação de contrato *(contract violating tests)*; e o segundo conjunto contém testes de regressão *(regression tests)* que não violam contratos, mas capturam um aspecto da implementação atual. O primeiro teste pode encontrar erros na implementação atual das classes em teste, enquanto o segundo pode encontrar erros em implementações futuras ou diferentes. Apesar da diferença

entre os testes realizados, ambos têm o propósito de achar erros no código que está sendo testado.

Randoop é executado em uma JVM Java 8 ou Java 11.

Em 31 de janeiro de 2022 foi lançado o manual do Randoop versão 4.3.0, (Zhang et. Al, 2022). De acordo com o manual, temos algumas formas típicas de usar o Randoop:

1. Se o Randoop gerar algum teste de revelação de erros, corrija os defeitos subjacentes, execute novamente o Randoop e repita até que o Randoop não gere nenhum teste de revelação de erros.
2. Adicione os testes de regressão ao conjunto de testes do seu projeto.
3. Execute os testes de regressão sempre que alterar seu projeto. Esses testes irão notificá-lo sobre mudanças no comportamento do seu programa.
4. Se algum teste falhar, minimize o caso de teste e investigue a falha.
    - Se uma falha de teste indicar que você introduziu um defeito de código, corrija o defeito.
    - Se uma falha de teste indicar que o teste foi excessivamente frágil ou específico (por exemplo, o valor de saída de um método foi alterado, mas o novo valor é tão aceitável quanto o valor antigo), desconsidere o teste.

3.3 Comparativo dos trabalhos relacionados

No quadro 1, podemos verificar os comparativos dos trabalhos relacionados mencionados anteriormente.

Quadro 1: Comparativo dos trabalhos relacionados

Critério	Ferramenta de parametrização automatizada	CodePro AnalytiX	Randoop
Utilizável em qualquer IDE	Sim	Não	Sim
Aplicável para aplicações Spring	Sim	Sim	Sim
Análise de código duplicado	Não	Sim	Não
Permite testes de Regressão	Sim	Sim	Sim

Fonte: Elaborado pelo autor, 2022

4 METODOLOGIA

A fim de facilitar a implementação de testes unitários para aplicações Spring, será proposta uma ferramenta capaz de automatizar o processo de parametrização em cenários de testes de *controller* para aplicações Spring.

Para a obtenção dos dados necessários para a realização do artigo, foram realizadas pesquisas em artigos e livros escritos em Português e Inglês, que continham informações relevantes quanto à problemática abordada.

Após o processo de pesquisa e análise, foi implementado um projeto piloto referente à uma API genérica além da escrita de testes unitários para que fosse demonstrada a viabilidade da ferramenta proposta. O conjunto de testes parametrizados foi obtido com o uso da própria ferramenta de parametrização automatizada em conjunto com uma suíte de testes implementada em JUnit.

5 PROPOSTA: AUTOMAÇÃO DE TESTES: UMA PROPOSTA PARA PARAMETRIZAÇÃO DE TESTES UNITÁRIOS PARA APIS SPRING

Para a criação da ferramenta, foram levados em consideração alguns elementos tais como a configuração do computador utilizado para a experimentação.

O computador utilizado para rodar os experimentos possui as seguintes configurações:

- Sistema Operacional: Windows 10 Pro 64 bits;
- Processador: Core i7 2.3 GHz;
- Memória instalada (RAM): 16 GB (utilizável 15.8 GB).
- Para a experimentação, foi utilizado a linguagem Java.
- Versão do JDK: 1.8. Versão do intellij: 2022.1.3

5.1 Modelagem da solução

Nesta seção será apresentada a modelagem da solução. Serão apresentados os requisitos funcionais e não funcionais para criação da ferramenta.

5.1.1 Requisitos não funcionais

- A ferramenta deverá ser desenvolvida utilizando a linguagem Java;
- A ferramenta deverá atender os cenários de teste para métodos HTTP que utilizam validação no corpo da requisição no *controller* de aplicações Spring;
- A IDE Intellij deve ser utilizada como ambiente de desenvolvimento da ferramenta;

5.1.2 Requisitos funcionais

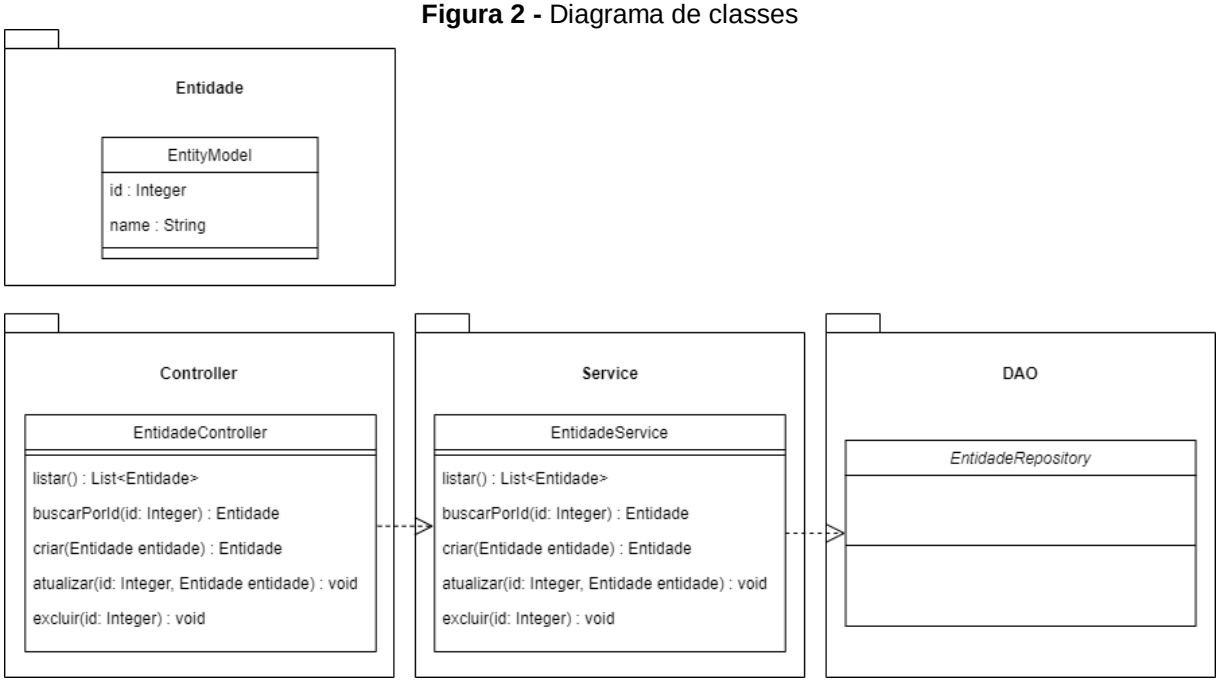
- A ferramenta deve identificar atributos com anotações de validação nas classes referentes às entidades da aplicação;
- A ferramenta deve disponibilizar opções para identificação dos cenários de testes;
- A ferramenta deve automatizar a parametrização dos atributos identificados com anotações de validação para os cenários de teste;

5.1.3 Diagrama de classe

Buscando um melhor entendimento sobre a estrutura, foi modelado o diagrama de classes para a proposta de automação de testes. O diagrama foi obtido

como representação das classes e seus relacionamentos no projeto como pode ser visto na Figura 2:

Figura 2 - Diagrama de classes



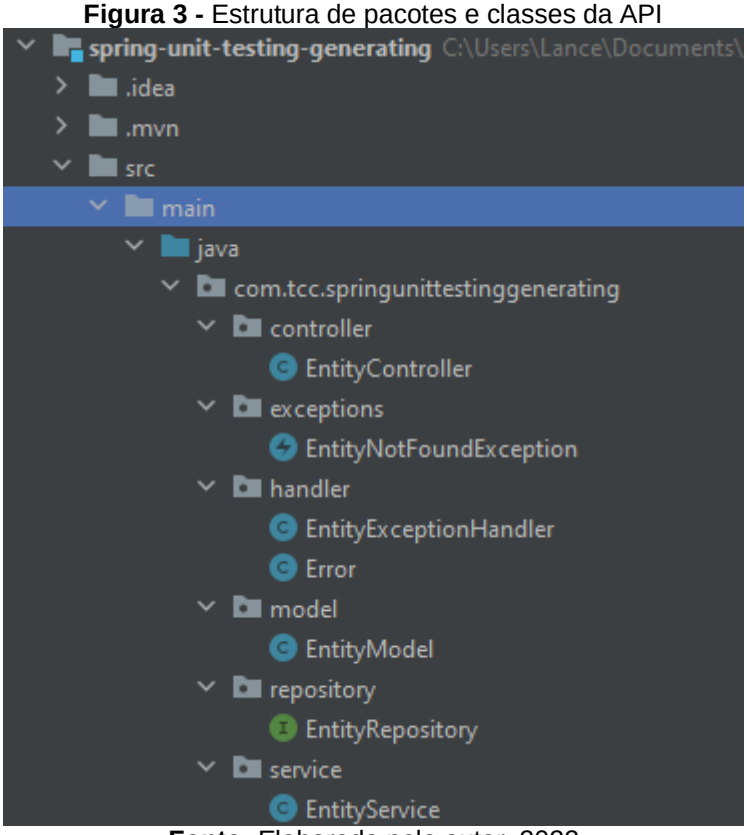
Fonte: Elaborada pelo autor, 2022

5.2 Desenvolvimento

Nesta seção serão mostrados a estrutura de pacotes e classes da API genérica (Figura 3), e a arquitetura de organização em camadas, onde:

- O pacote *model* corresponde a camada de modelo da aplicação;
- O pacote *repository* corresponde a camada de repositório da aplicação;
- O pacote *service* corresponde a camada de serviço da aplicação;
- O pacote *controller* corresponde a camada de controle da aplicação;
- O pacote *exceptions* corresponde a camada de exceções da aplicação;
- O pacote *handler* corresponde a camada de tratativas para *exceptions* da aplicação;

Figura 3 - Estrutura de pacotes e classes da API



Fonte: Elaborada pelo autor, 2022

No modelo implementado na classe "*EntityModel*", foi utilizada a anotação "*@Size*" com parâmetros de validação para tamanhos (mínimo e máximo), permitidos para a variável "*name*".

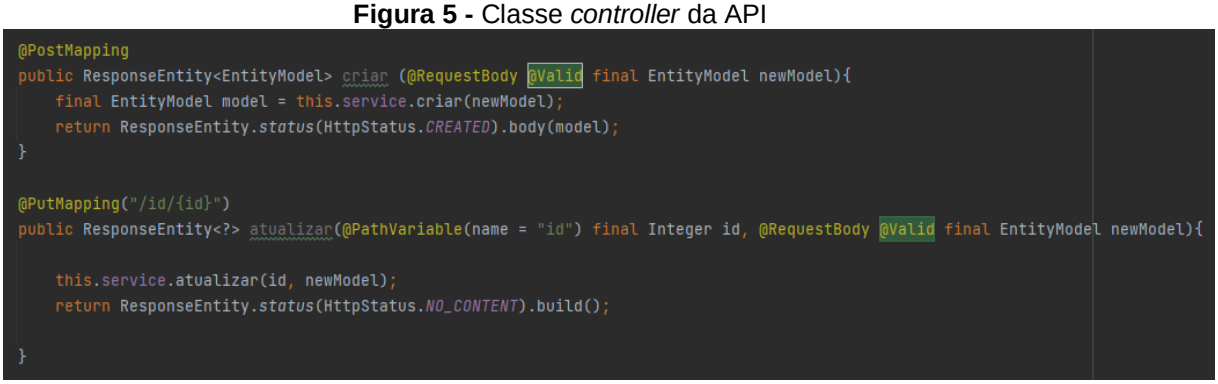
Figura 4 - Classe *model* da API

```
public class EntityModel {

    @Id
    private Integer id;

    @Size(min = 5, max = 50)
    @Column(name = "name")
    private String name;
```

Fonte: Elaborada pelo autor, 2022

No *controller* implementado na classe "*EntityController*", foi utilizada a anotação "*@Valid*" nos parâmetros referentes ao corpo das requisições, para que sejam habilitadas as validações definidas nas respectivas classes "*Model*".

Figura 5 - Classe *controller* da API



```
@PostMapping
public ResponseEntity<EntityModel> criar (@RequestBody @Valid final EntityModel newModel){
    final EntityModel model = this.service.criar(newModel);
    return ResponseEntity.status(HttpStatus.CREATED).body(model);
}

@PutMapping("/id/{id}")
public ResponseEntity<?> atualizar(@PathVariable(name = "id") final Integer id, @RequestBody @Valid final EntityModel newModel){

    this.service.atualizar(id, newModel);
    return ResponseEntity.status(HttpStatus.NO_CONTENT).build();

}
```

Fonte: Elaborada pelo autor, 2022

Os quadros a seguir demonstram os cenários de teste para os métodos criar e atualizar, com validação de tamanho para o campo *String*:

Quadro 2: Cenários de teste para o método criar

Tamanho do campo	Retorno da API
Vazio	400 BAD_REQUEST
Entre 5 e 50	201 CREATED
Menor que 5	400 BAD_REQUEST
Maior que 50	400 BAD_REQUEST

Fonte: Elaborado pelo autor, 2022

Quadro 3: Cenários de teste para o método atualizar

Tamanho do campo	Retorno da API
Vazio	400 BAD_REQUEST
Entre 5 e 50	204 NO_CONTENT
Menor que 5	400 BAD_REQUEST
Maior que 50	400 BAD_REQUEST

Fonte: Elaborado pelo autor, 2022

Para automatizar a parametrização de tais cenários, foi implementada uma ferramenta que recebe como parâmetros um objeto referente ao *Model* a ser parametrizado e um *enum* responsável por identificar o cenário a ser testado. As etapas de execução da ferramenta são:

1. Recebe um objeto genérico e uma constante referente ao cenário de teste a ser parametrizado;
2. Identifica os campos que possuem validação *Size*;
3. Identifica o valor mínimo ou máximo permitido;
4. Atribui o valor identificado em uma *String* auxiliar, para que esta seja incrementada ou decrementada dependendo do cenário testado;

5. Parametriza os campos com uma *String* genérica com base no cenário a ser testado;

A figura 6 representa o ENUM para as constantes representando os cenários de testes e respectivos incrementos e decrementos, nos quais a parametrização será realizada:

Figura 6 - Representação do ENUM

```
package com.tcc.springunittestinggenerating.utils.enums;

public enum BounderTestType {

    EMPTY( sizeIncrement: 0), UNDERFLOW( sizeIncrement: -1), MINIMUM_ALLOWED( sizeIncrement: 0), MAXIMUM_ALLOWED( sizeIncrement: 0),
    OVERFLOW( sizeIncrement: 1);

    private Integer sizeIncrement;

    BounderTestType(int sizeIncrement) {
        this.sizeIncrement = sizeIncrement;
    }

    public Integer getSizeIncrement() {
        return sizeIncrement;
    }

}
```

Fonte: Elaborada pelo autor, 2022

6 RESULTADOS

A seguir, serão apresentadas simulações referentes à execução de testes unitários com a utilização da ferramenta de parametrização automatizada. A figura 7, ilustra a suíte de testes:

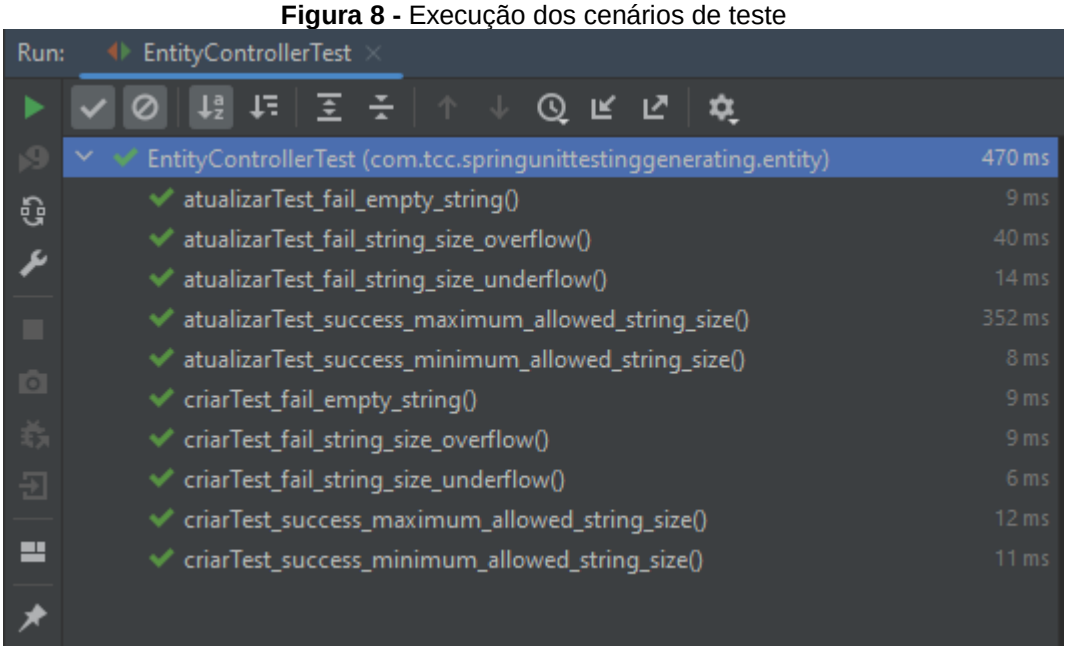
Figura 7 - Suíte de testes

```
@Test
public void criarTest_success_minimum_allowed_string_size() {...}

@Test
public void criarTest_success_maximum_allowed_string_size() {...}

@Test
public void criarTest_fail_empty_string() {...}

@Test
public void criarTest_fail_string_size_underflow() {...}

@Test
public void criarTest_fail_string_size_overflow() {...}

@Test
public void atualizarTest_success_minimum_allowed_string_size() {...}

@Test
public void atualizarTest_success_maximum_allowed_string_size() {...}

@Test
public void atualizarTest_fail_empty_string() {...}

@Test
public void atualizarTest_fail_string_size_underflow() {...}

@Test
public void atualizarTest_fail_string_size_overflow() {...}
```

Fonte: Elaborada pelo autor, 2022

A figura 8 apresenta o resultado da execução dos cenários de teste conforme esperado:

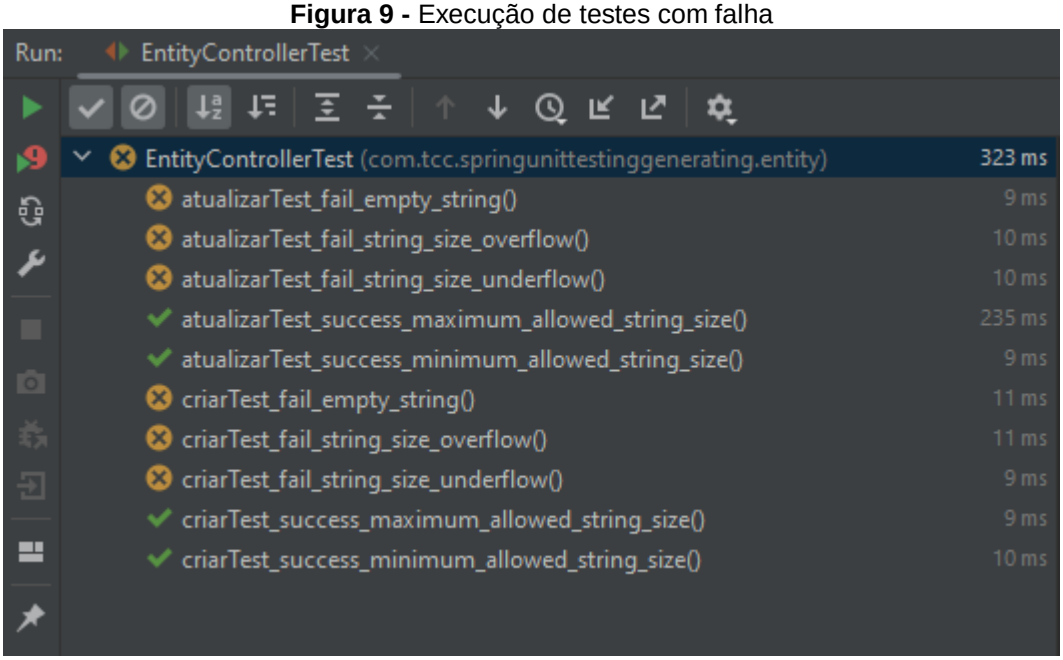
Figura 8 - Execução dos cenários de teste



Fonte: Elaborada pelo autor, 2022

Já na figura 9, podemos notar que ao retirar a anotação *@Size* no *Model* ou *@Valid* no *Controller*, a execução de testes falha nos cenários em que deveriam haver erros de validação, pois os cenários de testes para esses casos se tornam obsoletos:

Figura 9 - Execução de testes com falha



Fonte: Elaborada pelo autor, 2022

7 CONCLUSÃO

Considerando a importância da execução de testes unitários para a garantia de qualidade da aplicação, este trabalho apresentou uma ferramenta para auxiliar no processo de parametrização de testes.

Até então, a ferramenta é capaz de realizar a parametrização de cenários de teste para validação de tamanho de objetos do tipo *String*, abrindo possibilidades para vários domínios diferentes de trabalhos futuros, que possam abranger mais cenários de testes para as demais validações de *Model* em aplicações Spring.

Com isso, concluímos que em termos de eficácia, a ferramenta de parametrização desenvolvida neste trabalho, apresenta 100% de cobertura para os cenários de testes para os quais se destina.

REFERÊNCIAS

Automated Software Quality and Testing Tools - CodePro AnalytiX ™ for Eclipse, Rational® and MyEclipse: Disponível em: https://wiki.eclipse.org/images/7/75/CodeProDatasheet.pdf. Acesso em 21 de abril de 2022.

Barbosa, Andressa Garcia **Desenvolvimento de um Framework para Aplicação de Teste Unitário Orientado a Dados** / Andressa Garcia Barbosa. Marília, SP: [s.n.], 2012. 59f.

Ernst, Michael et all. **Manual Randoop.** Disponível em:
García, Boni. **Mastering Software Testing with JUnit 5 Comprehensive guide to-develop high quality Java applications,** 2017
https://randoop.github.io/randoop/manual/index.html#typical_use. Acesso em 21 de abril de 2022.

Oliveira, Daniel Gomes de. **Avaliação de Ferramentas de Geração Automática de Dados de Teste para Programas Java: Um Estudo Exploratório**, Dissertação (Pós-Graduação do Instituto de Informática) Universidade Federal de Goiás, Goiás. 2016.

Rojas, Jose Miguel et all. **A detailed investigation of the effectiveness of whole test suite generation.** Disponível em: https://www.evosuite.org/wp-content/papercite-data/pdf/emse16_effectiveness.pdf . Acesso em 21 de abril de 2022.

Santos, Caio Silva dos. Rastreador API [manuscrito] : **Uma ferramenta para rastreabilidade de agroalimentos** / Caio Silva dos Santos. - 2020. 56 p

Silva, Davi Augusto Gadêlha. **EvolUniT: Geração e Evolução de Testes de Unidade em Java utilizando Algoritmos Genéticos**, 2008 Disponível em: https://homes.cs.washington.edu/~mernst/pubs/pacheco-randoop-oopsla2007.pdf Acesso em 21 de abril de 2022.

WEISSMANN, Henrique Lobo. **Spring Boot: simplificando o Spring**. DevMedia, 2015. Disponível em: https://www.devmedia.com.br/spring-boot-simplificando-o-spring/31979. Acesso em: 04 de abril de 2020

APÊNDICE A - Código Fonte

```
public class StringBounderPopulator {

   public static void initializeSizeAnnotedStringFields(final
Object object, final BounderTestType bounderTestType) {

       final Class<?> objectClass = object.getClass();
       final Field[] objectFields =
objectClass.getDeclaredFields();
       Arrays.stream(objectFields).forEach(field -> {
           try {
               setStringValue(object, field, bounderTestType);
           } catch (IllegalAccessException e) {
               e.printStackTrace();
           }
       });
   }

   private static void setStringValue(final Object object,
final Field field, BounderTestType bounderTestType)
           throws IllegalAccessException {
       final boolean isSizeAnnotationPresent =
field.isAnnotationPresent(Size.class);
       if (isSizeAnnotationPresent){
           Integer stringSize = 0;
           char[] charFill = new char[stringSize];
           final Size sizeAnnotation =
field.getAnnotation(Size.class);
           if
(MINIMUM_ALLOWED.equals(bounderTestType)||UNDERFLOW.equals(bou
nderTestType)){
               stringSize += sizeAnnotation.min();
           } else if
(MAXIMUM_ALLOWED.equals(bounderTestType)||OVERFLOW.equals(boun
derTestType)){
               stringSize += sizeAnnotation.max();
           }
           stringSize += bounderTestType.getSizeIncrement();
           if (EMPTY.equals(bounderTestType)||stringSize > 0){
               charFill = new char[stringSize];
           }
           String stringFill = new String(charFill);
           field.setAccessible(true);
           field.set(object, stringFill);
       }
   }
}
```

APÊNDICE B - Código Fonte ENUM

```
public enum BounderTestType {

   EMPTY(0), UNDERFLOW(-1), MINIMUM_ALLOWED(0),
MAXIMUM_ALLOWED(0),
   OVERFLOW(1);

   private Integer sizeIncrement;

   BounderTestType(int sizeIncrement) {
       this.sizeIncrement = sizeIncrement;
   }

   public Integer getSizeIncrement() {
       return sizeIncrement;
   }

}
```

.

AGRADECIMENTOS

Aos meus pais, Magno (*in memoriam*) e Leonilsa, por todo amor e esforço para me proporcionar a melhor educação dentro do possível e motivação para lutar por minhas conquistas.

Aos meus irmãos Kaleo e Júnior, por todo apoio e companhia em todos os momentos.

Aos meus colegas de curso e trabalho Caio Lucena e Caio Santos, pelos momentos de reflexão e descontração.

Às minhas amigas Myrlla e Marcella pelos momentos felizes em meio a todas as dificuldades.

À Jana, minha namorada, por me apoiar e motivar em cada momento difícil e compartilhar os melhores momentos.

Ao meu orientador Fred, por todo apoio e direcionamento no decorrer da escrita deste trabalho.

Por fim, mas não menos importante, agradeço a Universidade Estadual da Paraíba, pela oportunidade de cursar o curso de Bacharelado em Ciência da Computação que já me abriu diversas portas na minha carreira profissional.